

EMAS @ AAMAS 2017

5th International Workshop on Engineering Multi-Agent Systems

Workshop held at AAMAS 2017

8-9 May 2017

Sao Paulo, Brazil

(Informal) Proceedings

Preface

This volume contains the papers to be presented at **EMAS 2017: 5th International Workshop on Engineering Multi-Agent Systems** held at AAMAS 2017, International Conference on Autonomous Agents and Multi-Agent Systems - May 8-9, 2017, Sao Paulo

EMAS aims to gather researchers and practitioners in the domains of Agent-Oriented Software Engineering (AOSE), PROgramming Multi-Agent Systems (ProMAS) and Declarative Agent Languages and Technologies (DALT) to present and discuss their research and outcomes in MAS engineering.

The purpose of this workshop is to facilitate the cross-fertilization of ideas and experiences in the various fields to: further our knowledge in MAS engineering and improve the state-of-the-art; define research directions that are useful to practitioners; and encourage practitioners to use established methodologies for the development of large-scale MAS.

EMAS 2017 attracted 18 submissions. Each submission was reviewed by 3 members of the Program Committee and finally 14 papers have been accepted to be presented during the workshop. A subset of this paper will be invited to be part of the post-proceedings of the workshop.

Acknowledgments

EMAS Organisers would like to thank the members of the Programme Committee for their work. Thanks to EasyChair guys for their great tool.

May 2017, Sao Paulo

Amal El Fallah-Seghrouchi (LIP6 -
University Pierre and Marie Curie,
Paris France)
Tran Cao Son (New Mexico State
University, Department of Computer
Science, USA)
Alessandro Ricci (DISI, University of
Bologna, Italy)

Program Committee

Natasha Alechina	University of Nottingham
Matteo Baldoni	Dipartimento di Informatica, Univ. di Torino
Luciano Baresi	DEIB - Politecnico di Milano
Cristina Baroglio	Dipartimento di Informatica, Università di Torino
Jeremy Baxter	QinetiQ
Olivier Boissier	ENS Mines Saint-Etienne
Rafael H. Bordini	PUCRS
Lars Braubach	University of Hamburg
Rem Collier	UCD
Massimo Cossentino	National Research Council of Italy
Fabiano Dalpiaz	Utrecht University
Mehdi Dastani	Utrecht University
Juergen Dix	Clausthal University of Technology
Amal El Fallah Seghrouchni	University Pierre and Marie Curie
Aditya Ghose	University of Wollongong
Adriana Giret	Technical University of Valencia
Jorge Gomez-Sanz	Universidad Complutense de Madrid
Sam Guinea	Politecnico di Milano
Christian Guttmann	Karolinska Institute, Sweden and University of New South Wales (UNSW), Australia
James Harland	RMIT University
Vincent Hilaire	UTBM/IRTES-SET
Koen Hindriks	Delft University of Technology
Tom Holvoet	Dept. of Computer Science, K.U.Leuven
Jomi Fred Hubner	Federal University of Santa Catarina
Michael Huhns	University of South Carolina
Franziska Klügl	Örebro University
Joao Leite	NOVA LINES, Universidade Nova de Lisboa
Yves Lespérance	York University
Brian Logan	University of Nottingham
Viviana Mascardi	DIBRIS (Department of Informatics, Bioengineering, Robotics and System Engineering), University of GENOVA, IT
Philippe Mathieu	University of Lille 1
John-Jules Meyer	Utrecht University
Frederic Migeon	IRIT
Ambra Molesini	Alma Mater Studiorum - Università di Bologna
Pavlos Moraitis	LIPADE, Paris Descartes University
Jörg P. Müller	TU Clausthal
Ingrid Nunes	Universidade Federal do Rio Grande do Sul (UFRGS)
Juan Pavón	Universidad Complutense de Madrid
Alexander Pokahr	University of Hamburg
Enrico Pontelli	New Mexico State University
Alessandro Ricci	University of Bologna
Ralph Rönquist	Realthing Entertainment Pty Ltd
Sebastian Sardina	RMIT University

Valeria Seidita	Dipartimento di Ingegneria Chimica Gestionale Informatica Meccanica-University of Palermo
Onn Shehory	Bar Ilan University
Viviane Silva	IBM Research
Guillermo Ricardo Simari	Dep of Computer Science and Engineering, Universidad Nacional del Sur in Bahia Blanca
Tran Cao Son	New Mexico State University
Pankaj Telang	SAS Institute Inc.
Wamberto Vasconcelos	Department of Computing Science, University of Aberdeen
Jørgen Villadsen	Technical University of Denmark
Gerhard Weiss	University Maastricht
Michael Winikoff	University of Otago
Wayne Wobcke	University of New South Wales
Pinar Yolum	Bogazici University
Neil Yorke-Smith	American University of Beirut

Additional Reviewers

Ahlbrecht, Tobias
Fiekas, Niklas
Spanoudakis, Nikolaos

Table of Contents

Towards Trusting Autonomous Systems	1
<i>Michael Winikoff</i>	
Approaching Interactions in Agent-Based Modelling with an Affordance Perspective	17
<i>Franziska Klügl and Sabine Timpf</i>	
Augmented Agents: Contextual Perception and Planning for BDI architectures	33
<i>Arthur Casals, Amal El Fallah Seghrouchni and Anarosa Alves Franco Brandão</i>	
PLACE: Planning based Language for Agents and Computational Environments	49
<i>Muhammad Adnan Hashmi, Muhammad Usman Akram and Amal El Fallah Seghrouchni</i>	
Argumentation Schemes in Multi-Agent Systems: A Social Perspective	65
<i>Alison R. Panisson and Rafael H. Bordini</i>	
A Decentralised Approach to Task Allocation Using Blockchain	81
<i>Tulio L. Basegio, Regio A. Michelin, Avelino F. Zorzo and Rafael H. Bordini</i>	
A Case Study of the Development of an Agent-based Simulation in the Traffic Signal Control Domain using an MDD Approach	97
<i>Fernando Santos, Ingrid Nunes and Ana L. C. Bazzan</i>	
Towards a Multi-Agent System Product Line Engineering Approach	113
<i>Downia Boufedji, Zahia Guessoum, Anarosa Brandao, Tewfik Ziadi and Aicha Mokhtari</i>	
Towards the Engineering of Agent-Based Augmented Worlds	129
<i>Angelo Croatti and Alessandro Ricci</i>	
Distributed Transparency in Endogenous Environments: the JaCaMo Case	142
<i>Héctor Xavier Limón Riaño, Alejandro Guerra-Hernández and Alessandro Ricci</i>	
Give Agents Some REST: A Resource-Oriented Abstraction Layer for Internet-scale Agent Environments	158
<i>Andrei Ciortea, Olivier Boissier, Antoine Zimmermann and Adina Magda Florea</i>	
Modeling for openness in MAS	174
<i>Daniela Maria Uez and Jomi Fred Hubner</i>	
Two Concepts of Module, for Agent Societies and Inter-Societal Agent Systems	190
<i>Antonio Carlos Rocha Costa</i>	
An Automated Approach to Manage MAS-Product Line Methods	206
<i>Sara Casare, Tewfik Ziadi, Anarosa Alves Franco Brandão and Zahia Guessoum</i>	

Towards Trusting Autonomous Systems

Michael Winikoff

Department of Information Science
University of Otago
Dunedin, New Zealand
Tel.: +64 3 479 8386, Fax: +64 3 479 8311
michael.winikoff@otago.ac.nz

Abstract. Autonomous systems are rapidly transitioning from labs into our lives. A crucial question concerns trust: in what situations will we (appropriately) trust such systems? This paper proposes three necessary prerequisites for trust. The three prerequisites are defined, motivated, and related to each other. We then consider how to realise the prerequisites. This paper aims to articulate a research agenda, and although it provides suggestions for approaches to take and directions for future work, it contains more questions than answers.

1 Introduction

The past few years have witnessed the rapid emergence of autonomous systems in our lives. Whether in the form of self-driving cars on the road, Unmanned Aerial Vehicles (UAVs) in the skies, or other, less media-grabbing forms, autonomous systems have recently been transitioning from labs and into our lives at a rapid pace.

A crucial question that needs to be answered before deploying autonomous systems is that of *trust*: to what extent are we comfortable with trusting software to make decisions, and to act on these decisions, without intervening human approval?

This paper explores the question of trust of autonomous systems. Specifically, it seeks to answer the question:

In what situations will humans (appropriately) trust autonomous systems?

In other words, assume that we are dealing with a specific problem and its context, where the context includes such things as the potential consequences (safety, social, etc.) of the system's behaviour. We then seek to know what prerequisites must hold in order for people to be able to develop an appropriate level of trust in a given autonomous system that solves the specific problem. By "appropriate level of trust" we mean that a system that is worthy of being trusted becomes trusted, but a system that is not worthy of trust becomes untrusted.

We consider the question of trust from the viewpoint of individual people. We choose to adopt this lens, rather than, say, considering the viewpoint of society as a whole, for a number of reasons. Firstly, individual trust is crucial: the viewpoint and policies of a society are clearly based on the viewpoints of the individuals in the society¹. Secondly, individuals are more familiar to us, and hence easier to analyse. Finally,

¹ Although not all individual viewpoints receive equal prominence, which can lead to government policies being out of step with the desires of the population.

and most importantly, we can study individual humans through various experiments (e.g. surveys). This allows us to seek to answer the question of the prerequisites for trust using experimental methods (e.g. social science, marketing, psychology).

Before proceeding to explore the prerequisites for trust, we need to briefly clarify what this paper is *not* about, and indicate the assumptions that we are making. This paper is about trusting autonomous systems (i.e. systems empowered to make decisions and act on them). Although autonomous systems often use Artificial Intelligence (AI) techniques, they are not required to be intelligent in a general sense. Thus this paper is *not* about the issues associated with trusting human-level AI, nor is it about issues relating to hypothetical super-intelligence [30]. This paper is also not about the broader social consequences of deploying autonomous systems. For example, the impact of AI and automation on the patterns and nature of employment [43,42,4,12]. These are important issues, and they do affect the extent to which a society will allow autonomous systems to be deployed. However, they are out of scope for this paper, since they require social rather than technological solutions.

We make two assumptions. Firstly, we assume that we are dealing with systems where the use of autonomy is acceptable. There are some systems where human involvement in decision making is essential. For example, an autonomous system that handed down prison sentences instead of a human judge may not be socially acceptable. There is also a strong case for banning the development of autonomous weapons². We do note that cases where autonomy is unacceptable are not fixed, and may vary as trust develops. For instance, if it is shown that software systems are able to make more consistent and less biased decisions than human judges, then it may become acceptable to have autonomous software judges in some situations. Secondly, in this paper we do not consider systems that learn and change over time. Learning systems pose additional challenges, including the potential inadequacy of design-time verification, and dealing with emergent bias [5].

The sorts of systems that are within scope include autonomous UAVs, self-driving cars, robots (e.g. nursebots), and non-embodied decision making software such as personal agents and smart homes.

This paper is a “blue sky” paper in that it doesn’t provide research results. Instead, it seeks to pose challenges, and articulate a research agenda. The paper does provide some answers in the form of suggestions for how to proceed to address the challenges, but largely it provides questions, not answers.

1.1 Related work

Whilst there is considerable literature devoted to the fashionable question of trusting human-level or super-intelligent AI, there is considerably less literature devoted to the more mundane, but immediate, issue of trusting autonomous (but less intelligent) systems.

Fisher *et al.* [22] consider trust in driverless cars. Like us, they flag legal issues and the importance of verification. This paper differs from their work in considering legal factors in a broader context of *recourse* (where legal recourse is only one of a range of

² <http://futureoflife.org/open-letter-autonomous-weapons/>

options), and in considering additional factors relating to formal verification. We also posit that *explanation* is important to trust. On the other hand, they also consider human factors, such as driver attention, which are relevant for cars that have partial autonomy, where the human driver needs to be ready to take back control in certain situations.

Helle *et al.* [27] consider, more narrowly, challenges in testing autonomous systems. They also reach the conclusion that formal verification is required, and, like the earlier work of Fisher *et al.* [18,21,22], propose verifying the decision making process in isolation. However, they also highlight the need to do complete system testing to ensure that the system works in a real environment. Where extensive real-world testing is impractical, they highlight *virtual testing* (with simulations) as an approach that can help. Helle *et al.* also have other recommendations that concern testing, such as using models, testing early and continuously, and automating test generation.

A recent Harvard Business Review article [5] argued that “*Trust of AI systems will be earned over time, just as in any personal relationship. Put simply, we trust things that behave as we expect them to*”. The article went on to highlight two key requirements for trust: *bias*, and more generally *algorithmic accountability*, and *ethical systems*. They argue that for AI to be trusted, there need to be mechanisms for dealing with bias (detecting and mitigating). More relevant to this paper, they go on to argue that bias is a specific aspect of the broader issue of algorithmic accountability, and they argue that “*AI systems must be able to explain how and why they arrived at a particular conclusion so that a human can evaluate the system’s rationale*”. They further propose that this explanation should be in the form of an interactive dialogue. They also argue that AI systems should include explicit representation and rules that embody ethical reasoning (see Section 3).

Abbass *et al.* [1] discuss the relationship between trust and autonomy, considering high-level definitions of concepts such as trust. The paper does not provide clear answers to what is required for trust. Similarly, a meta-analysis of literature on factors affecting trust in human-robot interaction [25] found that the most important factors affecting trust related to the *performance* of the robot (e.g. behaviour, predictability, reliability). However, they did not provide a clear picture of which specific factors, and also noted that further work was required, since some factors were not adequately investigated in the literature.

Finally, there is also a body of work on computational mechanisms to make recommendations, and to manage reputation and trust between software agents (e.g. [35,33]). However, this work focuses on trust of autonomous systems *by other software*, rather than by humans. This makes it of limited relevance, since it does not consider the complex psychological and social factors that inform human trust. A human does not decide to trust a system using just a simple calculation based on the history and evidenced reliability of the system in question.

The remainder of this paper is structured as follows. Section 2 introduces, defines, and motivates the three prerequisites that we identify. Section 3 introduces a fourth element (representing human values and using them in a reasoning process), that we do not consider essential, but that supports the prerequisites. Section 4 and 5 discuss how to tackle the prerequisite of being able to explain decisions, and verification & validation, respectively. We conclude in Section 6.

2 Prerequisites to Trust

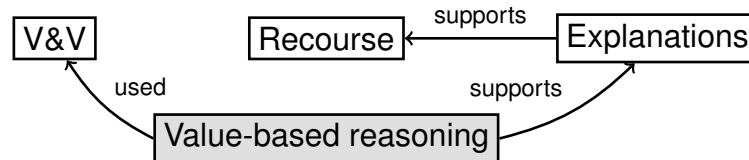
We propose that there are three required prerequisites to (appropriately) trusting an autonomous system:

1. a social framework that provides **recourse**, should the autonomous system make a decision that has negative consequences for a person;
2. the system’s ability to provide **explanations** of its behaviour, i.e. why it made a particular decision; and
3. **verification & validation of the system**, to provide assurance that the system satisfies key behavioural properties in all situations.

However, we do not claim that these three prerequisites are *sufficient*. We do argue that all three are *necessary*, but it may be that other prerequisites are also necessary. Identifying other prerequisites to trust is therefore an important part of answering the key question posed in Section 1.

A key message of this paper is that answering the key question requires a broad programme of research that spans technological sub-questions (e.g. formal verification, explanation) as well as social science sub-questions (e.g. when would humans trust autonomous software, what sort of explanations are helpful), and psychological sub-questions (e.g. how is trust affected by anthropomorphism, and how do characteristics of software affect the extent to which it is ascribed human characteristics).

The remainder of this section briefly outlines the three prerequisites. For each prerequisite we briefly define *what* it is, and motivate the need for that prerequisite (“*why*”). We also draw out the relationships between the three prerequisites (summarised in the diagram below). The subsequent sections consider for each prerequisite *how* that prerequisite might be addressed. Note that for recourse we only discuss “what” and “why” in this section, not “how” in a subsequent section. This is because the “how” is a social and legal question, and is out of scope for this paper. On the other hand, Section 3 discusses value-based reasoning, which is not an essential prerequisite (hence not in this section), but which can support both verification & validation, and explanations.



2.1 Recourse: law & social frameworks

We begin with the notion of *recourse*. In a sense, this prerequisite provides a safety net. We know that no person or system is perfect, and that even given a best possible set of practices in developing an autonomous system, it will have a non-zero rate of failure. The notion of recourse is that if an autonomous system does malfunction, that there is some way to be compensated for the negative consequences. We therefore argue that recourse is a necessary prerequisite to trust because it supports trusting a system that is less than 100% perfect, and in practice no system is 100% perfect.

Although the term “recourse” may suggest a mechanism where an affected individual uses the legal system to obtain compensation from another “person” (for autonomous systems, likely the corporation that developed the system), there are other possible social mechanisms that could be used, such as following an insurance model. For example, a form of “autonomous cars insurance” could cover people (pedestrians, cyclists, passengers, and other drivers) in the event that an autonomous vehicle malfunctioned in a way that caused harm. This insurance would ideally cover all people, and there are various models for universal insurance that could be used. For instance, New Zealand has a national comprehensive insurance scheme that automatically provides all residents and visitors with insurance for personal injury (www.acc.co.nz).

Being able to establish a justification for compensation, be it via legal proceedings or as some sort of insurance claim, would require that autonomous systems record enough information to permit audits to be undertaken, and the cause of harm identified. The ability of an autonomous system to explain why it made a decision can therefore support the process of seeking recourse by providing (part of) the evidence for harm.

While the existence of a recourse mechanism is identified as a prerequisite for trust of autonomous systems, this area is not a focus of this paper, and we do not discuss it further.

2.2 Explanations

A second prerequisite that we argue is essential to (appropriate) trust is the ability of an autonomous system to explain why it made a decision. Specifically, given a particular decision that has been made, the system is able to be queried, and provide to a human user an explanation for why it made that decision. The explanation needs to be in a form that is comprehensible and accessible, and may be interactive (i.e. take the form of a dialogue, rather than a single query followed by a complex answer).

There is a range of work, conducted in the setting of expert systems, rather than autonomous systems, that considers what is required for experts to trust systems. This work highlights explanation as an important factor in trust. For example, Teach & Shortliffe [41] considered attitudes of physicians (medical practitioners) towards decision support systems, including exploring the functionality and features that such systems would require in order to be acceptable to physicians. They noted (all emphasis is in the original) that

“An ability of a system to explain its advice was thought to be its most important attribute. Second in importance was the ability of a system to understand and update its own knowledge base. . . . Physicians did not think that a system has to display either perfect diagnostic accuracy or perfect treatment planning to be acceptable” (page 550)

They go on to recommend (p556) that researchers should:

“Concentrate some of the research effort on *enhancing the interactive capabilities* of the expert system. The more natural these capabilities, the more likely that the system will be used. At least four features appear to be highly desirable:

- (a) *Explanation*. The system should be able to justify its advice in terms that are understandable and persuasive. . . .
- (b) *Common sense*. The system should “seem reasonable” as it progresses through a problem-solving session. Some researchers argue that the operation of the program should therefore parallel the physician’s reasoning processes as much as possible. . . .
- (c) *Knowledge representation*. The knowledge in the system should be easy to bring up to date, . . .
- (d) *Useability* [sic] . . .”

they also recommend (p557) that researchers

“Recognize that *100% accuracy is neither achievable nor expected*. Physicians will accept a system that functions at the same level as a human expert so long as the interactive capabilities noted above are a component of the consultative process.”

In other words, the system always being right was seen by physicians as being less important, whereas the system being able to be understood was more important.

Stormont [40] considers trust of autonomous systems in hazardous environments (e.g. disaster zone rescue). He notes that while reliability is important, “*a more important reason for lacking confidence may be the unpredictability of autonomous systems.*” [40, Page 29]. In other words, autonomous software can sometimes do unexpected things. This can be a good thing: in some cases a software system may be able to find a good solution that is not obvious to a human. We argue that this supports the need for explanations: if a system is able to behave in a way that doesn’t obviously make sense to a human, but is nonetheless correct, then in order for the system to be appropriately trusted, it needs to be able to explain why it made its decisions. These explanations allow humans to understand and learn to trust a system that performs well. A difference between Stormont and Teach & Shortliffe is that the latter argue for the system to mirror human decision-making in order to be comprehensible (point (b) quoted above), whereas Stormont sees the benefit of allowing software to find solutions that may not be obvious to humans.

As noted earlier, providing explanations can support the process of building a case for compensation. The provision of explanations can benefit from using value-based reasoning (see Section 3).

2.3 Verification & Validation (V&V)

“It is possible to develop systems having high levels of autonomy, but it is the lack of suitable V&V methods that prevents all but relatively low levels of autonomy from being certified for use.” [14, Page ix]

Before deploying any software system, we need to have confidence that the system will function correctly. The strength of the confidence required depends on the consequences of the system malfunctioning. For non-safety-critical software, this confidence

is obtained by software testing. However, autonomous systems can exhibit complex behaviour that makes it infeasible to obtain confidence in a system via testing [47,49]. This therefore necessitates the use of formal methods as part of the design process.

While there may be situations where humans are willing to trust their lives to systems that have not been adequately verified, we argue that this is a case of excessive, and inappropriate, trust. If a system can potentially make a decision that, knowingly, results in harm to a human, then we should have strong assurance that this either does not occur, or occurs only under particular conditions that are well understood, and considered acceptable. The need for confidence in a system's correct functioning, and, for autonomous systems, the need to use formal methods, has been well-recognised in the literature (e.g. [27,22,14,16]).

3 Value-Based Reasoning

We have argued that recourse, explanations, and V&V are prerequisites that are essential (but not necessarily sufficient) to having appropriate human trust in autonomous systems. In addition we now propose a fourth element: value-based reasoning. We do not consider value-based reasoning to be an essential prerequisite, but explain below why it may be desirable, and how it supports two of the prerequisites. As noted earlier, a recent HBR article [5] argued that ethics can, and should, be codified and used in reasoning. Similarly, van Riemsdijk *et al.* [37] had earlier argued that socially situated autonomous systems (e.g. personal assistants and smart homes) should represent and use norms to reason about situations where norms may conflict.

By *value-based reasoning* we mean that the autonomous system includes a representation for human values (e.g. not harming humans), and that it is able to conduct reasoning using these human values in order to make decisions, where relevant. One (widely discussed) example is the use of ethical reasoning in autonomous vehicles [6]. However, using human values in the reasoning process can be beneficial not just in life-and-death situations. Consider a system that takes care of an aged person, perhaps with dementia or Alzheimer's disease. There are situations where competing options may be resolved by considering human values, such as autonomy vs. safety, or privacy vs. health. Perhaps the elderly person wants to go for a walk (which is both healthy, and is aligned with their desire for autonomy), but for safety reasons they should not be permitted to leave the house alone. In this example, the system needs to decide whether to allow the person it is caring for to leave the house, and, if so, what other actions may need to be taken. The key point is that in different situations, different decisions make sense. For instance, if a person is at a high risk of becoming lost, then despite their desire for autonomy, and the health benefits of walking, they should either be prevented from leaving, or arrangements should be made for them to be accompanied.

Value-based reasoning can be used to support two of the prerequisites. Firstly, we conjecture that the existence of a computational model of relevant human values could be used as a basis for providing higher level, more human-oriented, explanations of decisions. Secondly, in some situations, having an explicit model of values (or, perhaps more specifically, ethics) would be required to be able to verify certain aspects of an autonomous system's behaviour, for instance that the system's reasoning and decisions

take certain ethical considerations into account. For example, a recent paper by Dennis *et al.* [19] proposes to use formal methods to show that an autonomous system behaves *ethically*, i.e. that it only selects a plan that violates an ethical principle when the other options are worse. For instance, a UAV may select a plan that involves colliding with airport hardware (violating a principle of not damaging property) only in a situation where the other plans involve worse violations (e.g. collision with people or manned aircraft).

In some situations doing value-based reasoning will not be feasible. For instance, in a real autonomous vehicle, the combination of unreliable and noisy sensor data, unreliable actuators, the inherent unpredictability of consequences (partly due to other parties acting concurrently), and the lack of time to reason, means that in all likelihood, an autonomous vehicle will not be able to make decisions using utilitarian ethical reasoning. On the other hand, there may be applications (e.g. military) where software being able to conduct ethical reasoning would be considered to be very important [2].

Key research questions to consider in order to achieve value-based reasoning are:

- What values should be represented, and at what level of abstraction?
- How should reasoning about values be done, and in particular, how does this interact with the existing decision making process?
- How can values be utilised in providing explanations? And are such explanations more accessible to people than explanations that do not incorporate values?
- Given an agent with value-based reasoning, what sorts of verification can be done that makes use of the existence of values?

4 Explanations

As noted earlier, an important element in trust is being able to understand why a system made certain decisions, leading to its behaviour. Therefore, there is a need to develop mechanisms for an autonomous system to explain why it chose and enacted a particular course of action.

Since explanations can be complex (e.g. “I performed action a_1 because I was trying to achieve the sub-goal g_2 and I believed that b_3 held ...”), in order to be comprehensible, they need to be provided in a form that facilitates navigation of the explanation. This navigation can be in the form of a user interface that allows the explanation to be explored, or by having the explanations take the form of a dialog with the system (e.g. [13]).

Although there has been earlier work on explaining expert system recommendations, which may be useful as a source of ideas, the problem here is different in that we are explaining a *course of action* (taken over time, in an environment), not a (static) recommendation. Consequently, we are not dealing with deductive reasoning rules (as in expert systems), but with *practical* reasoning (although more likely to focus on means-end-reasoning than on deliberation, i.e. the focus is more likely to be on *achieving* rather than *selecting* goals).

Mechanisms for providing explanations obviously depend on the internal reasoning mechanism used and the representation of practical reasoning knowledge. For instance,

Broekens *et al.* [11] assume a representation in terms of a hierarchy of goals, also including beliefs and actions. If it turns out that explanations in terms of goals and beliefs are natural for humans to understand (which we might expect to be the case, since we naturally use “folk psychology” to reason about the behaviour of other humans), then that may imply that we want to have the autonomous system represent its knowledge in the form of plans to achieve its goals. However, it may also be possible to explain decisions made by a non-goal-based reasoning process, by using a separate representation in terms of goals. Although this would mean that the agent reasoning can use any mechanism and representation, it introduces the potential for the actual reasoning and the goal representation used for explaining to differ. Finally, it is also possible to provide explanations based solely on the observed behaviour of the system (i.e. without having an accessible or useful internal representation of the system’s decision making process), but this approach has drawbacks due to the limited information available [24].

There has been some work on mechanisms for autonomous systems to provide explanations (e.g. [11,26,50]), but more work is needed.

Harbers [26] assumes that there is a goal tree that captures the agent’s reasoning. The goal tree relates each goal to its sub-goals, and is indicated as being an “or” decomposition, “all” decomposition, “seq” (sequence) decomposition, or “if” decomposition. Each goal to sub-goal relationship is mediated by an optional belief that allows the sub-goal to be adopted (e.g. the sub-goal “prepare the fire extinction” is mediated by the belief “at incident location” [26, Figure 4.4]). The leaves of the tree are actions. The goal-tree is the basis for the implementation of the agent (using the 2APL agent programming language). A number of different explanation rules are considered. For instance, explaining an action in terms of its parent goal, or in terms of its grandparent goal, or in terms of beliefs that allowed the action to be performed, or in terms of the *next* action to be done (e.g. “I did action a_1 so I could then subsequently do action a_2 ”). Harbers reports on an experiment (with human subjects) using a simple fire fighting scenario, where the tree of goals contains 26 goals, and where the agent executes a sequence of 16 actions. The experiment aims to find out which explanation rules are preferred. She finds that in general there is not a consistent preference: for some actions a particular rule (e.g. the parent goal) is the commonly preferred explanation, whereas for other actions, the next action is the commonly preferred explanation. Harbers proposed that an action ought to be explained by the combination of its parent goal and the belief that allowed the action to be performed (which was not an explanation rule used in her experiment), but also defined two exceptional situations for which different explanations should be used. Broekens *et al.* [11] report on a similar experiment, and also find that there is not a single explanation rule that is the best for all situations.

One characteristic of the rules used by Harbers and by Broekens *et al.* is that they are (intentionally) incomplete: given an action, each rule selects only part of the full explanation. For instance, a rule that explains an action in terms of its parent goal ignores the beliefs that led to that goal being selected. By contrast, Hindriks [28] defines (informal) rules that yield a more complete explanation. More recently, Winikoff [50] builds on Hindriks’ work by systematically deriving formally-defined rules that are then implemented. Winikoff also explicitly defines (but does not prove) a completeness result: that, given their derivation, the rules capture *all* the explanatory factors. However, this

work aims to support programmers debugging a system, rather than human end-users trying to understand a system's behaviour (presumably without a detailed understanding of the system's internals!). Additionally, the completeness of the rules comes at a cost: the explanations are larger, and therefore harder to comprehend.

Finally, as mentioned in the previous section, it may be desirable to include human value-based reasoning into the decision process, which then poses the question of how to exploit this in the provision of explanations.

We therefore have the following research questions:

- How can an autonomous system provide explanations of its decisions and actions?
- What forms of explanation are most helpful and understandable? Is it helpful to structure explanations in terms of folk psychology constructs such as goals, plans and beliefs?
- How can explanations be effectively navigated by human users? In what circumstances is it beneficial to provide an explanation in the form of a dialogue?
- What reasoning processes and internal representations facilitate the provision of explanations? Does there need to be some representation of the system's goals?
- What is the tradeoff between using the same representation for both decision making and explanation, as opposed to using a different representation for explanation?
- How well can explanations be provided without a representation for the system's decision making knowledge and process (i.e. based solely on observing the system's behaviour)?
- How can explanations be provided that exploit the presence of representations of human values in the reasoning process? Are such explanations more accessible to people than explanations that do not incorporate human values?

Note that we are assuming a setting where a system deliberates and acts autonomously, and may be required to provide after-the-fact explanations (to help a human understand why it acted in certain ways, or to provide evidence for compensation, in the event of harm). However, another setting to be considered is where autonomous software works closely with humans, as part of a mixed team. In this sort of setting it is important not just to be able to explain after the fact, but also to provide updates during execution so that team members (both human and software) have sufficient awareness of what other team members are doing, or are intending to do. Doing this effectively is a challenge, since a balance needs to be struck between sharing too little (leading to inadequate awareness, and potential coordination issues) or too much (leading to overloading human team members with too much information). There has been some work that has explored this issue (e.g. [32,39]). However, this is not related to trusting autonomous systems in a general setting, but to the effectiveness of working with software in mixed human-agent teams.

5 Verification & Validation

We have already noted that we need to have a way of obtaining assurance that an autonomous software system will behave appropriately, and that obtaining this assurance

will require formal methods. We now consider the challenges involved in doing so, highlight some approaches, and pose research questions.

Work on techniques for verifying autonomous systems goes back at least 15 years (e.g. [52]). However, current state-of-the-art techniques are still only able to verify small systems [52,7,34,16,20,21,18]. Given the work that has been done, and the foundations provided by earlier work on verification of (non-autonomous) software, continuing to improve verification techniques is important future work, and eventually the techniques will be able to deal with realistically-sized systems. A number of ideas have been proposed that reduce the complexity of verification.

Firstly, Fisher *et al.* [18,22,21] have proposed to reduce the complexity of verifying autonomous systems by focussing on verifying the system’s decision making in isolation. The correct functioning of sensors and effectors is assessed separately, which requires end-to-end testing, possibly involving simulation [27]. Verifying decision making not only improves efficiency, but also allows verification to consider whether a bad decision is made in error (e.g. due to missing information), or intentionally, which is an important distinction [31,3].

Secondly, Bordini *et al.* [8] have proposed using slicing to reduce the complexity of verification. The basic idea is that given a particular property to be verified, instead of verifying the property against the agent program, one first generates a specialised version of the program that has been “sliced” to remove anything that does not affect the truth of the property being verified. The property is then verified against the “sliced” program. There is scope for further work, including considering other forms of program transformation prior to verification. For instance, there is a body of work on partial evaluation³ [29] that may be applicable.

Thirdly, there are various approaches that reduce the complexity of verifying a large system by verifying parts of the system separately, and then combining the verifications. One well-known approach uses assume-guarantee rules (e.g. [23]). It would be useful to consider adapting this approach for use with autonomous systems. In the case that the system’s decision making is represented in terms of a hierarchy of goals, it may be that sub-goals provide a natural point of modularity, i.e. that one can verify sub-goals in isolation, and then combine the results.

In addition to these research strands, which aim to make verification practical for real agent programs, there is another issue to consider: ***where does the formal specification come from?*** Verification takes a property and checks whether this property holds, but in order to be confident that a system (autonomous or not) will behave appropriately, we need to be confident that the collection of properties being verified adequately capture the requirements for “appropriate behaviour” [38].

In some cases there may be existing laws or guidelines that adequately specify what is “appropriate behaviour” for a given context, for instance, the Rules of the Air⁴ de-

³ Partial evaluation is the process of taking a program and some of its inputs and producing a specialised program that is able to accept the remaining inputs and compute the same results as the original program, but more efficiently.

⁴ <https://www.easa.europa.eu/document-library/regulations/commission-implementing-regulation-eu-no-9232012>

scribe how a pilot must behave in certain situations⁵, and can be used as a source for properties to be verified [46]. However, sometimes such guidelines do not exist, or they may be incomplete. For example, important constraints may not be explicitly stated, if they are “obvious” to humans, such as that a pilot should not accelerate in a way that exceeds human tolerances.

We therefore propose the development of a process for systematically deriving the properties to be verified from the system’s design and a collection of high-level generic properties (e.g. “cause no harm”, “always ensure others are aware of your intentions” - important for predictability). We assume that the autonomous software is developed using a well-defined methodology [51] which uses design models (e.g. goal model, interaction protocols) as “stepping stones” in the development process that results in software. The properties to be verified are derived by taking a collection of generic high-level properties which apply to any system, expressed in an appropriate notation, and applying a well-defined process for deriving a fault model [44] from the high-level properties and the system’s design models. We then need a well-defined process for deriving the required formal specification properties from the fault model.

For instance, given a high-level property of “not harming people”, one might examine the system’s design (along with information on its environment, and domain knowledge regarding the consequences of various actions) to derive a fault model that captures the specific ways in which the system’s decisions might lead to harming people. As an example, consider a robot assistant (“Care-O-bot”) [45] that resides in a home along with an elderly person being cared for. We would consider how harm to the person being cared for can occur in relation to the system’s requirements. Since the system is responsible for managing medication, we might identify that administering medication incorrectly, or failing to remind the person to take their medication, are possible ways in which harm can be caused. Similarly, the system failing to promptly seek help in the event of an accident, adverse medical event or other emergency (e.g. fire, earthquake) would be another way in which the person being cared for could be harmed. This analysis process *contextualises* the threats to the high-level properties in the circumstances of the system, and results in a fault model, which captures specific ways in which the system at hand might violate the high-level properties. We then need to have a way of deriving from the fault model specific properties to be verified, in an appropriate formal notation. The collection of high-level properties, process for deriving a fault model for a given system, and method for deriving formal properties from the fault model all need to be developed, along with appropriate notations.

Finally, as noted in the previous section, the internal reasoning process and associated representation matters. What sort of reasoning mechanisms and knowledge representations should be used to facilitate verification? Fisher *et al.* [21] have argued, in the context of verifying autonomous systems, that the systems should be developed in terms of beliefs, goals, plans, and actions, i.e. using a BDI (Belief Desire Intention) [36] agent-oriented programming language such as Gwendolyn⁶ [17].

⁵ For example, that when two planes are approaching head on and there is a danger of collision, that the pilots should both turn to their right.

⁶ Other prominent BDI agent-oriented programming languages include Jason [9], Jadex [10], JACK [48], and 2APL [15].

We therefore have the following research questions:

- How can agent program slicing be improved? What other forms of program transformation (e.g. partial evaluation) could be used to reduce the complexity of verification?
- Can the decision making process for a given autonomous system be verified in a modular way, perhaps using assume-guarantee reasoning (e.g. [23])? If so, can goals and sub-goals be used as a natural point to divide into independent components for verification?
- How can the properties to be verified be systematically derived?
- Should autonomous agents be programmed using a notation that supports representations for goals, beliefs, plans, and actions? If so, are existing BDI agent programming languages adequate, or do they need to be extended, restricted, or otherwise modified?

6 Discussion

In this paper we have considered the issue of *trust*, specifically posing the question: “*In what situations will humans (appropriately) trust autonomous systems?*”

We argued that there are three prerequisites that are essential in order for appropriate trust in autonomous systems to be realised: having assurance that the system’s behaviour is appropriate (obtained through verification & validation), having the system be able to explain and justify its decisions in a way that is understandable, and the existence of social frameworks that provide for compensation in the event that an autonomous system’s decisions do lead to harm (“recourse”). We also discussed using computational representations of human values as part of the decision making process in autonomous software, and how this can support the other prerequisites.

However, while we have argued that these three prerequisites are necessary, we are not in a position to claim that they are sufficient. Therefore, an overarching piece of research is to investigate experimentally the extent to which humans are willing to trust various autonomous systems given the prerequisites, and, especially, where people are not willing to trust a system, to identify what additional prerequisite might be required in order to enable (appropriate) trust.

We have discussed paths towards achieving the two technical prerequisites, and posed specific research questions, thereby defining a research agenda. There is much work to be done, and I hope that this paper will help to spur further discussion on what is needed to have appropriate trust in autonomous systems, and encourage researchers to work on the problems and questions articulated.

Acknowledgements

I would like to thank the anonymous reviewers for their comments, and Michael Fisher for discussions and pointers to literature.

References

1. Abbass, H.A., Petraki, E., Merrick, K., Harvey, J., Barlow, M.: Trusted autonomy and cognitive cyber symbiosis: Open challenges. *Cognitive Computation* 8(3), 385–408 (2016), doi:10.1007/s12559-015-9365-5
2. Arkin, R.C., Ulam, P., Wagner, A.R.: Moral decision making in autonomous systems: Enforcement, moral emotions, dignity, trust, and deception. *Proceedings of the IEEE* 100(3), 571–589 (March 2012), doi:10.1109/JPROC.2011.2173265
3. Atkinson, D.J., Clark, M.H.: Autonomous agents and human interpersonal trust: Can we engineer a human-machine social interface for trust? In: *Trust and Autonomous Systems: Papers from the 2013 AAAI Spring Symposium*. pp. 2–7 (2013)
4. Autor, D.H.: Why Are There Still So Many Jobs? The History and Future of Workplace Automation. *Journal of Economic Perspectives* 29(3), 3–30 (2015)
5. Banavar, G.: What It Will Take for Us to Trust AI. *Harvard Business Review* <https://hbr.org/2016/11/what-it-will-take-for-us-to-trust-ai> (November 2016)
6. Bonnefon, J.F., Shariff, A., Rahwan, I.: The social dilemma of autonomous vehicles. *Science* 352(6293), 1573–1576 (June 2016), doi:10.1126/science.aaf2654
7. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking AgentSpeak. In: *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 409–416. ACM Press (2003)
8. Bordini, R.H., Fisher, M., Wooldridge, M., Visser, W.: Property-based slicing for agent verification. *J. Log. Comput.* 19(6), 1385–1425 (2009), doi:10.1093/logcom/exp029
9. Bordini, R.H., Hübner, J.F., Wooldridge, M.: *Programming multi-agent systems in AgentSpeak using Jason*. Wiley (2007), ISBN 0470029005
10. Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: A BDI-Agent System Combining Middleware and Reasoning. In: Unland, R., Calisti, M., Klusch, M. (eds.) *Software Agent-Based Applications, Platforms and Development Kits*, pp. 143–168. Birkhäuser (2005), doi:10.1007/3-7643-7348-2_7
11. Broekens, J., Harbers, M., Hindriks, K.V., van den Bosch, K., Jonker, C.M., Meyer, J.C.: Do You Get It? User-Evaluated Explainable BDI Agents. In: Dix, J., Witteveen, C. (eds.) *Proceedings of the 8th German Conference on Multiagent System Technologies (MATES 2010)*. LNCS, vol. 6251, pp. 28–39. Springer (2010), doi:10.1007/978-3-642-16178-0_5
12. Brynjolfsson, E., McAfee, A.: *The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies*. W.W. Norton & Company (2014)
13. Caminada, M.W.A., Kutlák, R., Oren, N., Vasconcelos, W.W.: Scrutable plan enactment via argumentation and natural language generation (demonstration). In: Bazzan, A.L.C., Huhns, M.N., Lomuscio, A., Scerri, P. (eds.) *International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. pp. 1625–1626. IFAAMAS/ACM (2014), <http://dl.acm.org/citation.cfm?id=2616095>
14. Dahm, W.J.: *Technology horizons: A vision for air force science & technology during 2010-2030*. Tech. Rep. AF/ST-TR-10-01-PR, US Air Force (2010)
15. Dastani, M.: 2APL: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems* 16(3), 214–248 (2008)
16. Dastani, M., Hindriks, K.V., Meyer, J.J.C. (eds.): *Specification and Verification of Multi-agent systems*. Springer, Berlin/Heidelberg (2010)
17. Dennis, L.A., Farwer, B.: Gwendolen: A BDI Language for Verifiable Agents. In: Löwe, B. (ed.) *Logic and the Simulation of Interaction and Reasoning* (2008), AISB’08 Workshop
18. Dennis, L.A., Fisher, M., Lincoln, N.K., Lisitsa, A., Veres, S.M.: Practical verification of decision-making in agent-based autonomous systems. *Automated Software Engineering* 23(3), 305–359 (2016), doi:10.1007/s10515-014-0168-9

19. Dennis, L.A., Fisher, M., Slavkovik, M., Webster, M.: Formal verification of ethical choices in autonomous systems. *Robotics and Autonomous Systems* 77, 1–14 (2016), doi:10.1016/j.robot.2015.11.012
20. Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: Model checking agent programming languages. *Automated Software Engineering Journal* 19(1), 3–63 (March 2012), doi:10.1007/s10515-011-0088-x
21. Fisher, M., Dennis, L., Webster, M.: Verifying autonomous systems. *Commun. ACM* 56(9), 84–93 (Sep 2013), doi:10.1145/2494558
22. Fisher, M., Reed, N., Savirimuthu, J.: Misplaced Trust? In: *Engineering and Technology Reference*. The Institution of Engineering and Technology, doi:10.1049/etr.2014.0054 (2015)
23. Gheorghiu Bobaru, M., Păsăreanu, C.S., Giannakopoulou, D.: Automated Assume-Guarantee Reasoning by Abstraction Refinement. In: Gupta, A., Malik, S. (eds.) *Conference on Computer Aided Verification (CAV)*, pp. 135–148. Springer Berlin Heidelberg (2008), doi:10.1007/978-3-540-70545-1_14
24. Gomboc, D., Solomon, S., Core, M., Lane, H.C., van Lent, M.: Design Recommendations to Support Automated Explanation and Tutoring. In: *Conference on Behavior Representation in Modeling and Simulation (BRIMS)* (2005), <http://ict.usc.edu/pubs/Design%20Recommendations%20to%20Support%20Automated%20Explanation%20and%20Tutoring.pdf>
25. Hancock, P.A., Billings, D.R., Schaefer, K.E., Chen, J.Y.C., de Visser, E.J., Parasuraman, R.: A meta-analysis of factors affecting trust in human-robot interaction. *Human Factors* 53(5), 517–527 (2011), doi:10.1177/0018720811417254
26. Harbers, M.: *Explaining Agent Behavior in Virtual Training*. SIKS dissertation series no. 2011-35, SIKS (Dutch Research School for Information and Knowledge Systems) (2011)
27. Helle, P., Schamai, W., Strobel, C.: Testing of Autonomous Systems - Challenges and Current State-of-the-Art. In: *26th Annual INCOSE International Symposium* (2016)
28. Hindriks, K.V.: Debugging is explaining. In: Rahwan, I., Wobcke, W., Sen, S., Sugawara, T. (eds.) *PRIMA 2012: Principles and Practice of Multi-Agent Systems*. LNCS, vol. 7455, pp. 31–45. Springer (2012), doi:10.1007/978-3-642-32729-2_3
29. Jones, N.D.: An introduction to partial evaluation. *ACM Comput. Surv.* 28(3), 480–503 (1996), doi:10.1145/243439.243447
30. Kaplan, J.: Artificial Intelligence: Think Again. *Communications of the ACM* 60(1), 36–38 (January 2017), doi:10.1145/2950039
31. Lee, J.D., See, K.A.: Trust in automation: Designing for appropriate reliance. *Human Factors* 46(1), 50–80 (Spring 2004)
32. Li, S., Sun, W., Miller, T.: Communication in Human-Agent Teams for Tasks with Joint Action. In: Dignum, V., Noriega, P., Sensoy, M., Sichman, J.S. (eds.) *Coordination, Organizations, Institutions, and Norms in Agent Systems XI (COIN@AAMAS and COIN@IJCAI post proceedings)*, pp. 224–241. Springer International Publishing, Cham (2016), doi:10.1007/978-3-319-42691-4_13
33. Pinyol, I., Sabater-Mir, J.: Computational trust and reputation models for open multi-agent systems: a review. *Artificial Intelligence Review* 40(1), 1–25 (2013), doi:10.1007/s10462-011-9277-z
34. Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. *J. Applied Logic* 5(2), 235–251 (2007)
35. Ramchurn, S.D., Huynh, D., Jennings, N.R.: Trust in multi-agent systems. *The Knowledge Engineering Review* 19(1), 1–25 (2004), doi:10.1017/S0269888904000116
36. Rao, A.S., Georgeff, M.P.: BDI agents: From theory to practice. In: Lesser, V.R., Gasser, L. (eds.) *Conference on Multiagent Systems*. pp. 312–319. The MIT Press (1995)

37. van Riemsdijk, M.B., Jonker, C.M., Lesser, V.R.: Creating socially adaptive electronic partners: Interaction, reasoning and ethical challenges. In: Weiss, G., Yolum, P., Bordini, R.H., Elkind, E. (eds.) *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. pp. 1201–1206. ACM (2015), <http://dl.acm.org/citation.cfm?id=2773303>
38. Rozier, K.Y.: Specification: The biggest bottleneck in formal methods and autonomy. In: Blazy, S., Chechik, M. (eds.) *Verified Software. Theories, Tools, and Experiments (VSTTE)*, Revised Selected Papers. LNCS, vol. 9971, pp. 8–26 (2016), doi:10.1007/978-3-319-48869-1_2
39. Singh, R., Sonenberg, L., Miller, T.: Communication and shared mental models for teams performing interdependent tasks. In: Osman, N., Sierra, C. (eds.) *Autonomous Agents and Multiagent Systems: AAMAS 2016 Workshops, Best Papers*, pp. 163–179. Springer International Publishing (2016), doi:10.1007/978-3-319-46882-2_10
40. Stormont, D.P.: Analyzing Human Trust of Autonomous Systems in Hazardous Environments. In: Metzler, T. (ed.) *AAAI Workshop on Human Implications of Human-Robot Interaction*, <http://www.aaai.org/Library/Workshops/ws08-05.php>. pp. 27–32. The AAAI Press, Technical Report WS-08-05 (2008)
41. Teach, R.L., Shortliffe, E.H.: An Analysis of Physician Attitudes Regarding Computer-Based Clinical Consultation Systems. *Computers and Biomedical Research* 14, 542–558 (1981)
42. The White House: Artificial Intelligence, Automation, and the Economy. <https://www.whitehouse.gov/sites/whitehouse.gov/files/documents/Artificial-Intelligence-Automation-Economy.PDF> (December 2016)
43. The White House: Preparing for the Future of Artificial Intelligence. https://www.whitehouse.gov/sites/default/files/whitehouse_files/microsites/ostp/NSTC/preparing_for_the_future_of_ai.pdf (October 2016)
44. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault tree handbook. Tech. Rep. NUREG-0492, US Nuclear Regulatory Commission (January 1981)
45. Webster, M., Dixon, C., Fisher, M., Salem, M., Saunders, J., Koay, K.L., Dautenhahn, K., Saez-Pons, J.: Towards reliable autonomous robotic assistants through formal verification: A case study. *IEEE Transactions on Human-Machine Systems* PP(99), 1–11 (2015)
46. Webster, M., Cameron, N., Fisher, M., Jump, M.: Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. *Journal of Aerospace Information Systems* 11(5), 258–279 (2014), doi:10.2514/1.1010096
47. Winikoff, M., Cranefield, S.: On the Testability of BDI Agent Systems. *Journal of Artificial Intelligence Research (JAIR)* 51, 71–131 (2014), doi:10.1613/jair.4458
48. Winikoff, M.: JACKTM Intelligent Agents: An Industrial Strength Platform. In: Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.) *Multi-Agent Programming: Languages, Platforms and Applications*, pp. 175–193. Springer (2005)
49. Winikoff, M.: How testable are BDI agents? An analysis of branch coverage. In: Osman, N., Sierra, C. (eds.) *Autonomous Agents and Multiagent Systems: AAMAS 2016 Workshops, Best Papers*, pp. 90–106. Springer International Publishing, Cham (2016), doi:10.1007/978-3-319-46882-2_6
50. Winikoff, M.: Debugging Agent Programs with “Why?” Questions. In: Das, S., Durfee, E., Larson, K., Winikoff, M. (eds.) *Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (2017)
51. Winikoff, M., Padgham, L.: Agent Oriented Software Engineering. In: Weiß, G. (ed.) *Multi-agent Systems*, chap. 15, pp. 695–757. MIT Press, second edn. (2013)
52. Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multi-agent systems with MABLE. In: *Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. pp. 952–959. ACM Press (2002)

Approaching Interactions in Agent-Based Modelling with an Affordance Perspective

Franziska Klügl¹ and Sabine Timpf²

¹ School of Natural Science and Technology

Örebro University

SE 70182 Örebro (Sweden)

`franziska.klugl@oru.se`

² Institute for Geography

Augsburg University D 86135 Augsburg (Germany)

`sabine.timpf@geo.uni-augsburg.de`

Abstract. Over the last years, the affordance concept has attracted more and more attention in agent-based simulation. Due to its grounding in cognitive science, we assume that it may help a modeller to capture possible interactions in the modelling phase as it can be used to clearly state under which circumstances an agent might execute a particular action with a particular environmental entity.

In this discussion paper we clarify the concept of affordance and introduce a light-weight formalization of the notions in a way appropriate for agent-based simulation modelling. We debate its suitability for capturing interaction compared to other approaches.

1 Introduction

A critical part of building an agent-based model is related to interactions between agents, as well as between agents and other objects in their environment. There is an inherent gap between formulating agents, their properties, individual goals and/or behaviour at the micro level and the overall intended outcome observable at a macro level. When running the simulation, the simulated agents – put together and into an environment –, eventually *generate* this aggregated outcome. Interaction hereby forms the element of the model that connects micro- and macro level. Yet, one cannot easily foresee who will actually interact with whom in the running simulation. Diverse methodologies for developing agent-based simulation models propose different solutions to produce some form of predictability of interactions, defining a systematic approach to formulations in the model.

In this contribution we aim at clarifying the concept of an affordance so that it becomes a helpful notion for general agent-based simulation model development. We suggest a formalization that – if embedded into an appropriate development methodology – can support a more reliable model development by explicitly representing potential interactions. Affordances have been seen as useful in so diverse areas such as Human-Computer Interaction and Virtual Reality,

Robotics or spatially explicit agent-based simulation. Our goal is to develop a concept that supports a modeller in capturing interactions, not in a way to be able to automatically reason about them before running a simulation, but in a way to make the modeller aware of the circumstances in which interactions happen or not. Interactions that occur during a simulation run need to be fully explainable. Analysis of interactions that actually happened during the simulation, shall supporting understanding and thus quality control of the simulation model. We argue that affordances – due to their grounding in cognitive science theory – form a natural basis for guiding a modeller.

In the following we first set the scene by discussing how interactions are handled when developing agent-based simulation models, this is followed by a discussion of related work on affordances in agent-based simulation. We then introduce our particular interpretation of the original affordance notion, define affordances for use during simulation runtime and affordance schemata to be specified during model definition. We illustrate how those notions can be applied in a small example. The contribution ends with a discussion of challenges not yet addressed and our future planned work.

2 Formulating Interactions

There are different perspectives that a modeller may consider when developing with an agent-based simulation model. Depending on the particular methodology applied, the set of perspectives is different. Yet, there is a core set containing first, a model of the agents, and second, a model of the simulated environment in which the agents are embedded. The third perspective aims at capturing the interactions between the those elements of the model. A fourth perspective deals with the simulation infrastructure containing information about scheduling updates, time model, et cetera.

The perspective of a single agent is well understood - using techniques and meta-models elaborated in diverse agent architectures such as rule-based systems or BDI agents. The environmental perspective received much attention over the last years, sometimes mixed with the infrastructural elements especially when handling the environments' update from the actions of the agents that should happen in parallel. Yet, the interaction perspective in a general sense appears to be neglected.

The Merriam-Webster dictionary shortly characterizes "interaction" as "mutual or reciprocal action or influence". In addition it distinguishes between two forms of interaction: 1) Interaction as communication and 2) interaction as mutual effect. The first form is often adopted in Agent-Oriented Software Development when specification of agent interaction is reduced to specification of protocols for exchanges of structured messages. The second form is more current. Considering interactions is basically a first step to connect the micro-level behaviour of agents to observations at the system or macro-level.

2.1 Interaction in AOSE

It is not surprising that formulating and dealing with interactions is at the heart of developing and analysing multi-agent systems. Basically from the beginning, researchers analysed conditions and circumstances for interaction of all kinds of agents – simple reactive to intelligent agents. J. Ferber [7] systematically analysed interaction situations.

Organization models were spotlighted as a mean to structure societies of agents. Hereby, the number of potential interaction partners is restricted to agents within the group or to agents having adopted particular roles. The basic idea behind those endeavours is to make system-level behaviour more predictable. Organizational notions hereby allow determining with whom to interact, while what actually happens during interaction is formulated in a protocols. Many meta-models were proposed for organizational models (such as [8]).

AUML [3] became the de facto standard for representing agent communication protocols as it provided more flexibility and a higher abstraction level than plain UML sequence diagrams at that time. Meanwhile, the corresponding UML2 diagrams offer similar features [4]. At a higher abstraction level interactions and relations between agents can also be represented using UML Use Case diagrams [6].

2.2 Interaction in Agent-based Simulation

When specifying a particular behaviour with which an agent interacts or interferes with another entity, it is essential to understand in which context the interaction will actually happen during runtime. This reads strange as one may assume that only what is given during modelling, is actually happening during simulation. However, this is just the case in models in which interaction situations are fully given - e.g. the above mentioned pre-defined organizations exactly provide such fully determined place. Yet this is not the case in general. In a simulation with a kind of stigmergic interaction, an agent modifies an environmental entity, another agent perceives the result and reacts to it. Interaction here consists of action and perception in a decoupled way. Who actually reacts to the modification is unclear, when the modeller determines the agent behaviour and potential interaction. Stigmergic interaction may be an extreme example, but similar situations happen in all cases in which the agent behaviour definition contains elements that are determined during runtime – when the agent interacts in a particular situations with the entities that are actually there. This makes it so difficult to handle interaction in agent-based simulation modelling.

Definition and simulation of interactions consequently forms a major source of errors eventually leading to extensive debugging and analysing. Thus, it is a highly critical element of a modelling methodology to get the interactions right as early as possible. Their proper specification, documentation and analysis is essential.

Over the years, several approaches have been published that suggest ways of explicitly handling interactions when creating a model. Like in the AOSE

case, particular organization-level models have been proposed, such as using an institutional perspective as in the MAIA methodology [11]. Also integrating a model of social networks provides a structure who interacts with whom.

As in AOSE, UML Sequence diagrams that enable to formulate an interaction as a sequence of messages, can be used to specify interaction in agent-based simulation models [5]. For a higher abstraction level UML Use Case diagrams may be used. But, the flexible element may concern the interaction partner.

A basic framework for supporting the modelling of interaction is presented with the IODA approach [22]. In their methodology they propose to define interactions explicitly in a way separated for the actual agent models. This is also done when using explicit models of organizations, yet IODA is special as it directly couples interaction to agent action. The central element of IODA is a table that label how agents of one family interact with others. The label connected to a program or script as well as some form of condition. IODA is particularly apt for reactive agents. It does not cover selection of interaction partners – all agents of a particular type within a specified distance may interact.

In this contribution, we want to explore if the concept of affordances helps to capture possible interactions in the modelling phase. Affordances also could be used to select interaction partners during a running simulation. Before we elaborate on our thoughts, we give an overview on what affordances actually are supposed to be as well as how they are currently used in agent-based simulation.

3 Notions and Usages of Affordances

3.1 The concept of affordance

The notion of affordance is at the core of ecological psychology, brought forward by J.J. Gibson [12]. Gibson defined affordances as action potentials provided by the environment: "The affordances of the environment are what it offers the animal, what it provides or furnishes, whether for good or ill". For example, a bench affords sitting to a human. The potential action of 'sitting' depends on properties of the bench, properties of the human, and on the current activity the human is engaged in. Gibson put special emphasis on this reciprocity between animal and environment, insisting that affordances are neither objective nor subjective. Thus, Stoffregen [29] defined affordances as "properties of the animal-environment system [...] that do not inhere in either the environment or the animal".

In the context of cognitive engineering Norman [24] determines the usability of environmental objects for a human carrying out a specific task by considering not only the affordances but the "perceivable affordances" of objects and the ease of perception for humans. Norman is dedicated the designing objects in such a way that their affordances become immediately perceivable by a person engaged in some task. Transferred to the context of modelling interactions a modeller needs to anticipate the affordances that will be needed within a specific action context and that the agent should be able to perceive.

Affordances in robotics reasoning [2] are used for enabling robots to handle unexpected situations. The moment an object is recognized as for example a mug, the robot can retrieve what actions it affords from an object-affordance database. Based on this the robot may adapt its plan to water plants using the mug instead of another container which is not available. This approach requires an extensive database which first needs to be assembled. Raubal and Moratz [27] developed a functional model for affordance based agent, aiming at enabling robots to perceive action-relevant properties of the environment. This research clarifies the notions but stays at an abstract level of formalization. Although they don't name it "affordances" but services, [10] present an idea that is related to both the robotics idea of affordances. They use an action planner to configure a learning scenario an educational game. Appropriate objects providing the services that are needed in the scenario are integrated into the scenario. This is not a simulation application per se, but has some relation.

In Geographic Information Science the affordance concept has been used extensively in order to model and understand human environmental perception and cognition. Jordan et al. [18] created an affordance-based model of place, discovering that the agent, the environment and the task of the agent need to be modelled in order to be able to determine affordances of places. Raubal [26] based his model of wayfinding in airports on an extended concept of affordances, including social and emotional aspects, thus enabling agents to interpret the meaning of environmental entities relevant to the task at hand. Jonietz and Timpf [15], [16] interpret affordances as a higher-order property of an agent-environment system, which is determined by agent- and environment-related properties termed capabilities and dispositions at a lower level. As in the previous modelling approaches, affordances are interpreted as properties that may be modelled and not as something that emerges from the interaction between agent and environment. However, the affordance concept emphasizes the central role of action potentials and ties the afforded action and the respective environmental entities in a pragmatic sense [13].

3.2 Affordances in Agent-Based Simulation Modelling

During the last years the concept of affordances has become popular in agent-based simulation. Affordances were basically used to enable a modeller to formulate some element in the simulated environment that the agents could use for deciding about where to go next or what to do next.

There are a number of models that aim at reproducing how a human reasons about its environment for achieving more realism. These models are highly motivated by cognitive science. The basic assumption is that following hypotheses how humans really think, the model can achieve a higher degree of structural validity. Examples for those models are [27], [26], [25] or [15]. A formalisation focussing on affordances as an emergent property based on a detailed model of spatially explicit environment as well as actions and relations in that environment can be found in [1].

Other works interpret the notion of affordances more freely: Joo et al. [17] propose affordance-based Finite State Automata. They use affordance-effect pairs to structure the transitions between states of a simulated human. Kapadia et al. ([19]) use “affordance fields” for representing the suitability of possible actions in a simulation of pedestrian steering and path-planning behaviour. An affordance is hereby a potential steering action. The affordance field is calculated from a combination of multiple fields filled with different kinds of perception data. The agent selects the action with the best value in the affordance field. A particular interesting approach is suggested by Ksontini et al. ([21]). They use affordances in traffic simulation denoting virtual lanes as an occupy-able space. Agents reason about what behaviour is enabled by the environmental situation. The affordances offered by the environment are explicitly represented by those virtual objects that offer driving on them. [20] labelled environmental entities with “affordances” such as “provides medication” as counterparts of agent needs enabling the agents to flexibly search for interaction partners or destinations.

In these approaches, affordances are used as more as rules, for representing constraints or for identifying options. They serve as a tool for flexibly connecting an agent to interaction partners. There is no intention to advance the research in cognitive science.

3.3 Our concept of affordances

Affordances capture an emerging potential for interaction between an agent in a particular mind set intending to carry out a particular action and an environmental entity or ensemble of entities that the intended action involves. The entities need to have specific dispositions that can match up with the capabilities of the agent.

We use “affordance” as a kind of technical term capturing something that would be not be capturable otherwise. We do no claim to formalize the psychological, cognitive-science view on how humans actually reason about affordances. Our focus is on helping the modeller understand and think about interactions between agent and environment. Affordance shall make the potential for interaction between an agent and its environment explicit. So, we let the affordance stand per se for a potential interaction independent of how an agent selects its actions during simulation runtime. One can see it as a “shortcut” for representing what the agent perceives as relevant for selecting an entity as an interaction partner, without explicitly listing relevant features. In Gibson’s original affordance idea there is no space for explicit selection between different affordances - the potential for action is directly linked to action in a Boolean fashion.

4 From Affordances to Interaction

Our aim is to deal with interactions in an explicit and flexible way using affordances. Therefore, we need to differentiate between an *affordance* emerging

for a simulated agent while it “moves” through its environment, and between a representation that is defined by the modeller as some kind of declarative pattern from which the perception is generated. In principle, we assume that there is something like an explicit, declarative model that the modeller creates which is then interpreted or compiled for actually executing it during simulation runtime when the agent actually “lives”. We call the run-time representations “affordance”, and the modelling-time representation “affordance schema”.

For running the actual Agent-Based Simulation, there must be some process to generate affordances. Theoretically, there is no emergence involved when a simulated agent perceives simulated affordances, as everything is defined by the modeller. Yet, from the point of view of the agent, an affordances may be indeed unexpected. For a modeller an affordance cannot “emerge” surprisingly.

4.1 Affordance and Affordance Schema

We define an affordance as a relation between a potential action and an environmental configuration. So, theoretically, it is neither a part of the environment, nor a part of the agent, but connects both. The affordance becomes noticeable by an agent a at a particular time point t during simulation. $pAff_{a,t}$ are all affordances that the agent a can perceive at time t :

$$\langle a, act, x \rangle \in pAff_{a,t} \quad (1)$$

Such an affordance denotes the possibility of establishing a relation between an agent a and an entity x with respect to action act . x may serve as an interaction partner, if the given action is executed. With the perception of the affordance, the action becomes possible. Both, a and x are in a particular state at the time point t . We apply an extended view on “state” that goes beyond pure representation of kind-of metabolic values, but also contains activity, motivations and goals, beliefs, etc. We do not make assumptions on how this state looks like in a particular model. We also need to assume that the agent a has an explicit set of distinct, potential actions from which it selects one to perform in its environment.

As given in the previous section, an affordance links a potential action to an environmental *constellation*. Per se, such a constellation is not just a single entity in a particular state, but contains context. For example, a bench affords sitting-down just if the area to sit on is sufficiently stable (state of the bench entity). Selection is influenced by the context of the entity - whether it is below a tree casting shadow on it during sunny, too hot times or under a roof that protects it from rain on a rainy day. We assume that all information that qualifies an entity for offering a particular potential action is represented in its state; information that makes it more or less qualified in comparison to other entities affording the same potential action is determined by its context. Preferences or degrees of qualification are not considered in the classical affordance concept. In [14] this classical view is extended by elaborating gradation of affordances. Nevertheless, the selection of affordances depends on the particular way the agent reasons

about affordances which should be independent from the actual affordance. An agent might follow the first affordance relation that it encounters or a random one or might evaluate different options for determining which one to prefer.

This formalization is different from [28] who see an affordance as an acquired relation between a combination of an environmental object with behaviour and an effect of this behaviour. The idea of acquiring knowledge about an affordance illustrates their robotics perspective.

Thus, we image an affordance as an explicit object (as a kind of data structure) during simulation runtime about which the agent reasons with respect to carrying it out or not. Thus, there is a need for an higher-level data structure or schema that enables to create such runtime affordance objects. Such a schema must be more than a class in the object-oriented sense from which affordance instances can be created. For being useful in modelling per se, the schema needs to contain more contextually relevant information and conditions under which the affordance actually "emerges". We define such an *affordance schema* in the following way:

$$\langle AType, act, EType, hContext, sContext \rangle \quad (2)$$

Such an *affordance schema* can be seen as a "pattern" that can be used to generate or determine affordances present in the agents environment³. An affordance schema is specified during modelling, but does not necessarily exist as an explicit data structure during simulation runtime. When a modeller specifies such affordance schemata, she explicitly writes down under which circumstances an interaction might happen between an agent of type *AType* performing action *act* with an object of type *EType*. The action in the affordance can be a more specific and parametrized version of the action given in the affordance schema. The actual action representation may depend on the applied agent architecture. The fourth and fifth elements *hContext* and *sContext* capture the circumstances under which an affordance $\langle a, act, x \rangle$ can be really created for *a* being a kind Of *AType* and *x* being a kind Of *EType*, offering the action. The difference between *hContext* and *sContext* is that the former contains hard conditions that enable the affordance - focussing on object and agent properties directly; the latter contains weaker conditions or even just criteria that make a particular constellation more favourable than others.

For example, in a park simulation (such as [30]), during a hot day, an agent *a* enters a park that is equipped with a currently broken bench *b1* in the shadow, a clean bench *b2* in the sun and a nice looking stone *st1* under shady trees. The agent *a* entering the park is tired and searches for a place to sit down (action *sitDown*). Thus, *a* scans the environment using the affordance schema for *sitDown* and generates the following affordance objects: $\langle a, sitDown, b2 \rangle$ and $\langle a, sitDown, st1 \rangle$. It does not generate an affordance for *b1* because the bench is not apt for sitting on it due to the broken surface.

³ Our idea of an affordance schema is on a higher abstraction level than what W. Kuhn called "Image Schema" in [23]. He describes an environmental constellation using spatial categories and connects them to a process that they afford.

Depending on some form of ontology capturing environmental entities such as benches or stones as entities with flat surfaces, the modeller has defined the following affordance schema as a pattern to describe potential interactions: $\langle Visitor, sitDown, ObjectWithFlatSurface, hConditions, sConditions \rangle$. Agent a is of type *Visitor* and requires for the action *sitDown* an entity of type *EntityWithFlatSurface* which is the superclass of benches, stones, etc. However, not every one of those entities affords the action for a *Visitor* agent. This is represented in *hConditions* which define under which circumstances the environmental entity e affords the action *sitDown*: $\{stable(surface(e)), height(surface(e)) < 110cm), \dots\}$. The set of soft conditions would contain for example $\{inShadow(e)\}$. The conditions may also refer to other objects present in the vicinity of e affecting whether and how well the entity e actually can afford the given action.

It is important to stress that our definition of affordance and affordance schema does not contain a description of the effect of the action it refers to. The description as in the example just contained a label *sitDown*. What this means. In simulation, we are dealing with an environment for the agents' behaviour that is part of the model - that means fully defined by the modeller. With the environment the effect of actions is usually fully defined, even if a modeller follows the conceptually cleaner distinction between agent action and environmental reaction as described by [9].

Another essential aspect distinguishing the specification of affordances in agent-based simulation versus robotics (and also agent-oriented software development) is that actions in simulation may be defined at arbitrary levels of abstraction – adapted to the abstraction level of the environment. For example the action *sitDown* may not have a lower level correspondence when the agent executes it. There might be no going towards, arching joints, lowering backs, or whatever low-level commands are necessary to execute such an action. Abstraction levels might differ a lot between models describing the same phenomenon. This is also the reason why we would not expect to be able to create a set of "standard" affordances that can be used across many simulation applications.

Enabling a distinction between different environmental objects so that the agent may prefer one to another is NOT part of the original affordance idea. An affordance connects an environmental object to a potential action of the agent. How the agent reasons is not part of the affordance. Thus, conditions are intentionally only present on the affordance schema, i.e., the modelling level. The runtime affordance depends on the simulated agents' point of view within the simulated environment. But somehow during simulation, there must be a process generating the affordances that the agent then can select, etc. Thus, we need to discuss processes of how the affordance schemata generate affordances and determine the agents' actual behaviour and interaction.

4.2 Processes around Affordances

In theory, an affordance *emerges* as a potential action for an actor with a particular motivation (goal, desire...). In a simulation, it needs to be determined either by the simulated agent itself or by some higher level process which may

not be manifested as an actor in the simulation. In figure 1 an abstract view is visualized of how different elements of such a process can be connected.

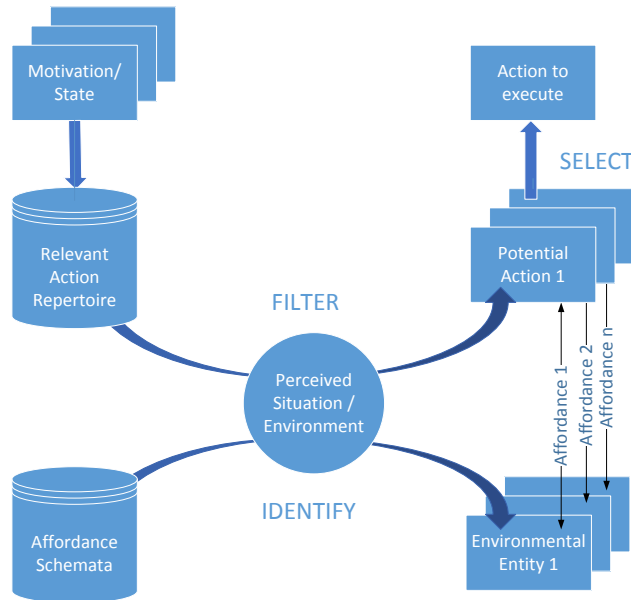


Fig. 1. Overview over processes related to affordance generation and usage.

Following figure 1 we need to elaborate partial processes relevant for creating interactive agent behaviour from explicitly defined affordances:

- Mechanism that connects agent goals to actions that are apt to achieve the agents' goal or satisfy its motivation. This process element is responsible for a pre-selection of actions from an action repertoire capturing what the agent is able to do in general. The selected actions need to be connected to the agents' motivational concepts and perceptions/beliefs in a classical way: the agent shall not select actions that it believes not to work in a particular environment, etc.
- Potential actions are filtered based on the environment checking whether the prerequisites for the actions are fulfilled or not. This is done by doing some kind of "pattern matching" of affordance schemata to perceived environment. This connects potential actions to environmental entities by generating (identifying) affordance relations.
- Having established a set of realizable actions with potential interaction partners, the agent can use the affordances connecting actions and entities to evaluate which of the combinations are the preferable ones. Such a preference relation between affordances (based on an evaluation of the context

information given on the modelling level) is then used to select the action that is executed.

5 Illustrative Example: A Supermarket

Consider the following situation and process: The agent *A* has collected a number of items in a supermarket and moves towards the cash points to pay. The agent has sufficient money (in cash or on/with card). There exist two manned cash points as well as one self-payment counter machine: $\{CashierRight, CashierMiddle, AutoCashier\}$. Of the two manned cash points, only *CashierRight* is actually busy. *CashierMiddle* misses the cashier agent. Each of the working cash points has a queue of agents waiting for their turn: In front of *CashierRight* there are 4 persons queuing up, in front of *AutoCashier* only another person is waiting for the current person to finish. So it is highly probably that *A* will be served earlier when queueing up at the electronic cash point. *A* strongly prefers to interact with humans, yet is under time pressure.

5.1 Description of Interaction with Affordances

When *A* approaches the cash point area with the intention of doing the action *Pay2Leave*, *A* perceives the three cashiers and immediately sees that only two are available for the intended action.

$$\begin{aligned} &\langle A, Pay2Leave, CashierRight, Cond_{CashierRight,now} \rangle \\ &\langle A, Pay2Leave, AutoCashier, Cond_{AutoCashier,now} \rangle \end{aligned}$$

with $Cond_{CashierRight,now} = \{queue(CashierRight, 4), female(CashierRight), young(CashierRight), friendly(CashierRight)\}$ describing the configuration of the particular cashpoint at time *now*. The configuration of *AutoCashier* is $Cond_{AutoCashier,now} = (queue(AutoCashier, 1))$.

There is no affordance for *CashierMiddle* as it is actually not working due to the missing cashier. Both affordances have particular properties that describe the current configuration the agent evaluates for making a decision for one of the interaction partner *CashierRight* or *AutoCashier*. The agent needs to evaluate whether it prefers to wait for the interaction with a nice human cashier or wants to go for the faster automated way. The *Pay2Leave* action may have a particular implementation for each of the interaction partners specified as a communication protocol as given below.

While this describes a simulation run-time situation, the modeller defines affordances schemata to specify the interaction. In this example case, the relevant affordance schema may look like that:

$$\langle SHOPPER, Pay2Leave, CASHpOINT, Prereq, PrefCriteria \rangle$$

The affordance schema contains the following elements: first, a combination of agent type *SHOPPER* and a particular action/activity that the agent wants

to do: *Pay2Leave*. Hereby, $Pay2Leave \in ActionRepertoire(SHOPPER)$, that means the action must be part of the default – as defined on the class level – action repertoire of any shopper agent. *A* as an instance of a *SHOPPER* has this action in its action repertoire. *CASHPOINT* is an abstract class from which the classes of *MANNED-CASHPOINT* AND *AUTOMATED-CASHPOINT* are inheriting. Both types can afford the *Pay2Leave* action of the shopping agent. Yet, additional conditions must be fulfilled. These conditions and criteria depend on the concrete type of *CASHPOINT* (see Table 1).

Table 1. Prerequisites and Conditions in Cashier Scenario

AType	Conditions	Preference Criteria
<i>MANNED – CASHPOINT</i>	manned(C)	Queue, Friendliness...
<i>AUTOMATED – CASHPOINT</i>	functioning(C), available(A,card)	Queue

For achieving a fully functional model clearly a lot of elements are missing. We just focus on a small number of potential interactions. Additional interactions could be between *A* and diverse products that *A* wants to buy. Hereby each product affords to be taken and put into the cart. Before we continue discussing our approach, we have a look how the corresponding formulations would look like when using ways of specification as introduced in section 2.

5.2 Description of Interaction with IODA or MAIA

In the following we intend to give a general impression of two rather extreme alternatives to formulate interactions: IODA [22] aiming more at simulation of emergent phenomena and MAIA [11] following an explicit organization-oriented approach. We do neither give full models, nor the does our description describe exactly the same part of the model. Thus, a lot of context is missing which would be necessary to precisely apply these two methodologies for developing agent-based simulations.

The central element of designing this scenario with IODA [22] is to define the interaction matrix as shown in table 2.

The table specifies that elements of one agent “family” interact with an element of another. For example, a Customer agent may initiate an interaction with a Cashier agent, if the distance between them is lower than 3 units. “Pay&Pack” is hereby a label for a sequence of actions describing the actions of the involved entities during the interaction. The model specifies what happens during an interaction, under which circumstances the interaction is triggered and what type of agents are involved. What is actually done is represented as a sequence of actions executed by the contributing agents. How the actual interaction is selected is determined by the actual agent architecture. Initially, Kubera et al. assumed reactive agents, that means agents that more or less directly connect perception to action without reasoning about explicit representations of agent goals.

Table 2. Raw Interaction Matrix in the supermarket scenario following IODA [22]. In following the steps of the overall methodology, interactions would be detailed and selection process is specified, etc.

Source \ Target	Shelf	Customer	Employee	Cashier
Customer	TakeGoods (d=0)	WaitBehind (d=2)	AskForHelp (d=2)	Pay&Pack (d=3)
Employee	ReFill (d=0)			
Cashier		RequestPayment (d=3)		

As introduced above, MAIA [11] forms a framework for agent-based simulation based on the formalization of a particular organizational model. It is especially apt for social models.

In a simulation reproducing how humans behave in a supermarket, one may assume two types of agents: Customers as individual actors and the supermarket as a composite actor, bringing together all its employees that temporally take over a particular role, such as **ReFiller** or **Cashier**. Agents may have particular attributes, such as contents of the shopping chart or entries on the shopping list. The roles have an associated objective, such as acquire and pay all items on the shopping list. There may be dependencies between roles based on dependencies between objectives - captured also in institutional settings. When adopting a role, an agent also gets capabilities that basically correspond to possible activities or actions that the agent with that role is able/permitted to perform. For the specification of interactions the set of rules and conventions that govern agent behaviour to be specified by institutional statements is particularly interesting. There are different types of those statements for describing which behaviour can be expected by an agent and what happens if the agent does not fulfil the expectations : not following a **rule** results in sanctions, a **norm** is behaviour without sanctions if not followed. The weakest notion of an institutional statement is **shared strategy**. In Table 5.2 we give a few examples of institutional statements of a customer actor in the supermarket scenario.

Table 3. Institutional Statements defining the expectations on how customer agents behave

Type of Statement	Statement
Rule	A customer always has to pay the goods before leaving
Norm	A customer has to wait in line behind earlier customers
Shared Strategy	Customers start to pack directly after the cashier accounted for a good
...	...

These elements set up the constitutional structure of the model. In addition, the modeller needs to specify the physical context (environmental model) and the operational environment, which describes how an agent influences the overall system state. A simulation has an *action arena* which contains so called *action situations*. The latter basically describes some kind of plan structure organizing atomic actions in an institutional context for an agent exhibiting a particular role. Interactions between different agents takes place within an entity actions. So, actually what happens during interaction is hidden quite deeply in the overall model specification.

In section 2 we mentioned that UML can be used to formulate interactions. How this could look like at a rather high abstraction level is shown in Figure 2.

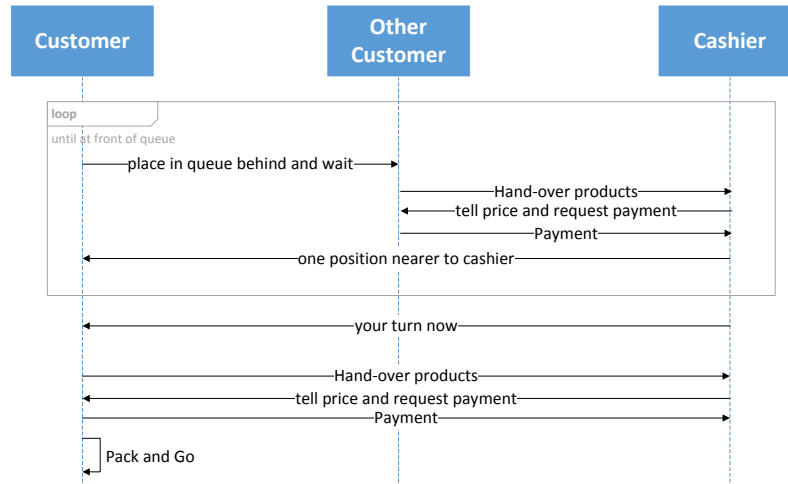


Fig. 2. Protocol-like definition of interactions between waiting customers and the cashier handling one after the other.

One can see that the different frameworks and approaches actually focus on different problems. Our affordance/affordance schema concepts actually concentrate on the selection of the interaction partner in a more flexible, yet less predictable way as in more organization-oriented approaches. One may interpret it as more specific and apt for agents that actually reason about their next action than in the IODA methodology.

6 Discussion and Conclusion

In this contribution we clarified the notion of affordances and introduced affordance schemata showing that such a distinction is necessary when distinguishing

between what happens during simulation runtime and what a modeller explicitly formulates. We put those concepts into an interaction modelling context. The questions remain whether these concepts can be really useful, what to do such that they become useful and how to evaluate their usefulness? The current stage of our research is quite preliminary, as we first wanted to clearly agree on what we actually model when specifying affordances. The current contribution thus cannot be more than a discussion paper. For creating a methodology we would need to make assumptions on meta-models formulating a context such that we can formalize every detail necessary to fully support the complete modelling and simulation process. Based on such a meta-model we could then create tools that directly support modelling - and as we explicitly approach interactions hopefully support model analysis in an improved way. However, we are not sure whether yet another methodology provides a good idea. What we actually want to achieve is to propose a suitable language that supports a modeller when formulating interactions. It should help the modeller to stay aware of when, if and under which circumstances interactions happen and which agents with which particular features participate in the interaction.

References

1. Z. Afoutni, R. Courdier, and F. Guerrin. A multiagent system to model human action based on the concept of affordance. In *4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH)*, page 8 pages, 2014.
2. I. Awaad, G. Kretschmar, and J. Hertzberg. Finding ways to get the job done: An affordance-based approach. In *24th Int. Conference on Automated Planning and Scheduling (ICAPS 2014), Portsmouth, USA, June 2014*, pages 499–503, 2014.
3. B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In *1st Int. Workshop on Agent-Oriented Software Engineering (AOSE 2000)*, pages 91–103. Springer-Verlag New York, Inc., 2001.
4. B. Bauer and J. Odell. Uml 2.0 and agents: How to build agent-based systems with the new uml standard. *Eng. Appl. Artif. Intell.*, 18(2):141–157, 2005.
5. H. Bersini. UML for ABM. *Journal of Artificial Societies and Social Simulation*, 15(1):9, 2012.
6. A. Chella, M. Cossentino, and U. L. Faso. Applying UML use case diagrams to agents. In *Proc. of AI*IA 2000 Conference*, pages 13–15, 2000.
7. J. Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Addison Wesley Longman, 1999.
8. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proc. of the 3rd Int. Conf. on Multi Agent Systems (ICMAS1998)*, pages 128–135, Washington, DC, USA, 1998. IEEE.
9. J. Ferber and J.-P. Müller. Influences and reactions: a model of situated multiagent systems. In *Proc. of the ICMAS'96*. AAAI Press, 1996.
10. G. R. Ferdinandus, M. Peeters, K. van den Bosch, and J.-J. C. Meyer. Automated scenario generation - coupling planning techniques with smart objects. In *Proc. of the 5th Int. Conference on Computer Supported Education, Aachen, Germany*, pages 76–81, 2013.

11. A. Ghorbani, P. Bots, V. Dignum, and G. Dijkema. MAIA: a framework for developing agent-based social simulations. *Journal of Artificial Societies and Social Simulation*, 16(2):9, 2013.
12. J. J. Gibson. *The Ecological Approach to Visual Perception*. Houghton Mifflin, 1979.
13. K. Janowicz, S. Scheider, T. Pehle, and G. Hart. Geospatial semantics and linked spatiotemporal data –past, present, and future. *Semantic Web*, 3(4):321–332, 2012.
14. D. Jonietz. *From Space to Place - A Computational Model of Functional Place*. Thesis, University of Augsburg, Geographical Information Science, 2016.
15. D. Jonietz and S. Timpf. An affordance-based simulation framework for assessing spatial suitability. In *Proc. of COSIT 2013, Spatial Information Theory (LNCS 8116)*, pages 169–184, 2013.
16. D. Jonietz and S. Timpf. On the relevance of Gibson’s affordance concept for geographical information science. *Cognitive Processing - International Quarterly of Cognitive Science*, 16 (1 suppl.):265–269, 2015.
17. J. Joo, N. Kim, R. A. Wysk, L. Rothrock, Y.-J. Son, Y.-G. Oh, and S. Lee. Agent-based simulation of affordance-based human behaviors in emergency evacuation. *Simulation Modelling Practice and Theory*, 13:99–115, 2013.
18. T. Jordan, M. Raubal, B. Gartrell, and M. J. Egenhöfer. An affordance-based model of place in GIS. In T. Poiker and N. Chrisman, editors, *Proc. of 8th Int. Symposium on Spatial Data Handling, Vancouver, CA*, pages 98–109, 1998.
19. M. Kapadia, S. Singh, W. Hewlett, and P. Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games (I3D ’09)*, pages 215–223, New York, NY, USA, 2009. ACM.
20. F. Klügl. Using the affordance concept for model design in agent-based simulation. *Annals of Mathematics and Artificial Intelligence*, 78:21–44, 2016.
21. F. Ksontini, R. Mandiau, Z. Guessoum, and S. Espié. Affordance-based agent model for traffic simulation. *Journal of Autonomous Agents and Multiagent Systems*, 29(5):821–849, 2015.
22. Y. Kubera, P. Mathieu, and S. Picault. IODA: an interaction-oriented approach for multi-agent based simulations. *Autonomous Agents and Multi-Agent Systems*, 23(3):303–343, 2011.
23. W. Kuhn. An image-schematic account of spatial categories. In *COSIT 2007*, volume 4736 of *LNCS*, 2007.
24. D. A. Norman. *The invisible computer*. MIT Press, 1999.
25. S. Paris and S. Donikian. Activity-driven populace: A cognitive approach to crowd simulation. *IEEE Computer Graphics and Applications*, 29(4):34–43, 2009.
26. M. Raubal. Ontology and epistemology for agent-based wayfinding simulation. *Int. Journal of Geographical Information Science*, 15:653–665, 2001.
27. M. Raubal and R. Moratz. A functional model for affordance-based agents. In E. Rome, J. Hertzberg, and G. Dorffner, editors, *Towards Affordance-Based Robot Control*, volume 4760 of *LNCS*, pages 91–105. Springer Berlin Heidelberg, 2008.
28. E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk. To afford or not to afford: A new formalism of affordances towards affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007.
29. T. Stoffregen. Affordances as properties of the animal environment system. *Ecological Psychology*, 15(2):115–134, 2003.
30. S. Timpf. Simulating place selection in urban public parks. In *Int. Workshop on Social Space and Geographic Space, SGS’07, Melbourne*, 2007.

Augmented Agents: Contextual Perception and Planning for BDI architectures

Arthur Casals¹, Amal El Fallah Seghrouchni², and Anarosa A. F. Brandão¹

¹ Laboratório de Técnicas Inteligentes - EP/USP

Av. Prof. Luciano Gualberto, 158 - trav. 3 - 05508-900 - São Paulo - SP - Brazil

{arthur.casals,anarosa.brandao}@usp.br

² Sorbonne Universités, UPMC Université Paris 06, CNRS, LIP6 UMR 7606

4 place Jussieu, 75005 Paris - France

amal.elfallah@lip6.fr

Abstract. Context-aware systems are capable of perceiving the physical environment where they are deployed and adapt their behavior, depending on the available information and how it is processed. Ambient Intelligence (AmI) represents context-aware environments that react and respond to the requirements of people. While different models can be used to implement adaptive context-aware systems, BDI multi-agent systems are especially suitable for that, due to their belief-based reasoning. Different BDI architectures, however, use different reasoning processes, therefore providing different adaptability levels. In each architecture, contextual information is adherent to a specific belief structure, and the context-related capabilities may vary. We propose a framework that can be used by BDI agents in a multi-architecture scenario in order to modularly acquire context-aware capabilities, such as learning, additional reasoning abilities, and interoperability. When this framework is combined with an existing BDI agent, the result is an augmented agent.

Keywords: context-aware systems, AmI, multiagent systems, BDI, contextual planning, learning

1 Introduction

Ambient Intelligence (AmI) [1] is a term originally created by the European Commission in 2001 [2] and represents the merging between physical environments and information technology, where embedded electronic devices can perceive and respond to the presence of people. When electronic devices or systems capture and use information on the surrounding environment to perform their functions, the interactions between these systems and the individuals present in the environment can be modified and refined in order to adapt and respond in a specific manner. The adaptability level attained by these systems and devices depend on how the surrounding environment information is collected, and the information collection process usually involves different technologies: electronic sensors (temperature, etc.), wireless networks, and human-centered interfaces are among them.

There are a few aspects, however, that need to be addressed to make this adaptability possible. One of these aspects refers to the information structure: it is necessary that information from the environment can be described and structured in order to be used by the adaptability process involved. Devices and systems capable of capturing this information and use it to adapt their functions accordingly are called context-aware systems [3], while information from the environment itself can be referred to as context [4]. Context can be represented and used in different ways – usually depending on which information dimensions or aspects are relevant to the context-aware system or device using it [5,6]. Different systems, however, present different data needs - both in terms of structure and relevance. Another aspect is the level of complexity involved when these systems interact among themselves. The addition or subtraction of another context-aware system can trigger a certain level of cooperation, which impacts the adaptability process.

Agent architectures are among the ones that can be used by context-aware systems [7]. The belief-desire-intention (BDI) architecture [8] is of particular interest due to its inherent use of contextual information. In fact, the context about the environment in which a BDI agent is situated is represented in its beliefs. Beliefs are used to determine its intentions - what the agent has chosen to do, and how committed it is to that choice [9]. However, different BDI architectures use different belief structures - therefore, translating context into beliefs is a problem highly dependent on the agent internal architecture. Thus, deploying agents implemented according to different BDI architectures into the same environment can become a challenging problem.

Deploying multiple BDI architectures into multiple environment present challenges related to all interactions between the agents and the environments. Establishing communication between different agents can be achieved through the use of a common communication protocol¹. Nevertheless, existing agents deployed to new environments may require changes in its belief structure in order to process the new context structure. Adapting an existing agent to a new environment is not only a matter of abstracting the available information, but also making sense of it - which also impacts the planning process used by the agent. When the same agents are used across different environments, multiple abstractions are required, resulting in a scalability problem.

Adding new functionalities to one specific agent could also require modifications in the other agents, as well as in the related environments (i.e., deploying coordination systems). Functionalities such as collective learning, experience sharing, or context-based planning mechanisms such as CPS-L [10] demand even more complexity to be added to the integration model used by the different agents.

With these considerations in mind, we propose a framework that can be used by different BDI agent architectures in order to augment them with context-aware capabilities. The objective of this framework is to serve as an initial step towards solving the problem of creating a multi-BDI environment that can be

¹ <http://www.fipa.org/specs/fipa00001/>

used in AmI scenarios. The initial version of this framework consists of two modules. The first module addresses the considerations regarding contextual information, augmenting the agent with context-aware capabilities through the use of multiple information sources and structures. The second module is an adaptation of the aforementioned CPS-L, which makes the agent capable not only of learning from its previous experiences, but to share these experiences among other agents. When these modules are combined with an existing BDI agent, the result is an augmented agent, with modular and inter-operable context-aware and learning capabilities. In the next sections we will refer to BDI agents simply as "agents".

It is important to mention that this framework does not replace or externalize the belief reasoning process. Depending on the deployment environment, context information may not be available all the time. In this case, the use of a context module may relieve the agent from continuously monitoring the environment. Context can be cached, preprocessed, and delivered to the agent, ready to be used in its reasoning process. Similarly, the CPS module makes the planning process more efficient by selecting in advance which plans are feasible. Agents are treated as black boxes, and the framework facilitates the integration between different contexts and agent architectures.

This paper is organized as follows: Section 2 details the general considerations used when constructing the proposed model. The augmented agent model is presented in Section 3, and the framework constructed from this model is detailed in section 4. In Section 5, we present the augmented agent implementation. Section 6 describes the application of a proof-of-concept in order to illustrate our work. Discussions and related work on the modeling and use of AmI agents and systems is presented in Section 7. In Section 8 we present our conclusions on the proposed model.

2 General considerations

Since the framework is intended to be used in conjunction with different BDI architectures deployed into context-aware scenarios, we focused our efforts on understanding the origins, limitations, and aspects related to the processing of the contextual information before it is used by the agent. The process of physically gathering contextual information is not part of the scope of this work, nor is comparing existing BDI architecture implementations. We divided our considerations into (i) the context itself - the presentation and relevance of contextual information available - and (ii) processing contextual information before delivering it to the agent.

2.1 Context

Contextual information can be collected and distributed differently. It can also be detailed and organized in different levels, depending on its intended use.

Different constraints can also determine its distribution model, such as interoperability with a preexistent communication model or bandwidth limitations. Therefore, the process of collecting and distributing context data can be broken into (i) *Gathering* (what is measured and how it is done), (ii) *Aggregation* (consolidation of collected data into information), (iii) *Representation* (structure used to represent the information), and (iv) *Transmission* (protocols and data contracts used in communication).

Aggregation is the most relevant aspect related to context, since information can be organized in different ways, depending on its purpose. This organization can also differ across different information dimensions. Mobile devices, for example, use different sensors to gather data mostly related to physical environment aspects, such as localization and acceleration. Information on the social information dimension is limited or non-existent. On the other hand, identification cards can retain organizational data – such as role in the company, unique identification record, and clearance level. In this case, the social information dimension is more detailed than in the previous one, while the physical information dimension is almost non-existent.

Different information dimensions can be used in different cases, mostly depending on what is being captured by which sensors. As a term, "information domain" is broadly used to refer to different aspects and purposes of information organization [11]. Generally speaking, information domains can be used to represent deterministic sets of information that are different among themselves in both content and organization [12]. Depending on how the information is organized, the content - or what is being represented - can be determined by its own representation. An ontology, for example, can be defined as a set of terms of interest in an information domain, along with the relationships among these terms [13].

In the scope of this work, context can be comprised of different information domains. Sensor data gathered and aggregated by an internal sensor network, for example, can be considered as an information domain within a given environment. Another information domain could be represented by user preferences stored and organized in a mobile device. When the user is in the environment, the contextual information is composed by both information domains - which can be used by an agent in its reasoning process.

Since our ultimate goal is to simplify the use of context by agents, it is important that we can be able to separate the information into different, treatable sets. These sets must be identified - in order to be distinguishable among themselves - and have a clear structure representation of the information they contain. Representing the information involves not only the data structure used, but also how the information is described. Various expressions can be used by different domains to convey information, which requires different processing rules for each domain. Data structures can also differ across domains. A sensor gateway, for example, can provide information in different formats – e.g. a hierarchically structured database (XML), a single text file or a serialized (binary) structure. A given format, however, doesn't necessarily determine how the information itself

is described: XML files use named elements to separate data, while a text file may require syntactic interpretation.

2.2 Information processing

Occasionally, information processing may be required before the agent can use the contextual information available. It is not uncommon for an agent to use discretized beliefs for its reasoning process. For instance, instead of relying on raw geographical information such as GPS coordinates, an agent may use determined known locations within a limited region (buildings, zones, rooms) on its planning process.

While associating discretized information and parametrized data may seem trivial, it has to be done at some point - and it is necessary to be taken into account in the framework modeling process. As an advantage, isolating the information processing from the agent implementation is beneficial in terms of overall implementation and change management. Different information domains may require different parametrization implementations for the same discretized information set, which may require different effort levels.

Another information process aspect to be taken into account is learning. Despite being able to reason over beliefs related to the environment in which they are situated, not all BDI architectures possess specific mechanisms or functionalities that allow agents to learn from past experiences or to adapt to new, different situations. Our approach towards learning is directly related to the contextual planning system, since the learning process is used as a tool for determining the best course of actions based on previous experiences. In a broader perspective, different learning processes can be used with different sets of experiences.

A specific learning process can be different according to which part of the context is relevant to the actions related to the experiences being processed. In that sense, separating the contextual information translation into beliefs from the agent architecture allows the implementation of learning functionalities associated with different information domains. While fuzzy techniques may be used to determine "hot" and "cold" temperatures, different algorithms may be applied to displacement information in order to establish optimal routes according to associated context conditions, for example.

Generally speaking, processing contextual information before delivering it to the agent means that its belief base is no more a pure reflection of the context as it exists; instead, it is a filtered perception of the surrounding environment. In that sense, it can be seen as a contextual filter to the agent. From the agent's perspective, having such filter allows for abstracting anything other than its own internal reasoning process. From an architectural perspective, this filter allows for other processing modules to use the refined contextual information separately from the agent.

3 Augmented agents model

With the aforementioned considerations in mind, we propose the following model for an augmented agent (Figure 1). We adopted a modular design to accommodate different BDI architectures within the proposed model.

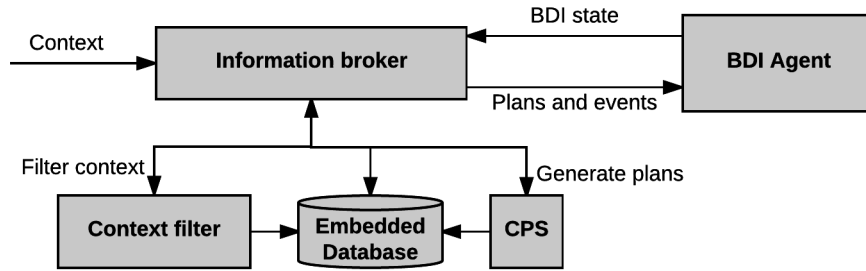


Fig. 1. Augmented agent model in perspective

All contextual information (context) is modeled as a set of different information domains, structured as described before. The module responsible for translating the contextual information received and applying any processing required before delivering it to the agent is called Context Filter. The CPS module, responsible for generating optimal plans for the agent, is also shown. All modules share an embedded database.

The Information Broker is responsible for orchestrating the other modules, as well as retrieving the BDI state from the agent and providing it with the generated plans and the expected events. These events are the result of the context filtering, and they are presented as the agent would expect from an external perception.

This modular design brings benefits to new implementations, such as loose coupling, orchestration, re-usability, and parametrization. Loose coupling is guaranteed through the separation of the framework modules from the agent implementation. While the context processing module can handle several information domains, the processed contextual information is delivered to the agent in a form that it already expects. It also allows for the different modules to be orchestrated through the implementation of a central communication and process coordinator. Also, any module changes or new module implementations remain transparent to the agent.

Using an internal database also facilitates re-usability and parametrization. Specifications related to information domains and learning algorithms can be re-used across existing framework deployments, for example. Additionally, pa-

rameters can be modified or transferred to another agent. Historical data can also be persisted and used to further increase the agent’s capabilities.

Modularizing and implementing an extensible generic outer layer to be used by different agent architectures is a relatively complex task. While modularizing a software layer and assigning responsibilities to its different components can be done by the use of existing process patterns or established models, implementing new modules within an existing system may be much more challenging. Orchestration schemes must be designed for change, and the information flow must be flexible enough in order to be altered with minimal effort. As mentioned before, there may also be limitations related to computational costs and deployment environments.

Design requirements were defined to instantiate the proposed model considering the aforementioned aspects. These requirements are listed in Table 1:

Table 1. Design requirements for the proposed model

Requirement	Description
R1	Loose-coupled component modularization within the framework should be used whenever possible.
R2	The framework should fit an internal database to be used by each of the modules in a on-need basis. Any persisted data should be retrievable, modifiable, and shareable.
R3	Different information sources and processing rules should be supported in order to allow for multiple, heterogeneous information domains. The addition of new information domains to an existing process should be as parametrized as possible.
R4	Different learning algorithms should be supported within the CPS. The use of CPS or its learning algorithms should also be optional, since computational constraints on existing systems may prevent its practical use.
R5	Module orchestration should be as simplified as possible. While the addition of new modules may not exempt new code implementation, the inter-module dependency should be kept at a minimal.

Providing an internal database (R2) allows for a higher level of abstraction between the Information Broker and the orchestrated models. The database uses and its relationship with the other modules will be described in the following paragraphs, along with details on each of the modules presented.

The third requirement involved defining the context structure to be used by the framework. While accounting for different information domains within a context is relatively simple, there’s the matter of establishing a structure model for a single information domain. For that purpose - and with the considerations presented in mind - we defined the following structure:

- **Identification:** refers to a local identifier for the information domain, unique for a given environment. While it is important for the agent to distinguish between different information domains, it is reasonable to assume that a global unique identifier may prove difficult to be implemented.
- **Grammar:** represents the language used within the information domain. Due to the considerations above, choosing a grammar over a simple reference matrix for representing the language used within the information domain allows for more flexible information usage scenarios.
- **I/O:** refers to communication metadata associated with the information domain, such as data structure (XML, text, binary) and channel endpoints (REST endpoints, etc.).

In the information domain structure presented above, it is important to notice that identifying communication metadata have certain advantages for the framework as a whole. As mentioned before, processing the contextual information before handling it to the agents is also part of the objective of the proposed framework. In scenarios where computational power or bandwidth is limited, for example, having this information available at a higher level may represent substantial gains in terms of communication efficiency and data processing.

It is also important to mention that in order to abide to the desired level of abstraction, the BDI architecture taken into consideration is as generic as possible [14]. The agent represented in the next section is an adaptation from the BDI architecture model presented in [15].

4 Framework architecture

The next paragraphs detail the proposed framework architecture, along with each model component. These components, once combined, produce the structure shown in Figure 2.

4.1 Context filter

The context filter is responsible not only for processing various information domains but also for extracting relevant information to be used by the agent and the other modules. From a design perspective, these responsibilities can be aggregated in two sub-modules, named *Translator* and *Updater*.

All contextual information is delivered by the Information Broker to the *Translator*, which transforms it into a common internal structure that can be used by the system. This process requires the use of information domain mappings, which contain individual grammar definitions that are used by an internal parser. Using a generic parser in association with a parametrized grammar allows for the framework to process new information structure without the need for implementing new code - only the new grammar structure is needed. The result of this process is the translated context.

Once the translated context is produced, the *Updater* sub-module persists it in the database. While rules and environment variables such as location and

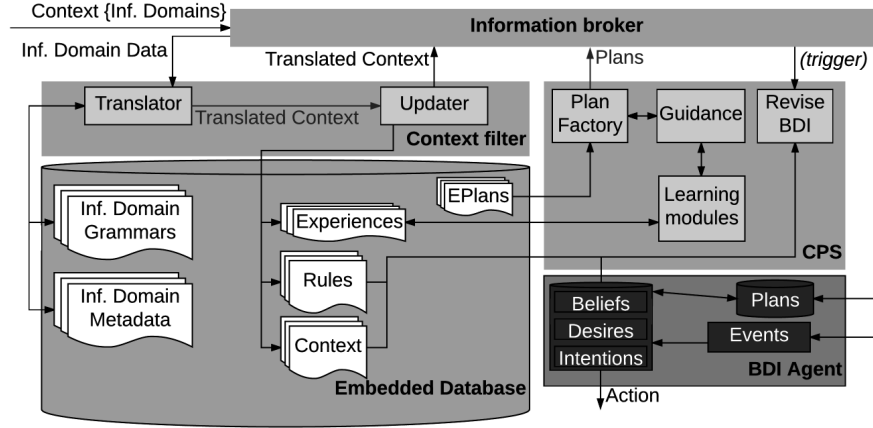


Fig. 2. Augmented framework in detail

temperature may be persisted immediately, identifying and persisting experiences may require access to historical data. Location information, for example, can be compared to older data to determine the time it took for an agent to move from one place to another.

4.2 CPS

We used the original CPS-L structure as reference when designing this module. CPS-L was originally presented as a planning process method to be used by an agent to select executable plans from its current set of intentions. These plans are the result of a selection process based on analyzing the current context associated with the agent, as well as its current set of intentions. This process also requires a revision of any restrictions that might be associated with each of the actions allowed to be performed by the same agent.

The CPS-L planning process also incorporates a guidance component that can learn from past experiences. This component uses experiences from previously executed actions to predict the outcome of the analyzed plans, and therefore influencing the planning process.

In addition to the planning process, we also used the original concepts related to plans. The original planning system describes the agent plans as a composition of sub-plans called *Intention plans*. These sub-plans are separated according to the agent's intentions, in a manner that each of them is dedicated to the achievement of one specific intention. They can be decomposed into several *Elementary plans*, which compose the set of all plans that can be executed by an agent. In that sense, all agent plans can be ultimately decomposed into a subset of the set of *Elementary plans*.

Our objective was not to propose a new context planning system; instead, our intention is to use it as-is to demonstrate that different context-related abilities can be incorporated into existing BDI architectures. We present the structure of the CPS module, comprised by the following components: (i) *Plan Factory*, (ii) *EPlans*, and (iii) *Guidance*.

Plan Factory is the component that builds the plan to be executed by the agent. This plan is a composition of *Elementary plans* built in a manner that different *Elementary plan* composition can lead to the same result. Additionally, *Elementary plans* have restrictions associated with them, allowing for the verification of viability for each of them, according to the context conditions. All *Elementary plans* are stored in a library named *EPlans*, which resides in the embedded database.

Guidance is the component that uses the experiences contained in the database to affect the whole plan generation process. That way, not only the feasibility of a plan can be verified, but also an optimal plan can be chosen among other multiple viable plans. All experiences are associated with an action and a set of contextual variables.

4.3 Information broker

The Information Broker module concentrates two main responsibilities: (i) orchestrating the other auxiliary modules, deciding and triggering their use when necessary; and (ii) delivering information (plans and events) to the agent.

Orchestrating the auxiliary modules is done through the use of a mapped information flow - each module is activated when needed, if needed. Delivering processed information to the agent requires knowing how it should receive this information. This is done by using grammars and auxiliary parametrization, making it possible to connect the framework to different agent architectures with minimal effort.

5 Implementation

As a proof-of-concept implementation for the framework, an implementation using Java was developed. Our choice was based on the availability of existing libraries that could be used by the different modules. In order to maintain its modular nature, each module of the framework was implemented separately. The information flow was done through the use of common interfaces. A simplified UML diagram of the implementation is shown in Figure 3. Other implementation details related to each of the modules and its sub-modules are described below.

5.1 Context filter

Although the framework modularity allows for different functionalities to be implemented as modules, the context filter must always be present. This module is responsible for the most part of the data transformations involved in the

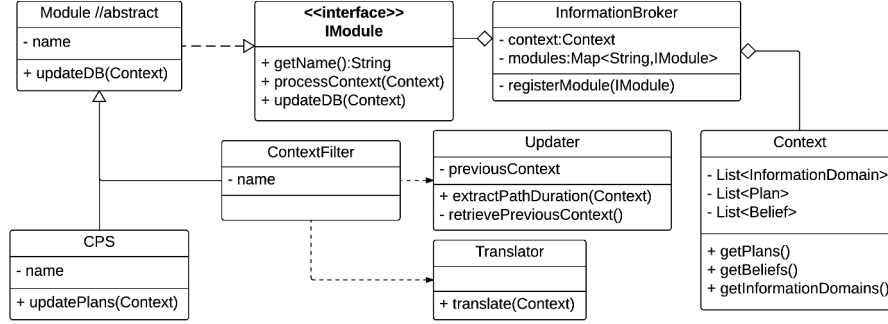


Fig. 3. Simplified UML diagram for the implementation

framework - including persisting data in the database. For this reason, we tried to keep its implementation as parametrized as possible, allowing it to be reused in future work. Each of the sub-modules were also implemented accordingly.

Each information domain is represented in the system as a pair of metadata (identification, original data structure) and associate grammar. Therefore, multiple information domains can be registered within the module through the use of parametrization. In order to use multiple grammars to translate different information domains into a common structure, we built the *Translator* sub-module to be used with a language parser generator called ANTLR¹. Since the responsibilities of the Updater sub-module include not only persisting the translated context in the database but also identifying and persisting experiences when required, we used the Active Record pattern² in its implementation.

5.2 CPS

We re-factored a previous proof-of-concept related to the original CPS-L when implementing this module. Refactoring this code involved adapting it to use the common information interfaces and the embedded database contained in the framework. It is also important to mention that this code uses a Java-based Prolog engine called tuProlog³ in its planning process.

5.3 Embedded database

Centralizing the orchestration of different modules while allowing for them to directly access the embedded database raises concerns on data integrity and database concurrency. In order to minimize these problems and to simplify our implementation, we use an embedded database library called MapDB¹. This

¹ <http://wwwantlr.org>

² <https://www.martinfowler.com/eaCatalog/activeRecord.html>

³ <https://sourceforge.net/projects/tuprolog/>

¹ <http://www.mapdb.org>

library allows the persistence of Java native collections directly in the memory, while mapping the database into a Java object that is managed by the JVM. Persisted objects can then be accessed as follows:

```
File dbFile = Utils.embeddedDbFile();
DB db = DBMaker.fileDB(dbFile).closeOnJvmShutdown().make();
Locations storedLocations = db.get("locations");
```

5.4 Information broker

As mentioned before, we implemented common information interfaces to be used by all modules. Therefore, every message passing through this module (before reaching the agent) used the same structure. In order to deliver the processed context (events and plans), we had to take into account the data structure used by the agent. Also, to be able to test different agent architectures, we addressed this problem through the use of a structure similar to the one used in the *Translator* sub-module.

6 Application

Having the proof-of-concept implemented, our next step was to test it against an application scenario. We considered the following situation: two different information domains data structures were used to define the context. The first information domain was used to provide information referencing the location in GPS coordinates. Information about time and temperature was provided by the second information domain. Temperature and time were provided in a comma-separated values format (CSV), while GPS coordinates were associated to identifiers. Each of the information domains were mapped to grammars so the information could be translated by the internal parser. The information on GPS coordinates is shown below, along with the associated grammar:

```
//GPS coordinates as received by the Information Broker:
LAT:-23.557164;LON:-46.730234
//Grammar used by the translator:
grammar CoordinatesID;
coordinates: id COLON latitude SEMICOLON id COLON longitude;
id: TEXT;
latitude: COORDINATE;
longitude: COORDINATE;
COORDINATE: ('-')? [0-9] [0-9] '.' [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] ;
SEMICOLON: ';' ;
COLON: ':' ;
TEXT: [a-zA-Z]+ ;
```

All required parser classes are created by the ANTLR library, allowing the *Translator* to retrieve the relevant contextual information for each information domain:

```
//class CoordinatesIDParser parser generated by ANTLR
String sLatitude = parser.CoordinatesContext.latitude.getText();
Double latitude = Double.parseDouble(sLatitude);
```

In order to discretize GPS coordinates into known locations we used a reverse geocoding process, which returns the nearest known location from a given set of coordinates. We adapted an existing library ¹ to make it inter-operable with our embedded database. This process was implemented in the *Updater* sub-module. Using reverse geocoding allowed us to discretize any set of coordinates as follows:

```
//Double latitude, longitude;
String location = storedLocations.findNearest(latitude,longitude);
```

All functionalities necessary to compute experience involving the movement from one location to another (duration, points of origin and destiny) were also implemented within the *Updater* sub-module. We used only one learning module to process the experiences within the CPS module. The learning algorithm was based on minimal movement duration over the five most recent experiences. Two kinds of agent architectures were used in conjunction with the same framework implementation in order to test its re-usability (plans and events output). Since our goal was to assure that the processed context and plans were being propagated to the agents accordingly, benchmarking the agents was not a part of this work.

We used two architectures of agent as adjacent structures to the framework: JASON² and BDI4JADE³. While JASON uses an AgentSpeak-based syntax to represent plans and beliefs, BDI4JADE is a pure Java implementation of a BDI architecture. This allowed us to test how plans and beliefs could be delivered to different agent architectures.

Our success criteria for this test was based on: (i) correct translation from the information domains data; (ii) correct calculations for the experiences; (iii) successful CPS processing; and (iv) successful delivery of processed context and plans to the two different agent architectures. All of them were met, and the test was concluded with success. A repository containing all the files used as inputs by the framework are available in a public repository⁴, as well as the generated plans and beliefs for both agent architectures.

7 Discussion and related work

The present work aims at serving as an initial step towards solving the problem of deploying different existing agents into multiple environments. The proposed framework accomplishes that by (i) providing means to add extra functionality to

¹ <https://github.com/AReallyGoodName/OfflineReverseGeocode>

² <http://jason.sourceforge.net/>

³ <http://www.inf.ufrgs.br/prosoft/bdi4jade/>

⁴ <http://gitlab.com/casals/AAF/>

existing agents and (ii) implementing a generic contextual translating matrix to be used in conjunction with different vocabularies used by agents. In that sense, this framework is not presented as an agent-based modeling tool or another agent architecture. Context processing is presented as a mandatory component model in the framework, since it is required for all subsequent processes. At the same time, we used the CPS module as an illustration of how existing functionalities could be easily adapted and used by other BDI architectures. Enhancing the CPS planning process or proposing an alternative to it were both out of the scope of this work.

Adding extra functionalities to agents is usually done by extending its architecture, bounding the extension to a specific agent programming language [16,17] - a monolithic approach in terms of implementation. While this solution tends to be more efficient, it is not scalable in a scenario where multiple agent architectures must be modified in order to receive a new functionality or capability.

Besides being an extra functionality by its own, context processing is also dealt with in different ways. Dealing with various contexts requires multiple context models, which must be implemented somewhere between the environment *per se* and the deployed agent. Different models may be implemented in different ways, which may lead to interoperability issues when the system is required to process multiple models at the same time. Using information domains is a way to preemptively solve this problem, while maintaining the context modeling process parameterizable enough to facilitate new implementations.

Integrating existing agents with other heterogeneous systems is also a matter solved by the use of a generic translation matrix. While there are obvious drawbacks in terms of required computational power, new communication modules can be implemented and make use of the translation matrix in place. Using a parser tree generator allows for new domain-specific languages to be implemented through the use of structured grammars, minimizing development efforts.

Providing a framework to support context-aware applications is not a novel idea. Most implementations, however, are bound to specific deployment environments or vocabularies [18–21]. Frameworks such as JCAF [20] and CoBrA [21] are deployed as middleware or publisher/subscriber services, requiring additional infrastructure resources and the use of a centralized communication structure. Being limited to a vocabulary also means that the context is translated into a fixed structure. Using this structure with an existing agent would require another translation process to be implemented. In the case of Intel’s Context-Sensing SDK¹, the abstraction layer between the application and the information capturing is also limited, restricted to the sensors available and their information data structures.

Our proposal intends to overcome these concerns. Using a framework attached to the agent instead of relying on a central coordination point simplifies its deployment, and re-configuring the communication flow is no longer necessary. Its modularity also allows the deployment of augmented agents with different capabilities to the same environment. Dividing the context into sep-

¹ <https://software.intel.com/en-us/context-sensing-sdk>

arated information domains and parameterizing their structures also simplifies the adaptation process that occurs when an environment is subject to changes such as the addition or removal of information sources.

On another perspective, CArtaGO¹ provides a workspace for abstracting the environment that addresses the heterogeneity problem. This concept, however, is structurally different from the model proposed. Our work is in its initial stage, but this difference will become more clear in the future.

8 Conclusion and future work

In this paper we presented a framework to add context-aware functionalities when combined with multiple existing BDI architectures. The contribution of this work resides in (i) proposing a context awareness extensibility model to be used in conjunction with multiple BDI architectures and (ii) proposing a context processing mechanism to be used with different context representations.

From a software engineering perspective, the first contribution promotes scalability and re-usability. Parametrization and modularization are used to minimize the implementation effort involved in extending an existing BDI architecture. Functionalities already implemented can be re-used as modules attached to different agents, and the parameterization mechanism simplifies the deployment of an augmented agent to different environments. Using an embedded database also provides means to the implementation of more robust functionalities.

The second contribution allows for the simplified capturing and processing of contextual information originated from different sources. We also presented an implementation and subsequent experimentation of the proposed framework in order to properly evaluate its feasibility.

Both the extensibility and the context processing mechanisms were successfully tested. The implementation presents drawbacks related to the use of available computational resources. The study of these limitations, however, was not part of the scope of this work. Future research will include new functionality modules and the evolution of the framework towards a solution for multi-BDI environments in AmI scenarios, as well as experiments involving different BDI agent architectures.

References

1. De Ruyter, B., Aarts, E.: Ambient intelligence: visualizing the future. In: Proceedings of the working conference on Advanced visual interfaces, ACM (2004) 203–208
2. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for ambient intelligence in 2010. Office for official publications of the European Communities (2001)
3. Hong, J.y., Suh, E.h., Kim, S.J.: Context-aware systems: A literature review and classification. *Expert Systems with Applications* **36**(4) (2009) 8509–8522

¹ <http://cartago.sourceforge.net/>

4. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: International Symposium on Handheld and Ubiquitous Computing, Springer Berlin Heidelberg (1999) 304–307
5. Kim, J., Chung, K.Y.: Ontology-based healthcare context information model to implement ubiquitous environment. *Multimedia Tools and Applications* **71**(2) (2014) 873–888
6. Nalepa, G.J., Bobek, S.: Rule-based solution for context-aware reasoning on mobile devices. *Computer Science and Information Systems* **11**(1) (2014) 171–193
7. Kwon, O.B., Sadeh, N.: Applying case-based reasoning and multi-agent intelligent system to context-aware comparative shopping. *Decision Support Systems* **37**(2) (2004) 199–213
8. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1991) 473–484
9. Cohen, P.R., Levesque, H.J.: Intention is Choice with Commitment. *Artificial Intelligence* **42**(2–3) (1990) 213–261
10. Chaouche, A.C., El Fallah-Seghrouchni, A., Ilić, J.M., Saïdouni, D.E.: Improving the contextual selection of BDI plans by incorporating situated experiments. In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*, Springer International Publishing (2015) 266–281
11. Hjørland, B.: Domain analysis in information science: eleven approaches—traditional as well as innovative. *Journal of documentation* **58**(4) (2002) 422–462
12. Hennessy, P.: Information domains in CSCW. *Studies in Computer Supported Cooperative Work: Theory, Practice and Design*, Eds. JM Bowers and SD Benford, Elsevier (1991)
13. Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.: Domain specific ontologies for semantic information brokering on the global information infrastructure. In: *Formal Ontology in Information Systems*. Volume 46., Amsterdam: IOS Press, MCB UP Ltd (1998) 269–283
14. Wooldridge, M.J.: Reasoning about rational agents. MIT press (2000)
15. Balke, T., Gilbert, N.: How do agents make decisions? a survey. *Journal of Artificial Societies and Social Simulation* **17**(4) (2014) 13
16. Buford, J., Jakobson, G., Lewis, L.: Extending BDI multi-agent systems with situation management. In: *Information Fusion, 2006 9th International Conference on*, IEEE (2006) 1–7
17. Singh, D., Sardina, S., Padgham, L.: Extending BDI plan selection to incorporate learning from experience. *Robotics and Autonomous Systems* **58**(9) (2010) 1067–1075
18. Dey, A.K.: Supporting the construction of context-aware applications. In: *Dagstuhl seminar on Ubiquitous Computing*. (2001)
19. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. *Journal of Network and computer applications* **28**(1) (2005) 1–18
20. Bardram, J.E.: The java context awareness framework (JCAF)—a service infrastructure and programming framework for context-aware applications. In: *International Conference on Pervasive Computing*, Springer Berlin Heidelberg (2005) 98–115
21. Chen, H., Finin, T., Joshi, A.: A context broker for building smart meeting rooms. In: *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium*. (2004) 53–60

PLACE: Planning based Language for Agents and Computational Environments

Muhammad Adnan Hashmi, Muhammd Usman Akram, and Amal El Fallah Seghrouchni

Department of Computer Science, University of Lahore, Lahore, Pakistan,
Department of Computer Science COMSATS Institute of I.T. Lahore, Pakistan
Laboratoire d'Informatique de Paris 6, 75016, Paris, France
Muhammad.Adnan@cs.uol.edu.pk,
musmanakram@ciitlahore.edu.pk
Amal.ElFallah@lip6.fr,

Abstract. We are interested in the development of a language, called PLACE, that allows to program agents as well as their environments. Agents' actions are durative and different priorities can be associated with the goals of agents. The agents autonomously achieve their goals by using the planning mechanism built in the language. The planning mechanism ensures the achievement of higher priority goals before the lower priority goals, but allows to perform low priority actions in parallel to the high priority actions. Plans are repaired if unanticipated changes in the environment cause the plan to become unfeasible. The environment is modeled visually to help the user simulate the behavior of agents and see the execution of agents' plans.

Keywords: Agent Oriented Programming, Temporal Planning, Plan Repairing, Environment Modeling

1 Introduction

Over the years software agents has proved to be an appropriate paradigm for the development of complex systems that are distributed in nature and require autonomy. In order to tackle the inherent complexity of such systems, three different abstractions have been defined i.e. the agents, the environments and the organizations. This separation of abstractions has lead towards the development of Agent Oriented Programming (AOP) Languages (see e.g. [1], [2], [3]), Environment Oriented Programming (EOP) Languages (see e.g. [4], [5]), and Organization Oriented Programming (OOP) Languages (see e.g. [6], [7]). This paper contributes to the AOP languages as well as to the EOP languages by proposing a language PLACE (Planning based Language for Agents and Computational Environments) that not only facilitates the user to program the agents but also allows him to program the environment as per his requirements. A platform is also developed that allows PLACE agents and different environment entities to be distributed on different hosts and allows user to visually monitor the interaction of agents with the environment.

PLACE has a syntactic structure close to the Belief-Desire-Intention (BDI) based AOP languages. Most of the BDI based languages (see e.g. Jason [1], 3APL [3]) do not

incorporate look-ahead planning. Sometimes the execution of actions without planning results in the inability to achieve the goals as the actions may not be reversible and the executed actions may have used the limited resources. Moreover conflicts could arise among simultaneously executing plans and redundant actions may also be ignored. So, last few years have seen a shift towards the look-ahead planning based approach for a BDI language (see e.g. [8], [9], [10]), but these languages do not take into account the duration of agents' actions, neither do they consider the uncertainty of the environment. These systems assume that the agents' actions are instantaneous and that the effects produced on the environment are only those which are produced by the agent's actions. But these assumptions are unrealistic for the development of real world applications. So, PLACE tries to fill this gap by using a look-ahead planning based approach, where the agents' actions are durative, priorities have been assigned to goals and plans are repaired if unprecedented changes in the environment cause the plan to become unfeasible. A planning based approach also allows to coordinate the plans of multiple agents which is otherwise difficult in a BDI model, as the BDI model in its definition is a single agent model and it is the responsibility of the programmer to explicitly state the preconditions in order to avoid conflicts among multiple agents.

Plan synthesis in PLACE is done by using the Hierarchical Task Network (HTN) planning [11] techniques. Specifically, we adapt an HTN planner JSHOP2 [12] for computing the plans of agents. HTN planning, as explored by [13], is a natural candidate for planning in the BDI style programming languages. A temporal converter procedure then converts a totally ordered plan generated by JSHOP2 into a parallel position constrained plan¹, where each action is assigned a time stamp and multiple actions can be executed in parallel if possible. The agents have the ability to execute the plans and monitor the execution. Moreover agents are continuously waiting for new tasks from the user, and if the new task requires immediate achievement, the agent preempts the execution of the task currently being executed and immediately plans for and achieves the new task. An important point is that those actions of the current plan that can be executed in parallel with the higher priority task are not postponed, they are executed in parallel and only those actions are postponed which can not be executed in parallel. For this purpose we use the Proactive-Reactive Plan Merging algorithm that we had originally proposed in [14]. The agent constantly monitors the execution of the plans and a plan mender component is added to the language that is used to repair the plan if some unexpected changes in the environment cause the failure of original plan.

In our framework it is easier for the user to monitor the execution of the agents, because the agents' environment is modeled visually. Our environment modeling approach has some similarities with the model of artifacts proposed in [5]. One important difference of our approach to theirs is the fact that in their model an artifact is a physical or computational entity in the environment e.g. a printer, a sensor, a web-service, but in our framework an artifact is the conceptual or logical entity e.g. a train, a room, a city. Another notable difference is that while [5] provides Java APIs for programming the environments, we are presenting a new language with its own syntax and semantics.

¹ Position constrained plans specify the exact start time for each action, whereas order constrained plans just specify the precedence constraints between actions.

Rest of the paper is organized as follows. Section 2 presents some related work. Section 3 discusses the environment modeling and syntactic aspects of PLACE. Planning related issues (planning, plan execution, plan repairing and merging) are presented in section 4. A case study is presented in section 5 which elaborates different planning and environment modeling concepts presented in the paper. Section 6 concludes the paper.

2 Related Work

2.1 Planning based AOP Languages

Sardina et al. [8] proposed a conceptual framework and agent programming language CANPLAN for incorporating HTN planning into a BDI like AOP language. This work is triggered by an earlier work of the authors [13] where they discussed about the similarities among BDI systems and HTN planning framework. CANPLAN provides flexibility to the programmers about when to choose full look-ahead planning. The proposed language extends the high level formal agent language CAN (Conceptual Agent Notation) [15] by adding more constructs to the language in order to incorporate HTN planning. The additional formal operational semantics for such constructs have been proposed. An important construct that has been added to the language CAN is `Plan`. If P is a BDI method body, then `Plan( $P$ )` in simple words mean, 'plan for P offline, searching for a complete hierarchical decomposition.' So the BDI agent using `Plan` has to perform a full look-ahead search before the execution commences.

Lesperance et al. [9] takes into account the uncertainty in the environment by proposing contingent planning model in an AOP language. They propose to compute plan, in advance, for different possibilities that can arise during execution of the plan. INDIGOLOG [16] is another language, in the context of situation calculus, that supports planning by including a deliberation module.

In [10], De Silva et al. proposes a *classical first principles* planning approach for BDI languages to find the plans that are not currently available in the plan library. The plans generated by their approach are called *hybrid plans*, and may contain abstract operators that can be mapped back to the goals, thus allowing the agent to execute the plan using its BDI plan library. Another framework incorporating classical planning in a BDI language is presented in [17]. It extends the X-BDI [18] model to use the propositional planning algorithms for performing means-end reasoning. It is a rather theoretical work concerning the mapping of BDI internal mental states to a STRIPS like notation and back.

In another work [19], Chaouche et al. proposes a concrete software architecture in the domain of Ambient Intelligence, that incorporates contextual planning in order to synthesize plans for agents taking into consideration the current and future context. In [20] they take this work forward by endowing the agents with the ability to learn the future context from the previous experiences of actions.

2.2 Environment Modeling in AOP Languages

Environment modeling is a core ingredient for any agent oriented programming language. Researchers in multi-agent systems community have long been working on the

lines of AEIO approach (Agent, Environment, Interaction, Organization) given in [21]. Any agent oriented programming language should be able to represent the agents and environment and most importantly the interaction between agents and the environment in which they reside.

An Action and Perception model sense-plan-act (SPA) for the exogenous environments has been proposed in [22] where the interaction of an agent with its environment is a three step process. In the first step the agent perceives its environment updating its beliefs, in the second step it plans the actions to be carried out and lastly it performs the actions causing the environment to be changed. This work is inspired by the author's previous work of simpA [23] which is agent-oriented approach for programming concurrent applications on top of Java. simpA is a theoretical framework which is later implemented in the form of CArtAgO [5]. CArtAgO has introduced a computational notion of artifacts called A&A (Agent and Artifact) to design and implement agent environments. Artifact based environment modeling has been successfully used for designing multi-agent systems for data mining domains [24]. CArtAgO is an annotation-based framework built on top of the basic Java environment. On the principles of artifacts [25] has proposed a unified interaction model with Agent Organization, and Environment.

Another approach is presented by [26] to model the Multi-Agent Based Social Simulations (MABS) in which a virtual environment is partitioned into areas called cells and is supported by an underlying autonomic software system consisting of specialized agents called controllers and coordinators. Controllers manage specific cells while coordinators monitor and guide controllers in the execution of their tasks. In such an approach the abstraction level for underlying complex environment is provided. This approach is somewhat similar to previously discussed artifact based approach, the difference is that the management of artifacts or cells is handled through dedicated agents. Another variation of designing intelligent environments is proposed by [27] in which the designer is interested in delegating a part of the agent's tasks to its body.

For context-aware agents [28] has proposed a model for the interaction between context-aware virtual agents and the environment. This work emphasizes the use of extensible agent perception module, allowing agents to perceive their environment through multiple senses. Perception combination for an agent is further investigated in [29] which proposes a multi-sense perception system for virtual agents situated in large scale open environments for the DIVA [30] project.

There are many different models for environment representation which are in use. There is a strong need to establish the uniformity of the environment's representation and the agents. In order to bridge this gap, [31] have proposed an interface for the environment which is discrete in space (grid-world) and time (step-wise evolution). This is an extra layer introduced between environments and agents so that different platforms could be able to interact with same environment. This new language is called Interface Intermediate Language (IIL) providing a conventional representation for actions and percepts. Another such approach is presented by [32] for the modeling of Agent-Environment Interactions in adaptive MAS. In this work, the interaction levels are further broken down in multiple abstraction layers.

There are many other computational frameworks for environment modeling. AGRE [33] integrates the AGR (Agent-Group-Role) organizational model with a notion of environment. In AGR the agents are considered to be working in groups in a space which contains them. AGRE is based on MadKit platform [34]. Problem with the AGR is that it is only suitable for geometrical environments. Logic based frameworks are also introduced such as GOLEM [35] to represent environments for situated cognitive agents. But these frameworks do not account for the mobility of the agents as the environment is modeled through objects. Logic based environments are well suited for game intelligence.

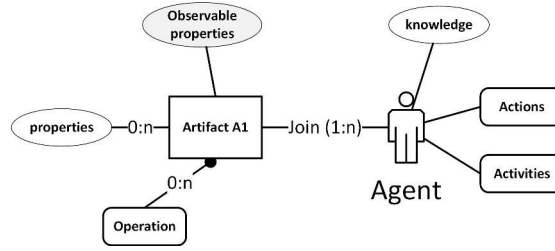


Fig. 1. PLACE Meta-Model

3 Syntax and Environment Modeling in PLACE

In PLACE we are concerned with two aspects of the environment programming: physical and logical. In the physical aspect we are concerned with the actual deployment of the agent i.e. the host and the network on which the agent is deployed etc. For this purpose, we have borrowed some ideas from AOP language CLAIM [2]. Like CLAIM, we have a central system in PLACE to which all the hosts are connected. The deployment of agents is then inside those hosts. The agents can join environment at any host. Physical mobility is possible for agents from one host to another, but in the visual modeling this physical mobility is transparent to the user, because he is more concerned with the logical mobility. In the logical aspect of environment, we are concerned with the environment as perceived by the user i.e. the user is perceiving the agent inside a train in the Paris city. For this purpose, our work has some similarities with the Agent and Artifact (A & A) meta-model of environment [5]. Artifact is an entity which presents the functionalities and knowledge to the agents. Figure 1 shows the meta model of the PLACE environment modeling. An environment in PLACE can contain multiple artifacts and agents within it. One artifact can contain multiple artifacts and can host multiple agents but an agent can be situated only in one artifact.

There are three different building blocks of PLACE language i.e. Agents, Artifacts and Environments. Following sections describe these three aspects of PLACE in detail.

Listing 1.1. PLACE Agent Definition

```
defineAgent agentName {
    knowledge = null; | { (knowledge;)+ }
    goals = null; | { (goal;)+ }
    actions = null; | { (action)+ }
    activities = null; | { (activity)+ }
    environment = null; | { environmentName }
    agentIn = null; | { artifactName }
    artifacts = null; | { (artifactName;)+ }
}
```

Listing 1.2. PLACE Artifact Definition

```
defineArtifact artifactName {
    parent = null; | { artifactName }
    connectedTo = null; | { (artifactName[{notBreakable}]);+ }
    environment = null; | { environmentName }
    properties = null; | { (property[{notChangeable}][observableTo = { (
        artifactName;)+});+ }
    operations = null; | { (operation)+ }
}
```

3.1 Agents in PLACE

An agent is defined as shown in Listing 1.1. Its components are described as follows:

- **knowledge** of the agent is what it believes about the world at a certain moment. It can be described with the help of first order propositions containing a name and list of arguments. Initially the knowledge can be empty or given by the programmer but it evolves over time as the agent executes its planned actions.
- **goals** represent current goals of the agent. The designer can give goals in the form of the tasks to be performed. Goals can be of high priority or low priority and it is the responsibility of planning and reasoning mechanism to ensure that high priority goals are achieved before low priority goals. A goal can be defined as:

*goal = {proposition [, **high** | , **low**]}*

- **actions** in PLACE are the primitive tasks that an agent is capable of carrying out. Actions are the way through which an agent can interact with the environment and manipulate it. If the agent has desired knowledge at the time of execution of an action, then the action adds/removes some knowledge to/from agent's belief base, hence modifying agent's beliefs. Actions in PLACE are durative in nature. Some actions are pre-defined in the language e.g. an agent can move from one artifact to another by using the pre-defined action *move(?source, ?destination)*. An action can be defined as:

```
action = actionSignature {
    [preconditions=precondition]
    [add_effects= {proposition (,proposition)+ }]
    [del_effects= {proposition (,proposition)+ }]
    [duration=number;]
}
```

Precondition is a collection of knowledge that an agent must have at the time of action's execution. A precondition can be a function that returns a *boolean*, a condition about agent's knowledge, a condition about being in a particular artifact, a condition about possession of an artifact, a condition about an artifact being inside another artifact, a condition about connection of an artifact to another artifact or a conjunction of any of these:

```
precondition = function(args) | hasKnowledge(knowledge) | agentIn(artifactName)
| hasArtifact(artifactName) | artifactIn(artifactName, artifactName)
| connectedTo(artifactName, artifactName) | and(precondition(,precondition)+)
```

- **activities** are the short plans that a designer can provide to the agent. Unlike actions, activities do not add or delete any knowledge in agent's knowledge base.

```
activity = activitySignature {
  [preconditions= precondition]
  do { ActivityActionSequence }
}
```

Activity's precondition is somewhat different from an action's precondition. Here the developer can use *and*, *or* and *not* operators and in any nested way he wants. The *do* element of an activity is the sequence of action or activity calls in the form of messages, that is in fact the short plan that designer is supposed to provide.

- **environment** represents the environment in which agent is situated. PLACE agent can only be the part of a single environment at any time but the agent's designer can change the information of the environment in one of its actions to move agent from one environment to another if the agent is required to perform its tasks in multiple environments.
- **agentIn** represents the name of artifact in which the agent is currently situated.
- **artifacts** are the artifacts currently possessed. When an agent moves from one place to another, it moves with all the artifacts that he currently possesses.

3.2 Artifacts in PLACE

Listing 1.2 shows, how an artifact can be defined in PLACE. Artifacts in PLACE environment present operations, properties and observable properties. Its components are described as follows:

- **parent** describes the artifact in which the artifact is currently situated.
- **connectedTo** represents a list of artifacts to which the artifact is connected. In order to model the static parts of the environment, a connection can be labeled *notBreakable* for the cases where designer wants to forbid agents from breaking connections.
- **properties** of an artifact is the knowledge which is directly accessible to the agent which is currently situated in that artifact. An artifact's property can be labeled *notChangeable* to forbid any agent from modifying it. Observable properties are accessible to all those agents which are situated in those artifacts to which the property is observable, by using the label *observableTo*.

- **operations** are the way for an artifact to expose its functionalities. Any agent attached to the artifact can perform the operation unless the artifact allows certain operations to some specific agents with the help of a keyword *agents*. On the agent's part, a boolean function *allowed* is used to check whether an agent is allowed to perform a certain operation or not.

```
operation = operationSignature {
  [agents=agentName (,agentName)+]
  [concurrent=number | agentName (,agentName)+]
  [preconditions=precondition]
  [add_effects_artifact= {proposition (,proposition)+ }]
  [del_effects_artifact= {proposition (,proposition)+ }]
  [add_effects_agents= {proposition (,proposition)+ }]
  [del_effects_agents= {proposition (,proposition)+ }]
  [duration=number;]
}
```

Operations has one-to-one mapping with actions of agent if the agent is situated inside the artifact. Operations are the only way through which an artifact's properties can be changed. Agents can also move the artifacts by changing their parent information, if such operation is allowed to the agent by the artifact. Sometimes an operation can only be performed concurrently by two or more than two agents. This is achieved through keyword *concurrent*. It can be the number of agents, or the names of agents that should perform a certain operation concurrently. Concurrency issues can quickly arise in such meta-model as multiple agents are linked to the same artifact and they can perform the operations concurrently. In order to resolve this issue, the operations of PLACE environment are thread locked and only one agent can perform operation at any given time. When the designer uses keyword *concurrent* inside an operation, the thread locked capability is turned off and the environment designer is supposed to provide preconditions accordingly.

When an operation is performed over an artifact, some facts are added/removed from the knowledge of those agents who are performing that operation. This is specified by keywords *add_effects_agents* and *del_effects_agents*. Moreover some properties of the artifact may also get changed, this is specified by keywords *add_effects_artifact* and *del_effects_artifact*.

3.3 Environment in PLACE

An environment can be defined as shown in Listing 1.3. Environment presents global knowledge to all the agents situated in the environment as the form of environment properties, through keyword *properties*. It is like a shared memory for the agents to facilitate them in communicating in decentralized manner. The global knowledge can only be accessed by the keyword *global*. Agents knowledge is thus a union of its own knowledge, global knowledge provided by the environment, artifact's knowledge and the knowledge of the artifacts which are observable from the artifact in which the agent is situated. One advancement in PLACE from the earlier version is that now the environment can also offer operations to the agent's designer. A designer now have the

Listing 1.3. PLACE Environment Definition

```
defineEnvironment environmentName{
    properties = null; | { (property);+}
    operations = null; | { (operation);+}
}
```

flexibility to design the global operations on the environment which would be available to all agents if the operation's agent property is left blank. For the cases in which designer wants to write global operations for some selected agents, he can do so by mentioning the names of the agents in individual environment operation.

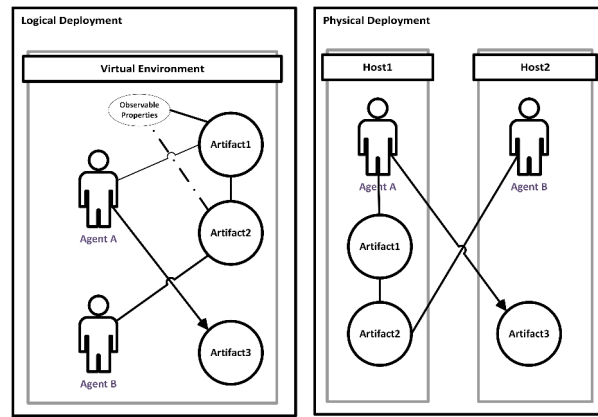


Fig. 2. Physical and Logical Deployment of PLACE Agents and Environment

3.4 Deployment and Visual Environment Modeling

The physical deployment is concerned with how the agents and artifacts are distributed among hosts on the network as shown in Figure 2 (right), whereas the logical deployment deals with how the agents perceive the environment. It shows the whole system hosted on one virtual machine as shown in Figure 2 (left). The physical mobility of agents in between hosts is hidden from the user. The logical deployment of agents and artifacts is represented by an Environment Graph, and can be viewed by user at any time during the execution of agent on the Graphical User Interface of PLACE. In an environment graph, the artifacts are represented with circle notations whereas agents are represented with a standard user like figure. The connected artifacts are shown with the help of a straight solid line e.g. Artifact1 and Artifact2 are connected. An agent can also possess artifacts which is shown with the help of a directed solid line e.g. Agent A is holding Artifact3. If an artifact is in the possession of an agent then it can only be connected with those artifacts which are in the possession of same very agent. An artifact's observable properties are shown with an oval connected to the artifact with a solid line. An artifact's observable properties can be observed by the agents from other

artifacts to which these properties are observable to e.g. Agent B is inside Artifact2 and it can see the Artifact1's observable properties. These properties are linked with the artifacts with a dashed line.

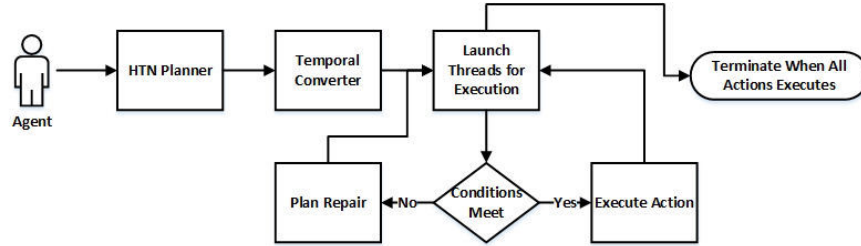


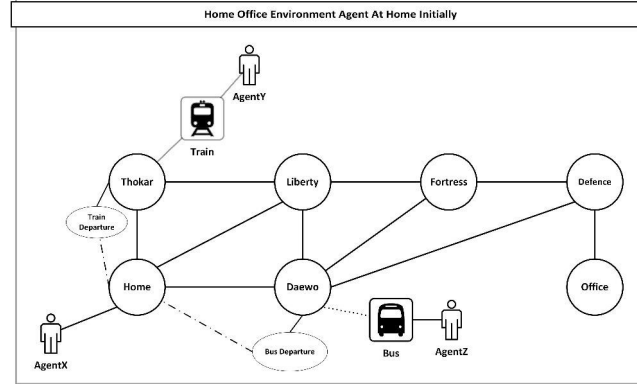
Fig. 3. PLACE Planning Work Flow

4 Planning for PLACE Agents

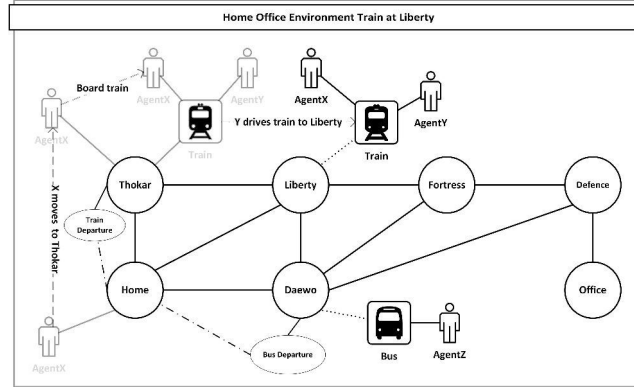
Planning is the core component of PLACE agents. Agents once launched in an environment are supposed to act intelligently without the intervention of designer. PLACE is equipped with built-in planning capability which is reusable and extendable in its nature. Figure 3 shows a work flow of PLACE planning process. At first the agent invokes an HTN planner to create a total order plan. A total order plan is the plan which is computed to generate a sequence of actions in exact order in which they are to be carried out in order to achieve the goals. For the purpose of total order planning we have chosen JSHOP2 planner in our framework. The reason to choose JSHOP2 for PLACE agents is two fold. Firstly, it is an HTN planner and the domain and problem information from PLACE can easily be translated to the domain and problem information required by JSHOP2 planner due to the similarities among HTN and BDI agent architectures [13]. Secondly, JSHOP2 plans in the same order in which the actions would be executed later e.g. JSHOP2 knows at each step the current state of the agent. PLACE agents are reactive in nature making them responsive to any change in environment. At any stage if a change occurs in the environment or the agent is given a new goal, the planner can then incorporate that change easily.

The actions in PLACE are temporal in nature e.g. each action has associative time duration. In second phase of the planning, the total order plan is converted into a parallel position constrained plan where each action is assigned a time stamp and multiple actions can be executed in parallel if possible. Temporal converter takes a total order plan as an input and specifies the start time for each action in such a way that the actions which can be carried out in parallel are scheduled with overlapping time windows.

After the planning is completed the sequence of actions generated is now passed to the executioner which launches a separate thread for each action at its start time. If the action's condition is met then the action is executed e.g. add effects are added and delete effects are deleted from agent's knowledge base and control returns to executioner with success message. If the condition is not met due to some influence of another agent or designer then the control is shifted to plan repair module.



(a) Case Study Initial State



(b) Case Study Execution

Fig. 4. Case Study

The main idea of plan repair algorithm is that if the state of world has been changed by some unanticipated event, and the preconditions of an action A no longer hold in the world, remove this action from the plan and try to compute a temporal plan from the current actual world state to a state where all the necessary effects of A hold. So, Plan Mender algorithm uses the well-known planner Sapa [36] to compute a temporal plan from the current actual world state to a state which has all the necessary effects of A . If such a plan is possible, it is returned as a replacement for the removed action. Otherwise, in the next iteration, the next action B of plan is also removed and algorithm tries to compute a plan for the achievement of the goals of the previous iteration, minus the preconditions of B , union the necessary effects of B . If such a plan is possible, it is returned as a replacement for the actions A and B . This gap is gradually widened, unless a plan is found or there is no more action to remove from the plan.

It is worth noting that the agent is receiving the goals *on the fly* and using the same procedure of planning for each goal. If the returned plan is for some reactive goal, it

Listing 1.4. PLACE: Environment & Artifacts Representation

```

defineEnvironment HomeOffice {
    properties = null;
    operations = null;
}

defineArtifact Thokar{
    environment = {HomeOffice}
    connectedTo = {Liberty,Home}
    parent = null;
    properties = {trainDepartureTime(this, 15){observableTo={Home}};}
    operations={
        sellTicket(?a){
            preconditions = {agentIn(?a,this)}
            add_effects_agents = {hasTrainTicket()}
        }
    }
}

defineArtifact Train{
    environment = {HomeOffice}
    connectedTo = null;
    parent={Thokar}
    operations={
        driveTo(?s,?d,?a){
            agents = {AgentY}
            preconditions = {
                and(agentIn(?a,this),
                    artifactIn(this,?s),
                    trainDepartureTime(?s,?t),
                    >={java.currentTimeMillis(),?t))
            }
            add_effects_artifact = {artifactIn(this,?d)}
        }
    }
}

```

is merged at the beginning of agent's global plan. We had presented a plan merging algorithm in [14], so here we are just making use of that algorithm without going into its details. If the returned plan is for some low priority goal, it is appended at the end of agent's global plan.

When all the planned actions are executed then the process of a single agent planning and execution terminates, and the agent waits for new goals.

5 Case Study

In order to demonstrate the capabilities of environment modeling and planning in PLACE, let's consider the case study presented in Figure 4(a). There are three agents AgentX, AgentY and AgentZ in the environment. Moreover, there are seven artifacts representing places, namely Home, Thokar, Liberty, Fortress, Defence, Daewoo and Office. In addition, there are two artifacts representing vehicles, namely Train and Bus. AgentX is situated at Home and its goal is to be in Office. The Train is situated in Thokar. AgentY is a driver which is present inside the Train and only he can drive train to stations Liberty, Fortress and Defence. Similarly Daewoo is a bus-stand which has a Bus situated inside it and AgentZ is its driver who can drive the bus to Defense.

Listing 1.5. PLACE: Agent Representation

```

defineAgent AgentX{
  agentIn = {Home}
  artifacts = null;
  environment = {HomeOffice}
  knowledge = null;
  goal = {{agentIn(this,Office)}};
  actions = {
    moveTo(?s,?d){
      preconditions ={and(agentIn(this,?s), connectedTo(?s,?d))}
      add_effects={agentIn(this,?d)}
    }
  }
}

```

A snippet of code for artifacts Train and Thokar is given in Listing 1.4 and a snippet of code for AgentX is given in Listing 1.5. While the AgentX is at Home it can see the knowledge of train departure time as the artifact Thokar has an observable knowledge *trainDepartureTime* to Home artifact.

At home the AgentX has three options to start its travel. It can move to the train station Thokar from where it can board the train after purchasing the ticket, or it can move to the bus-stand Daewoo and board the bus, or it can move to the train station Liberty and board the train from there if it has missed the train from Thokar. The agents AgentY and AgentZ are the driver agents which can move train and bus respectively. The *activities* of AgentX, that would help him in planning, are not shown for space reasons.

Suppose the AgentX generates a plan to move to Thokar and then to the Train, and AgentY moves the Train from station Thokar to Liberty then to station Fortress and Defence, from where the AgentX can move to its Office. The AgentX's execution of the plan upto Liberty is shown in Figure 4(b) as shaded path. Let us consider a scenario in which the station Fortress is under construction and the Train cannot continue its journey, then at this point the AgentX calls the Plan Mender to repair the plan. A new plan is given to AgentX to move from Liberty to Daewoo and then go to Defense by Bus.

6 Conclusion

We have presented an AOP language that endows agents with the capability to plan ahead and also facilitates the user to visualize the behavior of agents through environment modeling. The presented language is called PLACE (Planning based Language for Agents and Computational Environments). Agents are able to create temporal plans. Execution monitoring and plan repairing components are added. A balance between deliberation and reactivity has been established and the agents are able to turn their attention while planning to the newly arrived reactive goals. Currently, PLACE is not handling the goals with deadlines. Moreover, the time duration for an action is static i.e. it is not dependent on other parameters e.g. distance. We are working on these improvements in the language. We are also investigating the use of heuristics to guide the search for better plans in lesser time.

Bibliography

- [1] Rafael H Bordini, Jomi F Hübner, and Renata Vieira. Jason and the Golden Fleece of agent-oriented programming. *Multiagent systems artificial societies and simulated organizations*, Springer, 15:3–37, 2005.
- [2] Amal El Fallah-Seghrouchni and Alexandru Suna. CLAIM: A computational language for autonomous, intelligent and mobile agents. In *Programming Multi-Agent Systems*, pages 90–110. Springer, 2004.
- [3] Mehdi Dastani, M Birna van Riemsdijk, and John-Jules Ch Meyer. Programming multi-agent systems in 3APL. In *Multi-agent programming*, pages 39–67. Springer, 2005.
- [4] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.
- [5] Alessandro Ricci, Michele Pionti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.
- [6] Jomi F Hübner, Jaime S Sichman, and Olivier Boissier. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3):370–395, 2007.
- [7] Rosine Kitio, Olivier Boissier, Jomi F Hübner, and Alessandro Ricci. Organisational artifacts and agents for open multi-agent organisations: giving the power back to the agents. In *Coordination, Organizations, Institutions, and Norms in Agent Systems III*, pages 171–186. Springer, 2008.
- [8] Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1001–1008. ACM New York, NY, USA, 2006.
- [9] Yves Lespérance, Giuseppe De Giacomo, and Atalay Nafi Ozgovde. A model of contingent planning for agent programming languages. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 477–484. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
- [10] Lavindra De Silva, Sebastian Sardina, and Lin Padgham. First principles planning in BDI systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1105–1112. International Foundation for Autonomous Agents and Multiagent Systems, 2009.
- [11] K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1123–1123. John Wiley & Sons Ltd, 1995.
- [12] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20(1):379–404, 2003.

- [13] L. de Silva and L. Padgham. A comparison of BDI based real-time reasoning and HTN based planning. *AI 2004: Advances in Artificial Intelligence*, pages 271–299, 2005.
- [14] M.A. Hashmi and A. El Fallah Seghrouchni. Coordination of Temporal Plans for the Reactive and Proactive Goals. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 213–220. IEEE, 2010.
- [15] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufman, 2002.
- [16] Giuseppe De Giacomo, Yves Lespérance, Hector J Levesque, and Sebastian Sardina. IndiGolog: A high-level programming language for embedded reasoning agents. In *Multi-Agent Programming*., pages 31–72. Springer, 2009.
- [17] Felipe Rech Meneguzzi, Avelino Francisco Zorzo, and Michael da Costa Móra. Propositional planning in BDI agents. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 58–63. ACM, 2004.
- [18] M.C. Mora, J.G. Lopes, R.M. Viccari, and H. Coelho. BDI Models and Systems: Reducing the Gap. In *Proc. of ATAL-98, LNCS*, volume 1555, pages 11–27. Springer.
- [19] Ahmed-Chawki Chaouche, Amal El Fallah Seghrouchni, Jean-Michel Ilié, and Djamel Eddine Saïdouni. A higher-order agent model with contextual planning management for ambient systems. In *Transactions on Computational Collective Intelligence XVI*, pages 146–169. Springer, 2014.
- [20] Ahmed-Chawki Chaouche, Amal El Fallah Seghrouchni, Jean-Michel Ilié, and Djamel Eddine Saïdouni. Improving the Contextual Selection of BDI Plans by Incorporating Situated Experiments. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 266–281. Springer, 2015.
- [21] Yves Demazeau. From interactions to collective behaviour in agent-based systems. In *Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*. Citeseer, 1995.
- [22] Alessandro Ricci, Andrea Santi, and Michele Piunti. Action and perception in agent programming languages: from exogenous to endogenous environments. In *Programming Multi-Agent Systems*, pages 119–138. Springer, 2012.
- [23] Alessandro Ricci, Mirko Viroli, and Giulio Piancastelli. simpA: An agent-oriented approach for programming concurrent applications on top of Java. *Science of Computer Programming*, 76(1):37–62, 2011.
- [24] Xavier Limón, Alejandro Guerra-Hernández, Nicandro Cruz-Ramírez, and Francisco Grimaldo. An agents and artifacts approach to distributed data mining. In *Advances in Soft Computing and Its Applications*, pages 338–349. Springer, 2013.
- [25] Maicon R Zatelli and Jomi F Hübner. A unified interaction model with agent, organization, and environment. *Anais do IX ENIA@ BRACIS, Curitiba, Brazil*, 2012.
- [26] Mohammad Al-Zinati and Rym Wenkstern. A self-organizing virtual environment for agent-based simulations. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1031–1039. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

- [27] Julien Saunier. Bridging the gap between agent and environment: the missing body. *International Workshop on Environments for Multiagent Systems (E4MAS 2014)*. IFAAMAS, Springer, Paris, France, 2014.
- [28] T Steel, D Kuiper, and R Zalila-Wenkstern. Context-aware virtual agents in open environments. In *Autonomic and Autonomous Systems (ICAS), 2010 Sixth International Conference on*, pages 90–96. IEEE, 2010.
- [29] Dane M Kuiper and Rym Z Wenkstern. Virtual agent perception combination in multi agent based systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems*, pages 611–618. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
- [30] S Vosinakis, G Anastassakis, and T Panayiotopoulos. Diva: Distributed intelligent virtual agents. *behaviour*, 3:5, 1990.
- [31] Tristan Behrens, Koen V Hindriks, Rafael H Bordini, Lars Braubach, Mehdi Dastani, Jürgen Dix, Jomi F Hübner, and Alexander Pokahr. An interface for agent-environment interaction. In *Programming multi-agent systems*, pages 139–158. Springer, 2012.
- [32] R Zalila Mili and Renee Steiner. Modeling agent-environment interactions in adaptive mas. In *Engineering Environment-Mediated Multi-Agent Systems*, pages 135–147. Springer, 2008.
- [33] Jacques Ferber, Fabien Michel, and José Báez. Agre: Integrating environments with organizations. In *Environments for multi-agent systems*, pages 48–56. Springer, 2005.
- [34] Olivier Gutknecht and Jacques Ferber. The madkit agent platform architecture. In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, pages 48–55. Springer, 2001.
- [35] Stefano Bromuri and Kostas Stathis. Situating cognitive agents in GOLEM. In *Engineering environment-mediated multi-agent systems*, pages 115–134. Springer, 2008.
- [36] M.B. Do and S. Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *Proceedings of ECP-01*, pages 109–120, 2001.

Argumentation Schemes in Multi-Agent Systems: A Social Perspective

Alison R. Panisson and Rafael H. Bordini

Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Postgraduate Programme in Computer Science – School of Informatics (FACIN)
Porto Alegre – RS – Brazil
`alison.panisson@acad.pucrs.br,rafael.bordini@pucrs.br`

Abstract. Argumentation schemes are common patterns of arguments used in everyday discourse, as well as in contexts such as legal and scientific argumentation. The use of argumentation schemes may depend on the (social) context of the participating actors, the roles that they play in society, and so on. Based on this idea, this work proposes a conceptual and practical framework that combines argumentation schemes and social organisations in multi-agent systems. In our framework, the agents' social context constrains the usage of argumentation schemes and their associated critical questions. The framework has been developed on top of an existing multi-agent systems development platform, and we argue that our approach has advantages over traditional uses of argumentation schemes such as requiring less communication in multi-agent systems.

1 Introduction

Argumentation schemes are patterns for arguments (or inferences) representing the structure of common types of arguments used both in everyday discourse as well as in special contexts such as legal and scientific argumentation [31]. Clearly, different social contexts enable the use of various different argumentation schemes, based on the state of the environment (e.g., a university, a court, a hospital), the roles played by the participating actors, and the relations between their roles (e.g., professors and students, judges and lawyers, doctors and patients, etc.).

Characteristics of social contexts such as organisational roles and environment states are well-defined components in frameworks for the development of multi-agent systems, in particular those that follow an *organisation-centred* approach. Such approaches typically assume the explicit specification of an *organisation* as part of the system, where global constraints are publicly defined so that heterogeneous and autonomous agents can follow when playing some role within the (open) multi-agent system [12].

This work proposes to use the group structures, social roles, social plans, and social norms, usually available in such frameworks, as sources of possible extra elements for the specification of argumentation schemes to be used in agent dialogues for negotiation, coordination, deliberation, etc. Furthermore, by

extending the normative infrastructure of such systems, we are able to ensure that agents make appropriate use of the argumentation schemes depending on their social context, as specified using our approach to argumentation schemes in organisation-centred agent platforms.

Among the many organisational models for agents, for example [7, 8], the MOISE model [12] is a well-known practical approach, which has been integrated with the CArtAgO platform [25], used to develop a shared environment, and the Jason platform [5], used to develop autonomous agents. The resulting framework, called JaCaMo [4], provides support for the development of complex multi-agent systems covering the three main dimensions of such systems (the organisation, the agents, and the environment). In this paper, we make our proposal practical by extending the JaCaMo platform. In the resulting framework, called Soc^{ARG}, argumentation-based agent interaction becomes a fourth dimension that we add to multi-agent system specifications. As it is closely connected to the organisation specification, in this paper we focus our formalisation on the combination of these two dimensions of multi-agent specifications used in our JaCaMo extension. The main contribution of this work is the framework Soc^{ARG} we propose, including how the different components of such specifications are integrated and the benefits it brings to multi-agent system development.

2 Argumentation Schemes

Besides the familiar deductive and inductive forms of arguments, argumentation schemes represent forms of arguments that are *defeasible*¹. This means that an argument may not be strong by itself (i.e., it is based on disputable inferences), but they may be strong enough to provide evidence that warrant rational acceptance of its conclusion [30]. Conclusions from argumentation schemes can be inferred in conditions of uncertainty and lack of knowledge. This means that we must remain open-minded to new pieces of evidence that can invalidate previous conclusions [31]. These circumstances of uncertainty and lack of knowledge are, inevitably, characteristics of multi-agent systems, which deal with dynamic environments and organisations [33].

The acceptance of a conclusion from an instantiation of an argumentation scheme is directly associated with the so-called *critical questions*. Critical questions may be asked before a conclusion from an argument (labelled by an argumentation scheme) is accepted, and they point out to the disputable information used in that argument. Together, the argumentation scheme and the matching set of critical questions are used to evaluate a given argument in a particular case, considering the context of the dialogue in which the argument occurred [31].

Arguments instantiated from argumentation schemes, and properly evaluated by means of their critical questions, then can be used by agents in their reasoning and communication processes. In both situations, other arguments, probably instantiated from other argumentation schemes, are compared in order to arrive to an acceptable conclusion. After an argument to be instantiated

¹ Sometimes called *presumptive*, or *abductive* as well.

from an argumentation scheme and evaluated by its set of critical questions, the process follows the same principle of any argumentation-based approach, where arguments for and against a point of view (or just for an agent's internal decision) are compared until eventually arriving to a set of the acceptable ones.

In regards to *conflict* between arguments, we can consider the existence of two types, according to [31]: (i) a strong kind of conflict, where one party has a thesis to be proved, and the other part has a thesis that is the opposite of the first thesis, and (ii) a weaker kind of conflict, where one party has a thesis to be proved, and the other part doubts that thesis, but has no opposite thesis of his own. In the strong kind of conflict, each party must try to refute the thesis of the other in order to win. In the weaker form, one side can refute the other, showing that their thesis is doubtful. This difference between conflicts are inherent from the structure of arguments, and can be found in the work of others, e.g. [19].

To exemplify our approach, we adapted the argumentation schemes *Argument from Position to Know* from [32] to a multi-agent (organisational) platform, so that for example roles that agents play in the system can be referred to within the scheme. Consider the *Argument from role to know in multi-agent systems* (*role to know* for short) :

“Agent *ag* is currently playing a role *R* (its position) that implies knowing things in a certain subject domain *S* containing proposition *A* (**Major Premise**). *ag* asserts that *A* (in domain *S*) is true (or false) (**Minor Premise**). *A* is true (or false) (**Conclusion**)”.

The associated critical questions are: **CQ1**: Does playing role *R* imply knowing whether *A* holds? **CQ2**: Is *ag* an honest (trustworthy, reliable) source? **CQ3**: Did *ag* assert that *A* is true (or false)? **CQ4**: Is *ag* playing role *R*?

The argumentation scheme introduced above can be represented in the Jason multi-agent platform as a defeasible inference as follows (based on [18]):

```
def_inf(Conclusion,[role(Agent,Role), role_to_know(Role,Domain),
    asserts(Agent,Conclusion),about(Conclusion,Domain)])
    [as(role_to_know)].
```

where the agents are able to instantiate such argumentation schemes with the information available to them in their belief bases and to evaluate the acceptability of the conclusion based on the interactions among such instantiated arguments [18].

Formally, in our framework, an argumentation scheme is a tuple $\langle \mathcal{SN}, \mathcal{C}, \mathcal{P}, \mathcal{CQ} \rangle$ with $\mathcal{SN}$ the argumentation scheme name (which must be unique in the system), $\mathcal{C}$ the conclusion of the argumentation scheme, $\mathcal{P}$ the premises, and $\mathcal{CQ}$ the associated critical questions. Considering the example above, the corresponding components are $\mathcal{SN} = \text{role_to_know}$, $\mathcal{C} = \text{Conclusion}$, $\mathcal{P} = \text{asserts(Agent,Conclusion), role(Agent,Role), role_to_know(Role,Domain) and about(Conclusion,Domain)}$, $\mathcal{CQ} = \langle \text{cq1, role_to_know(Role,Conclusion)} \rangle, \langle \text{cq2, honest(Agent)} \rangle, \langle \text{cq3, asserts(Agent,Conclusion)} \rangle$ and $\langle \text{cq4, role(Agent,Role)} \rangle$.

3 The MOISE model

In this section we describe the MOISE model based on the work presented in [12], where the authors describe a set of computational tools that supports the development and programming of such organisation-centred systems, providing also a middleware, where agents (developed using Jason [5]) can perceive and act upon the organisation. In this context, our work proposes the representation of argumentation schemes at a social level, so that they can be adapted to the social context of such organisation-based systems.

The MOISE organisational model [12] has three main specifications: the *structural*, *functional*, and *normative* specifications. The *structural* specification (SS) has three levels: (i) the behaviour that agents are responsible for when they adopt a role (*individual* level); (ii) the acquaintance, communication, and authority links between roles (*social* level); and (iii) the aggregation of roles into groups (*collective* level) [12]. The *functional* specification (FS) is composed of a set of schemes, each scheme being similar to a goal decomposition tree, which represent how a multi-agent system usually achieves its social (organisational) goals. A scheme states how such goals are decomposed (as high-level social plans) and allocated to the agents (through so-called *missions*). Finally, the *normative* specification (NS) that explicitly states what is permitted and obligatory to agents playing roles within the organisation. The specification describes the permissions and obligations of roles in respect to missions specifically (for example, the permission $permission(r_i, m_i)$ states that an agent playing role r_i is allowed to commit to mission m_i) [12]. Therefore, a formalisation of the basics of the MOISE model can be done through a tuple $\langle \mathcal{SS}, \mathcal{FS}, \mathcal{NS} \rangle$ where (based on [3]):

- The *structural specification* (SS) is represented as a tuple $\langle \mathcal{R}, \sqsubset, rg \rangle$ with $\mathcal{R}$ a set of roles, $\sqsubset$ the inheritance relation between roles, including communication link ($link_{com}$), authority link ($link_{aut}$), and acquaintance link ($link_{acq}$), and rg the organisation root group specification.
- The *functional specification* (FS) is represented as a tuple $\langle \mathcal{M}, \mathcal{G}, \mathcal{S} \rangle$ with $\mathcal{M}$ the set of *missions*, consisting of groupings of collective or individual goals, $\mathcal{G}$ is the set of the *collective* or *individual goals* to be satisfied, and $\mathcal{S}$ is the set of *social schemes*, tree-like structures of plans for goals.
- The *normative specification* (NS) contains a set of tuples $\langle id, dm, r, m \rangle$ with id a norm identifier, dm a normative modality (obligation or permission), r is the role concerned by the normative modality, and m is a mission. This specification can be read as “any agent playing role r has dm to commit to mission m ”.

4 The Soc^{ARG} model

In this section, we propose a formal specification for multi-agent systems called Soc^{ARG}, including a new dimension which allows the specification of argumentation schemes that can be used during interaction/reasoning within an instantiated multi-agent system. As part of our framework, we also extend usual normative specifications in order to specify constraints over the use of argumentation

schemes (and their instantiated arguments) in the system, considering the roles and context of agents. Further, we describe a series of relations between the types of specifications in our approach.

Formally, Soc^{ARG} is a tuple $\langle \mathcal{SS}, \mathcal{FS}, \mathcal{AS}, \mathcal{NS} \rangle$, with $\mathcal{SS}$ the structural specification, $\mathcal{FS}$ the functional specification, $\mathcal{AS}$ the argumentation-scheme specification, and $\mathcal{NS}$ the normative specification. This formalisation, as described, is inspired in the MOISE organisational model. However, note that our proposal is not tied to the organisational specifications of MOISE, any specification of (open) multi-agent systems using those same concepts (groups, roles, social plans, sets of goals allocated to agents, and simple norms) can be used to combined with our argumentation-scheme specifications for open multi-agent systems.

The specification of argumentation schemes in Soc^{ARG} is independent from the other organisational specifications, but it is particularly connected to the normative specification which links the structural, functional, and argumentation-scheme specifications. In the argumentation-scheme specification, the argumentation schemes and their corresponding critical questions are defined. After that, in the normative specification, we can specify which argumentation schemes and corresponding critical questions can be used by agents depending on the roles they play in the multi-agent system and the communication links between such roles, both declared in the structural specification. The usage of argumentation schemes and critical questions can also consider the context of the social goals, which are associated with the functional specification. The links between the specifications present in Soc^{ARG} are represented in Figure 1.

For example, the argumentation scheme *Argument from role to know in multi-agent systems*, introduced in section 2, is specified in the XML file that specifies the multi-agent systems in Soc^{ARG} . The code in XML is presented below.

```
<argumentation_scheme_specification>
<argumentation_scheme id="as1" name="role_to_know">
  <conclusion language="Prolog" content="Conclusion"/>
  <premise language="Prolog" content="role(Agent,Role)"/>
  <premise language="Prolog" content="role_to_know(Role,Domain)"/>
  <premise language="Prolog" content="asserts(Agent,Conclusion)"/>
  <premise language="Prolog" content="about(Conclusion,Domain)"/>
  <critical_questions>
    <critical_question id="cq1" content="role_to_know(Role,Conclusion)"/>
    <critical_question id="cq2" content="honest(Agent)"/>
    <critical_question id="cq3" content="asserts(Agent,Conclusion)"/>
    <critical_question id="cq4" content="role(Agent,Role)"/>
  </critical_questions>
</argumentation_scheme>
</argumentation_scheme_specification>
```

In our current implementation of Soc^{ARG} , argumentation schemes are defined in XML, but other languages such as AML (Argument Markup Language), introduced in [23], could be used instead. This is possible because the high-level language is interpreted by a form of “management infrastructure”, like the one developed in [11]. This interpretation by a management infrastructure allows agents developed in different agent-oriented programming languages to participate in the open system and become aware of such specifications (argumentation schemes, norms, goals, etc.) through this interface.

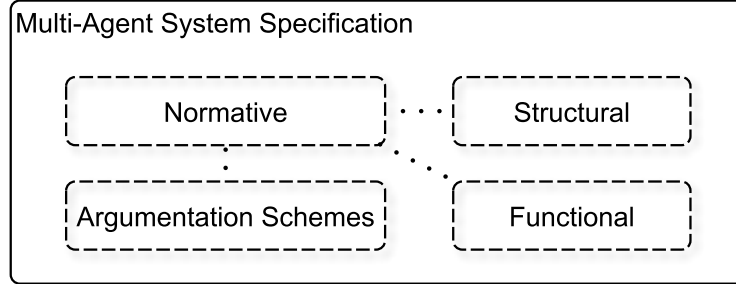


Fig. 1. Soc^{ARG} links between specifications.

In order to make explicit the use of argumentation scheme in the normative specification, the MOISE normative specification was extended² to the form of $\langle id, dm, p, scope \rangle$, where the *scope* can assume two forms: (i) $do(m)$ considering the execution of a mission m (the usual case in MOISE model), and (ii) $use(as, cons)$ referring the use of an argumentation scheme as and its respective constraints $cons$. The normative specification in XML is extended as well, to include the **scope** declaration, which allows us to determine if the norm refers to a mission or to the use of an argumentation scheme. When doing so one can specify whether that use is permitted or not, including the constraint related to the critical questions and context. An example is presented in the XML code below.

```

<normative-specification>
<norm id="n1" type="obligation" role="r1"> <scope type="do" mission="m1"/> </norm>
<norm id="n2" type="permission" role="r1"> <scope type="use" arg_scheme="as1">
  <context m_id="m1"> <except cq_id="cq1" content="role_to_know(r2,Conclusion)"/>
</scope> </norm>
</normative-specification>

```

The specification of constraints in the normative specification is common, given that agents are supposed to be able to reason about the normative specification when they are playing some role in the multi-agent system. That is, with this approach the agents are almost directly able to reason about the argumentation schemes constraints as well. Another important point is that the multi-agent organisation, considering the extended normative specification, is able to monitor the use of arguments (instantiated by the argumentation schemes) among the agents. The main specification brings up some relations to the other (organisational) specifications in Soc^{ARG}. Although the argumentation schemes, the structural and the functional specifications are independent and connected through the normative specification, as shown in Figure 1, we can define some relations between such specifications, as detailed in the next sections.

² That extension was inspired by [3].

4.1 Argumentation Schemes and the Structural Specification

The relation between the argumentation-scheme and the structural specifications is established, for the most part, in the normative specification, where the roles (from the structural specification) are linked to the argumentation schemes (in the argumentation-scheme specification) that are permitted/obligatory for each role. An indirect, but important, relation between the argumentation-scheme and the structural specifications is induced by the *communication link* between agents. Before any consideration of an agent having permissions/obligations to use certain argumentation schemes, clearly it will not use arguments instantiated from argumentation schemes (or any other kind of argument) towards agents with which it has no communication link. This consideration does not describe anything about the normative specification. Any agent can violate the norms in order to communicate to other agent using arguments. However, this communication is not possible without the communication link, because the infrastructure layer is supposed not to allow the exchange of messages by agents without communication link.

Another relation between the argumentation-scheme and the structural specifications is the declaration of groups, where it is possible to specify the cardinality of each role within each group. The relation comes from the definition of groups where agents can make extensive use of argumentation in order to achieve the system's goals. This is a direct relation with the permissions/obligation that each member of the group has to use argumentation schemes and the communication link between the members of the group.

Definition 1 (Argumentative Groups) *A group gr is an argumentative group iff there are agents $ag_i, ag_j \in gr$ playing some roles $r_i, r_j \in \mathcal{SS}_R$ and $gr \in \mathcal{SS}_{rg}$, where $\langle n_i, dm, r_i, use(as_i, cons_i) \rangle, \langle n_j, dm, r_j, use(as_j, cons_j) \rangle \in \mathcal{NS}$, $dm \in \{\text{permission, obligation}\}$, and $link_{com}(r_i, r_j) \in \mathcal{SS}_\square$.*

Note that we do not require the roles of the agents to be different, because we could have a role in the group with cardinality greater than one, and agents from the same role could argue within the group in that system.

Other relations can arise from the argumentation-scheme and the structural specifications, which will depend on the domain. For example, the argumentation scheme described in Section 2 is directly related to the roles specified in the structural specification. Any argument instantiated from this scheme should be consistent with some concrete role from the structural specification.

Definition 2 (Consistent Argumentation Schemes) *We define a consistent argumentation scheme, related to the multi-agent system specification, if all components mentioned in the scheme, involving components of the multi-agent system (e.g., roles, groups, etc.) are defined in the structural specification.*

4.2 Argumentation Schemes and the Functional Specification

Similarly to the structural specification, the relation between the argumentation-scheme and the functional specification is established, for the most part, in the

normative specification. For example, the normative specification, shown in Section 4, allows us to define the contexts where the argumentation schemes could be used in order to achieve particular goals or missions (from the functional specification). Thus, the functional specification is used to restrict the use of argumentation schemes, therefore any argumentation scheme that the agent role has permission/obligation to use could be used to achieve the goals the agent has committed to achieve, but argumentation schemes with context restrictions can be used only within a particular mission as specified.

Definition 3 (Contextual Argumentation Schemes) Contextual Argumentation Schemes are those argumentation schemes that can be used only in a particular context (to achieve a particular goal, following a particular mission, etc.). Formally, an argumentation scheme is contextual for a mission m_i if $\langle as, \varphi, \mathcal{P}, \mathcal{CQ} \rangle \in \mathcal{AS}$ and $\langle n_i, dm, r_i, use(as, cons) \rangle \in \mathcal{NS}$ with $dm \in \{\text{permission}, \text{obligation}\}$, and $\langle m_i \rangle \in cons$ for some $m_i \in \mathcal{FS}_{\mathcal{M}}$.

Contextual argumentation schemes are useful to restrict some types of argumentation schemes depending on the contexts determined by the functional specification. For example, formal dialogues could use stronger types of arguments as in the *Argument from role to know in multi-agent system* introduced in Section 2, while some other dialogues could use weaker argumentation schemes. Therefore, depending on the organisational goal to be achieved by the agents, they could be restricted by the normative specification to use only arguments instantiated from the schemes that are adequate for that context. Further, in some scenarios, the agents could be obliged to use some specific types of arguments.

4.3 Argumentation Schemes and the Normative Specification

In previous sections, we have described the argumentation-scheme specification and how the normative specification links it to the others (i.e., structural and functional specifications), including some indirect relations. However, besides the normative specification being used to link the schemes to the other specifications, it has itself some relations with the argumentation-scheme specification. In particular, we can introduce the notion of Norm-Conforming Argumentation.

Definition 4 (Norm-Conforming Argumentation) A Norm-Conforming Argumentation is an argumentation process (e.g., an ongoing instance of an argumentation-based dialogue protocol) that does not violate the normative specification, i.e., the participating agents only use arguments and critical questions instantiated from argumentation schemes that the agents are permitted to use, and satisfying also other constraints on such permissions, e.g. that the agents are playing the roles associated with the permission.

The normative system used to regulate the behaviour of autonomous entities (agents) is not the aim of our work. Therefore, we will not discuss the internals of a normative system here. For our purposes, it suffices to understand that there is some normative system regulating the behaviour of the agents, enforcing the

agents to follow the norms, for example by applying sanctions in order to try and stop agents from acting in (socially) undesirable ways [2].

We assume multi-agent systems where agents use standard languages to communicate and to represent arguments. Argumentation-based dialogues can be created based on a set of performatives (also called locutions or speech-acts) that the agents can use, and a protocol defining which moves/performatives are allowed at each step of the protocol based on some form of constraints [14, 15, 17]. One way to support agents in engaging in norm-conforming argumentation-based dialogues is defining a protocol that restrains the violation of norms, which are followed or not by agents in an autonomous way. In order to guide the definition of protocols that do not violate the normative specification of the Soc^{ARG} model, we introduce some definitions below. We start by defining norm-conforming claims.

Definition 5 (Norm-Conforming Claims) *A claim uttered by an agent ag is considered norm-conforming if it is instantiated from an argumentation scheme permitted to the role played by ag . Formally: a claim ψ is norm-conforming iff $\langle as, \psi, \mathcal{P}, \mathcal{CQ} \rangle \in \mathcal{AS}$ and $\langle n_i, dm, \text{role}(ag), use(as, cons) \rangle \in \mathcal{NS}$ with $dm \in \{\text{permission}, \text{obligation}\}$.*

Beside norm-conforming claims, we introduce also norm-conforming questions. When agents have permission to instantiate argumentation schemes for creating new arguments, there might be certain constraints related to use the critical questions to which the agents should conform as well.

Definition 6 (Norm-Conforming Questions) *A question φ for some claim ψ asked by an agent ag is considered norm-conforming if and only if it is instantiated from an argumentation scheme permitted for the role played by ag , and there are no exclusions for φ specifically in that permission. Formally: a question φ is a norm-conforming question for ψ iff $\langle as, \psi, \mathcal{P}, \mathcal{CQ} \rangle \in \mathcal{AS}$, where $\langle cq_i, \varphi \rangle \in \mathcal{CQ}$, and $\langle n_j, dm, \text{role}(ag), use(as, cons) \rangle \in \mathcal{NS}$ with $dm \in \{\text{permission}, \text{obligation}\}$ and $\langle cq_i, \varphi \rangle \notin cons$.*

5 Benefits of the Soc^{ARG} Model

There are some clear benefits in using our approach, most of them resulting from the extended normative specification, and the shared argumentation schemes. In this section, we discuss the main benefits of our approach, including an example of how our approach makes protocols more efficient, by means of reducing the number of messages exchanged by agents.

Firstly, we argue that there are some benefits which come from the definitions presented in previous sections (namely Definitions 1, 2, and 3), whereby we are able to specify, in the Soc^{ARG} specification: (i) groups of agents that are able to argue; (ii) contexts in which they could argue; and (iii) to guide agents to argue according to norm constraints. In (open) multi-agent systems, having this kind of “control” in the multi-agent specifications is rather valuable, given

that different application domains could require different constraints in regards to communication. As an example, we could mention multi-agent applications recently developed on mobile systems. Normally, such application domain has to restrict communication over the mobile network, using an architecture based on *personal* and *server* agents, where (i) *personal* agents are only responsible for collecting user information, sending it to corresponding *server*-side agents, and to interact with users by means of an interface, and (ii) *server* agents are responsible for most of the processing, decision-making, and (normally intensive) communication with others users' *server* agents. Examples of such systems are found in [9, 28]. Our approach allows for grouping agents and making them argumentative groups or not, depending on the application needs. Further, our approach allows specifying the contexts in which certain kinds of arguments could be permitted or prohibited, thus making the use of arguments specific for particular tasks in a multi-agent coordinated activity.

Secondly, besides of the benefits related to the control of communication, our approach encourages argumentation schemes (reasoning patterns) to be shared knowledge within an agent system. Such an approach allows us to assume a more rational position for an agent in argumentation-based dialogues, where they are able to answer more critical questions by themselves. This occurs given the consideration of social/organisational components in the argumentation schemes, where such components are common knowledge to all agents.

In order to demonstrate such improvement in agent communication, we present a simple and restrict protocol³, and we demonstrate how it works in our approach. Although the protocol is simple and restrictive, it could be seen as “a part of” protocols based on argumentation schemes, therefore the benefits, we will demonstrate here, are valid for other (more realistic) protocols as well. In this simple protocol, each dialogue starts with claiming a certain proposition (*assert p*), which is a conclusion from an argument $\langle \mathcal{P}, p \rangle_{sn_i}$ instantiated from an argumentation scheme, here called sn_i , available for all agents, i.e., $\langle sn_i, p, \mathcal{P}, \mathcal{CQ} \rangle \in \mathcal{AS}$, with $\mathcal{P}$ and $\mathcal{CQ}$ the instantiated sets of premises and critical questions for that scheme, respectively. Then, the claim can be conceded (*concede p*) or challenged (*why p*). A challenge can be replied to with an argument $\langle \mathcal{P}, p \rangle_{sn_i}$. Arguments can be replied to by conceding its conclusion p , or questioning the argument by the critical questions pointed out by the scheme – i.e., *question cq_i* for any $cq_i \in \mathcal{CQ}$. Questions can be answered appropriately (*answer cq_i*), or the initial proposition can be retracted (*retract p*). Although we are not interested in the properties of such protocol (it was presented just to show the benefits of our approach in regards to message exchange requirements), it is easy to note that dialogues end when all critical questions were answered appropriately, the agent concedes to the initial claim, or when the initial claim is retracted, i.e., when the agent is unable to answer any critical question adequately.

Dialogues following the protocol introduced above will have: (i) in the best case, 2 message exchanged, and (ii) in the worse case, $4 + 2|\mathcal{CQ}|$. For example,

³ We took inspiration from the protocol presented in [20].

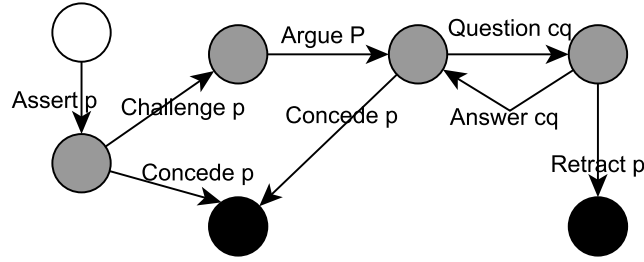


Fig. 2. A Simple Protocol.

if an agent instantiates the argumentation scheme *Argument from role to know* we presented, and starts a dialogue: (i) in the best case, the agent will only just assert the claim A (a propositional content) and the other agent engaged in such dialogue will concede to A (2 messages exchanged); and (ii) in the worst case, the agent will assert A , the other agent will challenge A , it will argue that A was asserted by an agent ag who is currently playing the role R that implies knowing things in the subject S , which contains proposition A , therefore A is true (so far, 3 messages exchanged). After that, the other agent will question **CQ1**, the agent that started the dialogue will answer, the other agent will question **CQ2**, and so on. There will be 4 questions and answers (8 messages exchanged). After all critical questions being answered, the other agent will concede to A , totalling 12 exchanged messages.

With argumentation schemes being shared by all agents in that society/organisation, and such argumentation schemes making references to components of the organisation, such as roles, authority link, etc., agents are (rationally) able to identify critical questions for which they already have the answer. For example, in the argumentation scheme *Argument from role to know*, the critical question **CQ4**: “*Is ag playing role R ?*” is an information that comes from the organisation, and all agents are aware of that, therefore such question message does not need to be exchanged. Of course, this benefit is inherent from the shared knowledge, and although our approach encourages such shared knowledge, it comes from organisation-based approaches that are typically used in multi-agent systems rather than from our framework. However, in open multi-agent systems, even with agents knowing such information, they could still autonomously question that, overloading the system unnecessarily. In this respect, our approach allows us to specify which agents are able to use such argumentation schemes (which is needed for them to argue), but constraining the use of critical questions which refer to organisational structure/components. For example as in $\langle n_1, permission, ag, use(as_1, cons) \rangle \in \mathcal{NS}$, with n_1 the norm identifier, as_1 the argumentation scheme identifier, the argumentation scheme *Argument from Position to Know*, and $cons$ an exception for not using the critical question **CQ4**. In this case, continuing our example, the worst-case dialogue will have 10 messages exchanged (for that simple protocol), considering a norm-conforming dialogue.

The agents are able to violate the norms in order to benefit themselves or according to their intentions, and the normative system is supposed to apply sanctions when this occurs. This topic is not the subject of this paper, and we argue that normative systems could be efficiently modelled in order to constrain undesired agent behaviour. In such cases, agents will have just norm-conforming argumentation-based dialogues, making/asking only norm-conforming claims and questions. Therefore, the argumentation-based communication will occur just as specified in the Soc^{ARG} model, allowing a complete specification for multi-agents systems able to use argumentation techniques, which are already known for improving and enriching communication in such systems [1, 13, 16, 21, 22, 29]. Such specification allows more “control” over the communication that will occur in such systems. Further, it is possible to improve argumentation-based dialogues by means of high-level constraints for using the argumentation schemes, as well as by the shared knowledge, as argued above.

6 Example

As an example, we use a hospital scenario. The *structural specification* (SS) of our scenario includes roles ($\mathcal{R}$) **doctor**, **nurse** (both extending the role of **employees**) and **patient**. They compose the **hospital** root group specification (rg). The role **doctor** has an authority link towards role **nurse**, i.e., $link_{aut}(\text{doctor}, \text{nurse}) \in \sqsubset$. Further, all roles have a communication link, i.e., $link_{com}(\text{employees}, \text{patient}), link_{com}(\text{patient}, \text{employees}) \in \sqsubset$.

The *functional specification* (FS) has only one scheme with the top-level goal to **cure** the patient, which is decomposed into the parallel goals of **recover_health** and **attend_patient**, where **attend_patient** is decomposed into sequential goals to **examine** and **medicate** the patient. The components mentioned above specify the goals $\mathcal{G}$ and the tree-like structure $\mathcal{S}$ in the *functional specification* of the organisation. The missions $\mathcal{M}$ are associated with the goals, for example the mission **medicate_patient** refers to the goal **medicate**, **examine_patient** to **examine**, and so on.

In order to save space and for simplicity, we consider only the argumentation scheme from *role to know* in the argumentation schemes specification, which was presented in Section 2 and its XML format was listed in Section 4. As described above, the links between the *structural*, *functional*, and *argumentation schemes* is given by the *normative specification*. For our scenario, we have the following normative specification $\mathcal{N}$:

```
<n1,obligation,employees,do(attend_patient)>,
<n2,obligation,doctor,do(examine_patient)>,
<n3,obligation,nurse,do(medicate_patient)>,
<n4,obligation,patient,do(recover_health)>.
<n5,permission,doctor,use(as1,[context(examine_patient)]>>,
<n6,permission,nurse,use(as1,[context(medicate_patient)],
  except(cq1,role_to_know(doctor,Conclusion))>>>),
```

```

⟨n7, permission, patient, use(as1, [context(recover_health),
  except(cq1, role_to_know(doctor, Conclusion)),
  except(cq1, role_to_know(nurse, Conclusion))])⟩.

```

The norms *n1*, *n2*, *n3*, and *n4* impose organisational obligations for the agents playing each role to commit themselves to some organisational goal, while norms *n5*, *n6*, and *n7* give permissions to use a particular argumentation scheme in the organisation, in particular contexts and with some restrictions. For example, a **nurse** can use arguments from *role to know*, but it is supposed not use the critical question *cq1* instantiated with agents playing the role of **doctor**, considering that the **doctor** is in a particular role which could not be questioned to be in a position to know or not about medical knowledge expressed in that environment (the hospital). Of course, the **nurse** may well choose to construct arguments contrary to the **doctor**'s point of view, or criticise premises, but they would not question that the doctor is in a position to know.

7 Related Work

Hübner et al., in [10], propose to extend the MOISE model to a fourth dimension called *dialogic* dimension which is focused in the communication between roles. That work defines the protocols used for communication between roles. The authors argue that communication is one of the main tools that agents have to coordinate their actions at the social level, the specification of the exchanges/communications between roles could be a tool for the regulation of the exchanges between agents. They suggest the specification of communication between roles (which agents adopt in the organisation) at the organisational level. The dialogic dimension, proposed in [10], extends the MOISE model inspired by the theoretical PopOrg model [6, 26].

Boissier et al., in [3], put forward the idea that the normative specification should include also the control of communication modes in the organisation. Therefore, the authors propose to extend the normative organisation model of MOISE in order to specify the interaction modes between agents participating within the organisation. That work aims to allow the multi-agent organisation monitors the interaction between agents, and to make the agents able to reason over the communication modes, similarly to the way they do with norms. The proposal aims to unify a model for interaction called EASI [27] and an organisation model, enriching the MOISE organisation modelling language with a new and independent dimension connected to the other ones by the normative specification.

Reed and Walton, in [24], propose a formal representation and implementation of argumentation schemes in multi-agent systems. The main aims of the paper are: (i) to allow individual agent to reason about arguments, as well as to develop arguments that employ schemes, and (ii) to explore the communication structure that can be built up around those schemes. The authors find it fundamental to demonstrate concrete implementations, showing that the claimed

advantages of schemes can be achieved in practice (while the implementation makes specific choices regarding the development, the formal component guarantees the broader applicability of the approach).

Our work differs from all these. Similarly to Hübner et al. [10] and Boissier et al [3], we use the MOISE model specification as a guide to our formalisation (although as we said any other organisational model could provide the information about roles, goals, social plans, etc. that we need) but more importantly to implement our approach. However, we focus on argumentation schemes, which can be instantiated by agents for reasoning and communication, differently from [10] and [3] that are concerned with usual protocols and communication modes from EASI [27], respectively, but neither is based on argumentation.

Further, our work differs from Reed and Walton’s [24] in that we do not aim at defining particular protocols for argumentation-based dialogues using argumentation schemes, but rather to extend multi-agent development frameworks with the possibility of argumentation schemes being specified in an integrated way with cognitive autonomous agents and complex multi-agent organisations. However, we argue that our framework can support protocols as well (as indeed done by others as discussed above with the argumentation base), perhaps more complex protocols given the links with other dimensions of the system. Combining protocols — for example the protocol from [24] itself — with our framework is indeed one of the future directions of our research.

8 Conclusion

In this work, we introduced the Soc^{ARG} model for multi-agent system specification. The Soc^{ARG} model extends multi-agent specifications with an argumentation-scheme dimension that is strongly connected with the organisational dimension of a multi-agent platform. Our formalisation is inspired by MOISE organisational model [12], which has structural, functional, and normative specifications for multi-agent systems, and is the standard organisational model within the JaCaMo framework [4] that we choose to make the work in practice. However, the idea of referring to roles, groups, goals, social plans, etc. when defining acceptable uses of argumentation schemes can be applied to any organisational model that provides similar abstractions. In the Soc^{ARG} model, the normative specification is extended in order to link the argumentation schemes to the other specifications. This allows the functional, structural, and argumentation-scheme specification to be independent from each other, being linked mostly through the normative specification. Further, we have defined some direct and indirect relations from the argumentation-scheme specification to the other elements of multi-agent organisations and platforms. Also, we have described the benefits our approach brings to multi-agent systems development, the main of them: (i) more control in the argumentation-based communication; and (ii) the possibility to constrain agent’s moves (in dialogues) by means of the normative specification, turning protocols, when norm-conforming conducted, more efficient.

Our view of argumentation schemes in multi-agent system specifications is flexible to allow agents to decide, in an autonomous way, whether to violate or not the norms, in particular norms about respecting the proposed uses of the argument schemes for their interaction. This is an important characteristic, considering the great challenge to manage communication in *open* systems. As future work, we intend to develop applications based on the Soc^{ARG} model, defining different argumentation schemes and analysing the interaction among the arguments instantiated from these different schemes for both reasoning and communication purpose. Also, we intend to define protocols for argumentation-based dialogues based on such approach of using argumentation schemes, which for the best of our knowledge, is an underexplored area and potentially interesting.

Acknowledgements

This research was partially funded by CNPq and CAPES.

References

1. Besnard, P., Hunter, A.: Elements of Argumentation. The MIT Press (2008)
2. Boella, G., van der Torre, L.: Permissions and obligations in hierarchical normative systems. In: Proc. 9th int. conf. on AI and law. pp. 109–118. ACM (2003)
3. Boissier, O., Balbo, F., Badeig, F.: Controlling multi-party interaction within normative multi-agent organizations. In: Coordination, Organizations, Institutions, and Norms in Agent Systems VI, pp. 357–376. Springer (2011)
4. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming 78(6), 747–761 (2013)
5. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology). John Wiley & Sons (2007)
6. Costa, A.C.d.R., Dimuro, G.P.: Introducing social groups and group exchanges in the poporg model. In: AAMAS. pp. 1297–1298. (2009)
7. Dignum, M., et al.: A model for organizational interaction: based on agents, founded in logic. SIKS (2004)
8. Esteva, M., Rosell, B., Rodriguez-Aguilar, J.A., Arcos, J.L.: Ameli: An agent-based middleware for electronic institutions. In: AAMAS. pp. 236–243. (2004)
9. Fagundes, M.S., Meneguzzi, F., Vieira, R., Bordini, R.H.: Interaction patterns in a multi-agent organisation to support shared tasks. In: International and Interdisciplinary Conference on Modeling and Using Context. pp. 364–370. (2013)
10. Hübner, A., Dimuro, G.P., da Rocha Costa, A.C., Mattos, V.: A dialogic dimension for the moise+ organizational model. In: MALLOW (2010)
11. Hübner, J.F., Boissier, O., Bordini, R.H.: A normative programming language for multi-agent organisations. Annals of Mathematics and AI 62(1-2), 27–53 (2011)
12. Hubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. Int. J. of Agent-Oriented Software Engineering 1(3-4), 370–395 (2007)
13. Hussain, A., Toni, F.: On the benefits of argumentation for negotiation-preliminary version. In: Proceedings of 6th European workshop on multi-agent systems (EUMAS-2008) (2008)

14. McBurney, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information* 11, 2002 (2001)
15. McBurney, P., Parsons, S.: Dialogue games in multi-agent systems. *Informal Logic* 22, 2002 (2002)
16. Panisson, A.R., Freitas, A., Schmidt, D., Hilgert, L., Meneguzzi, F., Vieira, R., Bordini, R.H.: Arguing About Task Reallocation Using Ontological Information in Multi-Agent Systems. In: 12th International Workshop on Argumentation in Multiagent Systems (2015)
17. Panisson, A.R., Meneguzzi, F., Vieira, R., Bordini, R.H.: Towards practical argumentation-based dialogues in multi-agent systems. In: IEEE/WIC/ACM International Conference on Intelligent Agent Technology (2015)
18. Panisson, A.R., Bordini, R.H.: Knowledge representation for argumentation in agent-oriented programming languages. *Brazilian Conference on Intelligent Systems, BRACIS*, pages 13–18, 2016.
19. Prakken, H.: An abstract framework for argumentation with structured arguments. *Argument and Computation* 1(2), 93–124 (2011)
20. Prakken, H., Reed, C., Walton, D.N.: Argumentation schemes and burden of proof. In: Workshop Notes of the Fourth Workshop on Computational Models of Natural Argument (2004)
21. Rahwan, I., Ramchurn, S.D., Jennings, N.R., McBurney, P., Parsons, S., Sonenberg, L.: Argumentation-based negotiation. *The Knowledge Engineering Review* 18(04), 343–375 (2003)
22. Rahwan, I., Simari, G.R., van Benthem, J.: *Argumentation in artificial intelligence*, vol. 47. Springer (2009)
23. Reed, C., Rowe, G.: Araucaria: Software for Puzzles in Argument Diagramming and XML. Tech. rep., Department of Applied Computing. University of Dundee Technical Report. (2001)
24. Reed, C., Walton, D.: Towards a formal and implemented model of argumentation schemes in agent communication. *Autonomous Agents and Multi-Agent Systems* 11(2), 173–188 (2005)
25. Ricci, A., Piunti, M., Viroli, M.: Environment programming in multi-agent systems: An artifact-based perspective. *Autonomous Agents and Multi-Agent Systems* 23(2), 158–192 (Sep 2011)
26. da Rocha Costa, A.C., Dimuro, G.P.: A minimal dynamical mas organization model. *Handbook of Research on Multia-Agent Systems: Semantics and Dynamics of Organizational Models* pp. 419–445 (2009)
27. Saunier, J., Balbo, F.: Regulated multi-party communications and context awareness through the environment. *Multiagent and Grid Systems* 5(1), 75 (2009)
28. Schmidt, D., Panisson, A.R., Freitas, A., Bordini, R.H., Meneguzzi, F., Vieira, R.: An ontology-based mobile application for task managing in collaborative groups. In: Florida Artificial Intelligence Research Society Conference (2016)
29. Toniolo, A., Norman, T.J., Sycara, K.: On the benefits of argumentation schemes in deliberative dialogue. In: AAMAS. pp. 1409–1410. (2012)
30. Toulmin, S.E.: *The Uses of Argument*. Cambridge University Press (1958)
31. Walton, D., Reed, C., Macagno, F.: *Argumentation Schemes*. Cambridge University Press (2008)
32. Walton, D.: *Argumentation schemes for presumptive reasoning*. Routledge (1996)
33. Wooldridge, M.: *An introduction to multiagent systems*. John Wiley & Sons (2009)

A Decentralised Approach to Task Allocation Using Blockchain

Tulio L. Basegio*, Regio A. Michelin, Avelino F. Zorzo, and
Rafael H. Bordini

Postgraduate Programme in Computer Science, School of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, RS - Brazil

{tulio.basegio, regio.michelin}@acad.pucrs.br
{avelino.zorzo, rafael.bordini}@pucrs.br

Abstract. One of the challenges in developing multi-robot systems is the design of appropriate coordination strategies in such a way that robots perform their operations efficiently. In particular, efficient coordination requires judicious task allocation. Without appropriate task allocation, the use of multi-robot systems in complex scenarios becomes limited or even unfeasible. Real-world scenarios usually require the use of heterogeneous robots and task fulfillment with different structures, constraints, and degrees of complexity. In such scenarios, decentralised solutions seem to be appropriate for task allocation, since centralised solutions represent a single point of failure for the system. During the allocation process, in decentralised approaches, there are often communication requirements, as participants need to share information. Maintaining data integrity, resilience, and security in data access are some of the important features for this type of solution. In that direction, we propose an architecture for dynamic and decentralised allocation of tasks built on the idea of having communication and coordination in a multi-agent system through a private blockchain.

Keywords: task allocation, multi-robot systems, multi-agent systems, blockchain.

1 Introduction

One of the challenges in developing multi-robot systems today is the design of coordination strategies in such a way that robots perform their operations efficiently [21]. Without such strategies, the use of multi-robot systems in complex scenarios becomes limited or even unfeasible.

An important aspect considered in coordination problems is task allocation [11, 21]. There are several features that should be considered by a mechanism for allocating tasks to multiple robots in real-world scenarios such as considering

* Partly supported by Federal Institute of Rio Grande do Sul (IFRS) – Campus Feliz.

the heterogeneity of robots, the impact of individual variability to assign specific roles to individual robots, and the definition and allocation of different types of tasks. This is particularly true for disasters such as flooding [17].

During a rescue phase in a flooding disaster, teams are called into action to work in tasks such as locating and rescuing victims [13]. Such teams are normally organised by a hierarchy model [15], with individuals playing different roles during a mission. Task fulfillment during the rescue stage poses a number of risks to the teams. Using robots in a coordinated way to help the teams may minimise those risks.

Our work on task allocation has been inspired by the typical tasks in flooding disaster rescue. In particular, we needed an architecture for a dynamic and decentralised task allocation mechanism that takes into account different types of tasks for heterogeneous robot teams, where robots can play various different roles and carry out tasks according to the roles they can play. Although the actual flooding rescue tasks we are dealing with is not the focus of this paper, the architecture we presented here was inspired and is being developed for such a disaster response application, in particular in case of flooding disasters.

In order to manage the information exchanged during the allocation process, our architecture proposes the use of Blockchain Technology [14]. Blockchain is becoming increasingly popular as it provides data integrity, resilience, user confidence, fault-tolerant storage (decentralisation), security, and transparency, among other features. The use of blockchain as a technology to manage information is an innovative aspect of the proposed architecture and seems to be a promising way to deal with issues of consistency, integrity, security, and so on.

The main contribution of our work is therefore an architecture for dynamic and decentralised allocation of tasks built on the idea of having communication and coordination through a private blockchain. The architecture should support a dynamic and decentralised task allocation mechanism that considers different types of tasks to heterogeneous robot teams, where robots can play different roles and carry out tasks according to the roles they play. We use the term agent to refer to the main control software of an individual robot, so our multi-robot system is effectively treated as a multi-agent system.

The paper is organized as follows. Section 2 provides the background on blockchain and task allocation. Section 3 presents the proposed task allocation architecture. Section 4 discusses a particular case study. Section 5 describes related works. Finally, in Section 6 we conclude.

2 Background

2.1 Multi-Robot Task Allocation (MRTA)

Task allocation among multiple robots (and more generally among multiple agents) consists of identifying which robots should perform which tasks in order to achieve cooperatively as many global goals as possible and in the best possible way.

Market-based approaches have been largely studied for use in multi-robot task allocation. In these approaches, the robots are usually designed as self-interested [4] and have an individual utility function which quantifies how much a task contributes to the robots' objective when executed by it. The utility function can combine several factors (such as payoff to be received, the costs incurred, etc.) [6]. The global team utility can be quantified as a combination of the individual utilities. A common mechanism used in market-based approaches are called Auctions [4]. When allocating tasks through auctions, the robots provide bids, which are usually computed based in the utility values. The robot with the highest utility value for each task wins the task. In other words, the robots need to have information about the tasks and share information (bids) with each other.

Tasks: Different type of tasks can be used to address tasks in real-world scenarios, which cannot be adequately represented by only one type of task due to their complex structures and other domain-specific characteristics. In this paper we are considering the following type of tasks defined in [22]:

- Atomic task (AT): a task is atomic if it cannot be decomposed into subtasks.
- Decomposable simple task (DS): a task that can be decomposed into a set of atomic subtasks or other decomposable simple tasks as long as the different decomposed parts have to be carried out by the same robot.
- Compound task (CT): task that can be decomposed into a set of atomic or compound subtasks. When each of the subtasks need to be allocated to a different robot we call it CN task (N subtasks that need exactly N robots). When there are no constraints, the subtasks can be allocated to one up to M robots, where M is the number of subtasks (CM tasks).

2.2 Blockchain

In 2008, Satoshi Nakamoto published a paper presenting an electronic peer-to-peer cash system, called Bitcoin [14]. His proposal is based on removing a third party, allowing two willing parties to transact directly. Bitcoin is the first truly decentralised global currency system, and it is based on hash algorithms and asymmetric cryptography. Since the network is decentralised, it relies on a network of volunteer nodes to collectively implement a replicated ledger. This public ledger tracks all transactions and the balance of all system accounts.

This public ledger, also known as blockchain, is applied in Bitcoin context to avoid a double spending problem, as well as giving publicity to all transactions. The transactions performed in the Bitcoin network are grouped in order to create a block with several transactions. Figure 1 shows a version of a block content. The block is divided into four main structures: **Size** - the size of the block, in bytes; **Header** - is composed of *version*, which is a software/protocol version, *merkle tree root* is a hash of the merkle tree root of block's transaction, *difficulty* is this block target that should be achieved throughout proof-of-work,

previous block hash a hash value from the previous block in the chain, *timestamp* block creation time and a *nonce* which is a counter used for the proof-of-work algorithm; **Transaction counter** identifies how many transactions are stored in the current block and **Content block** where all block transactions are stored.

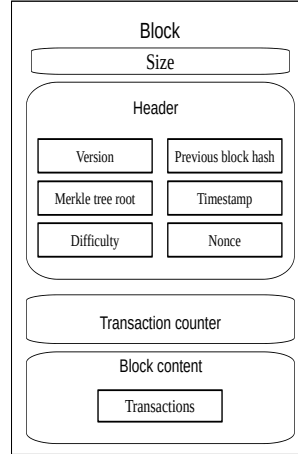


Fig. 1. Blockchain block structure

The previous block hash field, in the block header, is used to create a link between a new block and its predecessor, i.e., the hash of the previous block is stored in the previous block hash field of the new block, thus creating a chain. That is the reason for the public ledger to be called blockchain. The link between blocks is also the way how the historical information has its integrity ensured.

Since the blockchain is decentralised, each node in the Bitcoin network keeps a copy of the ledger where all transactions are stored, thus improving the blockchain resilience. Since each node has a copy of all transactions and every node is able to create a new transaction, a mechanism should be applied to avoid malicious nodes to be able to change information. In order to define which node is able to create a block and insert it into the blockchain, a consensus algorithm is applied. The Bitcoin blockchain uses a proof-of-work consensus algorithm [18].

Proof-of-work (PoW) is an algorithm that produces a block hash value which identifies the block. The operation to produce this piece of information, is very high cost in terms of CPU and power. In contrast to validate the produced information is a very cheap operation. In Bitcoin, the work consists of creating a block hash that is compliant with some rules, for example, the hash must have the N first digits consisting of zeroes. So any client that wants to insert a new block into the blockchain, must change the nonce field in the block header until its block hash value matches the defined zeroes. In order to solve this puzzle, brute-force is used, where the client must repeat the process until they find the solution, leading to a problem related to consuming CPU and power. This led some researchers to identify the use of different consensus algorithm like Proof-of-Stake [23].

The consensus algorithm is a real need in the public blockchain, in order to establish an organized way to the block insertion. This consensus criteria could be softer when running the blockchain in a controlled environment like in a private corporate network. Hardjono [9] in his research proposes a permissioned blockchain applied to the Internet of Things context, where only some nodes (defined by the sensor manufacturer) are able to write data to the blockchain. In contrast, every network node is able to read the information available in the blockchain.

Based on the characteristics of the blockchain we can highlight its attributes: *Data Integrity; Resilience; Decentralisation; Transparency; Immutability.*

3 The Task Allocation Architecture

In this section we describe our architecture for dynamic and decentralised allocation of tasks, which is built on the idea of having communication and coordination through a private blockchain. Considering a market-based task allocation approach where an organisation provides tasks to the agents, the organisation and the agents share information with each other in a dynamic process.

In our architecture, the sharing of information regarding the allocation process, either among agents, or among the organisation and the agents, is performed through the blockchain. The idea is to have blockchain acting as a decentralised database allowing the sharing of information. Due to its decentralised attribute, the blockchain is replicated to every participant, which ensures the architecture resilience.

In order to understand the proposal, first we present a general view of a basic task allocation process considering that the robots are part of an organisation. Next we describe the basic agent and organisation structures used in the architecture. Then we provide a detailed description about the interaction of the organisation and the agents with blockchain. Finally we describe our task allocation mechanism and how it uses blockchain.

3.1 Task Allocation Process – Overview

Figure 2 shows the parts considered in a task allocation process. Initially, we consider the existence of an organisation that is responsible for announcing the tasks that need to be carried out by the agents in a given mission. We use the term agent to refer to the main control software of an individual robot of any kind. The tasks provided by the organisation can be requested by the agents available for the mission. Finally, the environment is the place where agents carry out the tasks.

Regarding the process itself, it is initially considered that an organisation has a set of agents to carry out a mission and that these agents are waiting for the tasks they will be asked to carry out (the agents start executing having no assigned tasks). At a certain moment, the organisation announces a set of tasks.

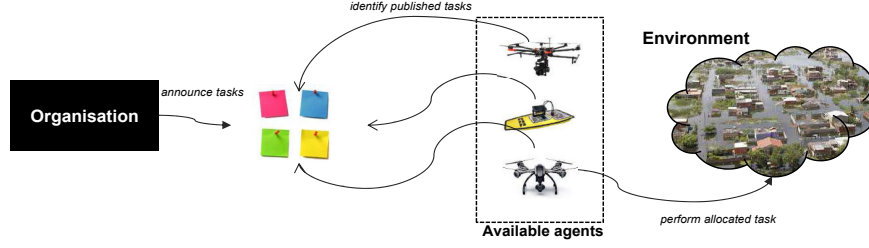


Fig. 2. Task allocation process – Overview.

When a new set of tasks is noticed, the agents begin the allocation process based on the architecture we introduce in this paper.

When the agents finish the task allocation process, those with allocated tasks start to carry them out. At the end of the allocation process, there might be agents without allocated tasks as well as tasks that could not be allocated to any suitable/available agent. Such results depend on the constraints indicated and the features of available agents.

3.2 Basic Agent and Organisation Architecture

Figure 3 shows the main aspects considered in the organisation and agent architecture. Initially, we consider agents based on the Belief-Desire-Intention (BDI) architecture [16]. The BDI architecture was used because it is widely used in several approaches. However, the proposed architecture could be easily integrated with other agent architectures, since its components preserve some independence from the other mechanisms of the agent itself.

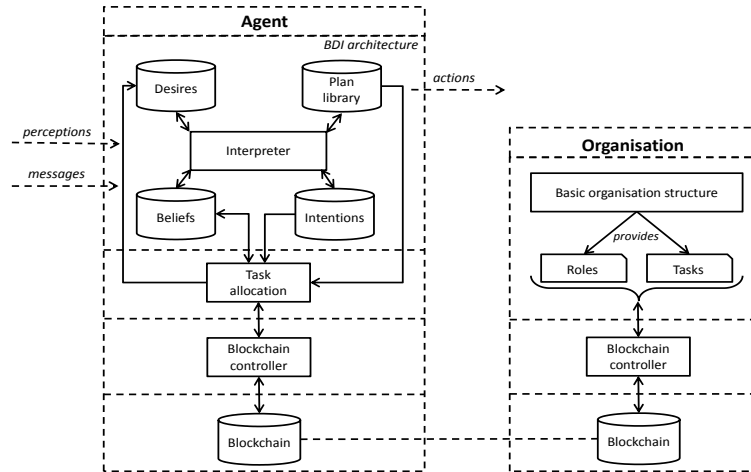


Fig. 3. Basic Organisation and Agent architectures.

Figure 3 shows a (BDI) agent with the addition of the components of our architecture. The task allocation mechanism interacts with the plan library, the belief base, and the intention base in order to have inputs for the allocation process and it also interacts with the desire base by adding new desires (goals) related to the tasks allocated to the agent. There is also an interaction between the task allocation mechanism and the blockchain controller component in order to get shared information and also to share information with other participants in the task allocation process. The blockchain controller is responsible for managing the interactions with the blockchain.

The organisation interacts with its own blockchain controller to add information related to the tasks that need to be carried out by the agents, as well as the available roles in the organisation (with capabilities required by each role).

Simply put, the organisation uses the blockchain to share information about the tasks that need to be carried out and the available roles in the organisation as well as the capabilities needed to play each role. The agents use the blockchain to share information such as their bids for the tasks during the allocation process. A detailed description of the interaction between the organisation and agents through the blockchain controller is described in the next section.

3.3 The Blockchain Controller

Here we introduce the actions which allow organisations and agents to communicate and coordinate through blockchain. To make the description easier we will use the term entities to refer to organisation and agents. The blockchain controller provides a set of actions to allow the entities to manage and share information through the blockchain although not all are currently used in our task allocation process. The proposal is supposed to be generic enough to be useful for other solutions and not just for task allocation. Figure 4 shows these actions, which are shortly described below.

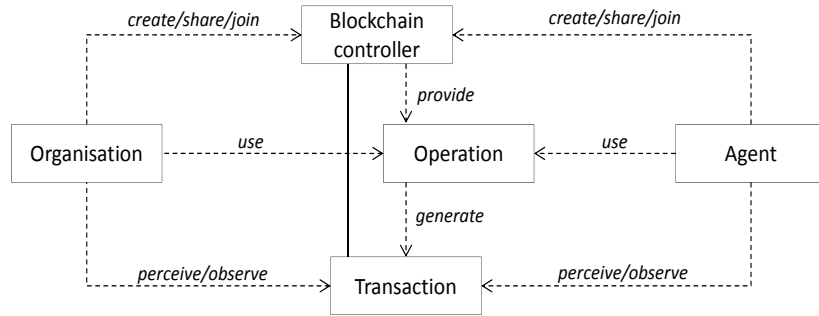


Fig. 4. Conceptual model for the interaction through blockchain.

- **Create/share/join blockchain:** the following actions can be performed by the entities to manage the blockchain.

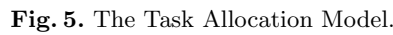
- *createBlockchain*: instantiates a new blockchain for the entity which executed this operation;
 - *shareBlockchain*: allows to share a blockchain with other entities;
 - *joinBlockchain*: allows an entity to join and focus on a blockchain in order to obtain and share information through that blockchain;
 - *stopPerceivingBlockchain*: allows an entity to stop receiving new transactions added to a blockchain. The entity is still able to access the blockchain data and adding new transactions to it.
 - *deleteBlockchain*: removes the blockchain from the entity that executed it.
 - *duplicateBlockchain*: create a private copy of the blockchain for an entity's own use.
- **Use operation**: An operation represents a set of instructions to allow entities to access transaction data. The following operation can be performed by the entities to add new transactions to the blockchain.
 - *insertTransaction*: allows to insert a new transaction into a blockchain.
 - **Perceive/observe transaction**: the transaction represents information shared by some entity.
 - perceiving transaction: every time a new transaction is added to the blockchain the entities will be able to perceive it (entities that are sharing that blockchain). The *stopPerceivingBlockchain* action is used to stop perceiving.

3.4 The Task Allocation Mechanism

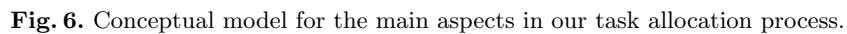
In this section, we describe our task allocation mechanism and how it interacts with the blockchain controller. Each agent in the organisation executes the task allocation mechanism, shown in Figure 5, characterising a decentralised solution.

Simply put, each agent initially perceives through the blockchain controller the tasks that need to be carried out. Based on the perceived tasks the agent identifies, through task allocation mechanism, the tasks it can carry out based on the roles it can play in the organisation, which are also perceived through the blockchain. The agent then identifies the tasks it will try to allocate to itself and calculates its bids for those tasks. The agent then communicate its bids putting that information in the blockchain through the blockchain controller (i.e., executes the *insertTransaction* operation). The bids added to the blockchain will be perceived by all other agents who will then check if some of the bids improve on its own bid for a task it allocated to itself. If that is the case, the agent withdraws that task from the list of its pre-allocated tasks and then checks which task it will bid for next, to replace the task it relinquished. These steps are repeated until all agents agree on the allocation, that is, until the tasks allocated to all agents do not undergo any further modifications.

Figure 6 presents a model with the main concepts of the proposed task allocation mechanism. As shown in the figure, we assume that an agent can have different capabilities that can be related to its type of locomotion (e.g., the possibility of sailing or flying) or even to the resources available to the agent (i.e.,



the robot's payload such as cameras, sensors, etc.). An agent may play one or more roles. The roles are defined by the organisation the agents belong to and each role is related to a set of capabilities that an agent needs to have in order to play that role. The organisation is also responsible for defining the tasks that are required in a given mission.



As described in Section 2, a task can be atomic, simple decomposable or compound. Compound tasks have a list of subtasks that can be, in turn, atomic or compound (this way it is possible to create a complex hierarchy of tasks,

which increases the applicability of this approach to different scenarios). Similar structures are also possible for decomposable simple tasks. Atomic tasks will not be directly considered in the proposed mechanism (they will be indirectly regarded as subtasks of both compound tasks and simple decomposable tasks). The tasks require one or more agents able to play particular roles to carry them out. That is, agents may not be able to carry out certain tasks if they cannot play the required role. We consider that each agent has a maximum number of tasks that can be allocated to itself. This constraint may be related, for example, to the amount of energy (fuel) available to the robot. This may vary among robots as well as it may vary while the tasks are being carried out.

A version of the task allocation mechanism without the use of blockchain technology has been implemented and runs in BDI agents developed in JaCaMo framework. The algorithms that constitute the mechanism are detailed in [1], where initial results, obtained from Monte-Carlo simulations, demonstrate that the proposed mechanism seems to scale well, as well as provides near-optimal allocations.

Figure 7 shows the average results of simulations using our framework, varying the number of agents (from 5 to 35) when allocating 24 tasks. Figure 7(a) shows that the performance of the proposed solution improves and is closer to the optimal solution (i.e., 100%) as we increase the number of agents. Figure 7(b) shows a small standard deviation in all simulations (comparing with the optimal solutions). The average execution time of these simulations was 4 seconds for each simulation with 5 agents up to 15 seconds with 35 agents. In [1], tests with up to 60 tasks were also performed with similar results.

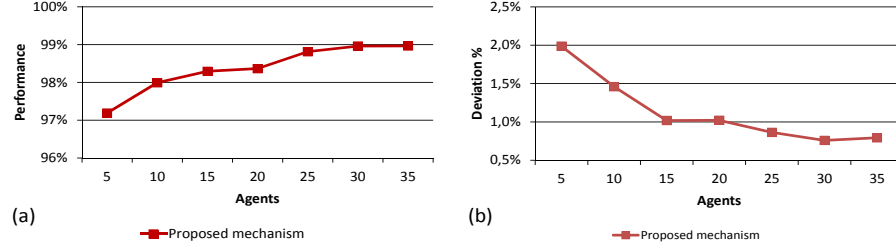


Fig. 7. Simulation results by varying the number of agents.

4 Case Study: Allocating Tasks in a Flooding Scenario

This section describes the use of our architecture through a case study on a flooding scenario. We chose this scenario because it represents a real multi-robot application scenario with several constraints that need to be considered by the software architecture, such as the heterogeneity of the robots, the impact of individual variability to assign specific roles and the accomplishment of different types of tasks. According to Murphy, there are several tasks that can be performed or assisted by robots during flood disasters [12]. One of the key tasks

to be accomplished is to obtain situational awareness of the affected region [13]. This task involves mapping the affected areas, where the robots are allocated to obtain images of a region. In order to accomplish this task, in this case study, the robot needs to have flight capability and a camera to obtain the images. Another important task in flood disasters is the collection of water samples for analysis, such as verifying the level of water contamination [17]. To perform this task, the robot must have navigation capability and be able to collect water samples. In this case study we will focus on these two specific tasks.

Consider an organisation that needs to work on a flooding disaster by performing tasks such as mapping areas and collecting water samples for analyses. The organisation has three robots available to help in those tasks: one USV (Unmanned Surface Vehicle) and two UAVs (Unmanned Aerial Vehicle), which we will call respectively USV1, UAV1, and UAV2. USV1 has sailing capability and resources to collect water samples while UAV1 and UAV2 have flying capabilities and cameras to take images. The predicates below represent the information each robot has about itself.

USV1: *capabilities*([*sail*, *waterCollector*])
 UAV1: *capabilities*([*fly*, *camera*])
 UAV2: *capabilities*([*fly*, *camera*])

In the organisation, there are two possible roles to be played: mapper and collector. In order to play the mapper role, a robot must have the capability to fly and must have a camera to take pictures. For the collector role, robots must have the capability to navigate and resource to collect water samples. The predicates below represent this information.

role(*mapper*, [*fly*, *camera*])
role(*collector*, [*sail*, *waterCollector*])

Considering the flooding scenario, the organisation has defined the following tasks to be performed. The predicates below are composed of (and in this particular order): the task identifier, the task name, the region where the task is to be performed, and the role a robot needs to perform that task.

task(*t1*, *collectWater*, *regionA*, *collector*)
task(*t2*, *takeImage*, *regionA*, *mapper*)
task(*t3*, *takeImage*, *regionB*, *mapper*)

Considering the above, we describe now how the task allocation process works using blockchain. First, the organisation creates a new blockchain using the *createBlockchain* action. This action returns an identifier for the created blockchain. To facilitate the explanation, consider that the identifier returned by the action is *bcTaskAlloc1*.

The organisation then can share the blockchain with the available robots using the action *shareBlockchain*, using as parameter the name of the blockchain being shared and a list of the robots with which it should be shared. The following action shares the blockchain *bcTaskAlloc1* with the robots in the organisation.

```
shareBlockchain(bcTaskAlloc1, [USV1, UAV1, UAV2]);
```

After that, each robot will have a copy of the shared blockchain *bcTaskAlloc1*. The organisation then uses the *insertTransaction* operation to share role and task information with the robots through the blockchain. The following operations are used to share information about the mapper and collector roles.

```
insertTransaction(role(mapper, [fly, camera]));
insertTransaction(role(collector, [sail, waterCollector]));
```

Block 1 in Figure 8 represents the role information added to the blockchain by the organisation. The blockchain technology is responsible for synchronising the role information with the copy of the blockchain available in the robots. Once the robots' blockchains are updated with the new transactions, the blockchain controller in each robot will generate a percept to the robot about the new information. Each robot is now able to identify which roles it can play in the organisation. USV1 identifies it can play role collector while UAV1 and UAV2 identify they can only play the mapper role.

The following operations are used to share information about the tasks.

```
insertTransaction(task(t1, collectWater, regionA, collector));
insertTransaction(task(t2, takeImage, regionA, mapper));
insertTransaction(task(t3, takeImage, regionB, mapper));
```

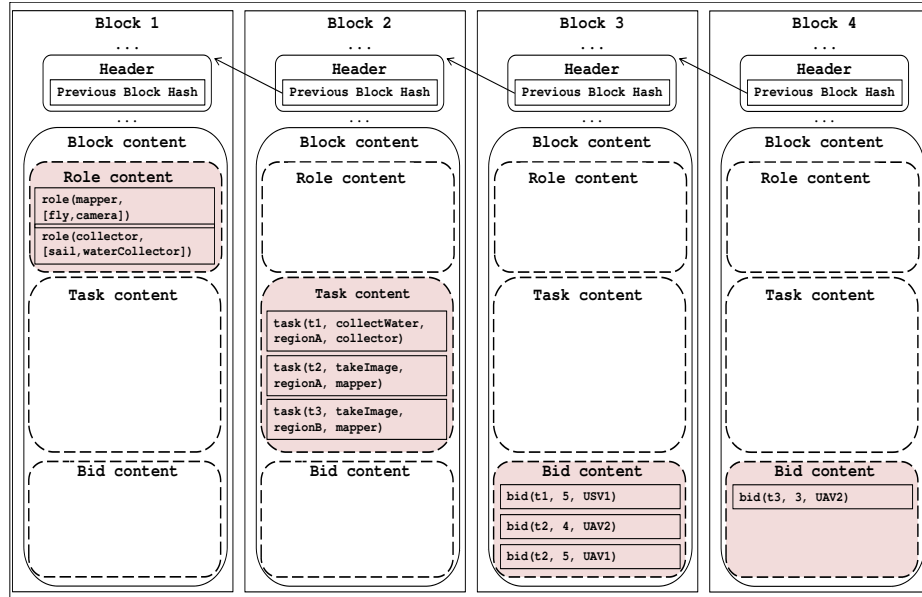


Fig. 8. Example of our blockchain content.

Block 2 in Figure 8 represents the task information added to the blockchain by the organisation. Again, the blockchain is responsible for synchronising the

information with the blockchain in the robots, and the blockchain controller in each robot will generate percepts for the robot when the new information is added to the blockchain. Each robot is now able to identify the tasks available in the organisation and the ones it can bid for based on the roles it can play. USV1 realises it can bid only for task t1, while UAV1 and UAV2 realise they can bid for tasks t2 and t3. With information about the roles and the tasks, the robots are able to start the allocation process.

We also assume that all robots start bidding at the same time, each one inserting a new transaction within their copy of the blockchain. The following operations represent the bids from each robot. In order to calculate a bid, each robot uses inputs from the plan library as well as the belief and intention bases.

USV1 operation: *insertTransaction*(*bid*(t1, 5, USV1));
 UAV1 operation: *insertTransaction*(*bid*(t2, 5, UAV1));
 UAV2 operation: *insertTransaction*(*bid*(t2, 4, UAV2));

Each operation will add the transaction to the blockchain in the respective robot. The blockchain in each robot is responsible for synchronising the information about the bids with the blockchain in the other robots and also in the organisation blockchain. Block 3 in Figure 8 represents the bidding information after the synchronisation. For each new transaction updated in the robots blockchain, the blockchain controller will generate a percept to the robot (information about the current bids). Each robot is now able to identify if it has lost to another robot some of the tasks for which it placed bid (i.e., when another robot has provided a higher bid). In our scenario, UAV1 and UAV2 provided bids for the same task t2. Since UAV1 bid for task t2 is higher than UAV2 bid for the same task, the UAV2 will provide a bid to another task as specified in the following operation.

UAV2 operation: *insertTransaction*(*bid*(t3, 3, UAV2));

The operation will add the transaction to the blockchain in the respective robot which again will be synchronised with the other robots and the organisation, generating percepts to the robots. Block 4 in Figure 8 represents the bidding information after the synchronisation.

Assume that robots agreed on the allocated tasks and the allocation finished after this last bid. Thus, USV1 won the bid for task t1, UAV1 won for task t2, and UAV3 won for task t3. The information about the allocated tasks will be added to the belief base, as well as new desires (goals) will be added to the desire base of each robot, so that they can start the execution of the allocated tasks with the support of their architectural components.

5 Related work

There are several works on task allocation, some of them aim at allocating an initial set of tasks to a set of robots as in [3, 11, 19], while others focus on allocating tasks that arise during the execution of other tasks as in [20].

Regarding the tasks, most of the solutions available in the literature, such as the ones presented in Gernert [7] and Settini [19], focus only on atomic tasks, unlike our proposal, which comprises other types of tasks as well. Das [3] and Luo [11] are examples of work that deal with subtasks in some way.

In Gernert [7], the authors propose a decentralised mechanism for task allocation along with an architecture that focuses on exploring disaster scenarios. However, in the solution, as in many others, the robots can carry out any task, i.e. heterogeneity and capabilities are not considered. There are also works like Settini [19] and Das [3], where heterogeneous robots and their capabilities are considered in the task allocation.

In Gunn [8] is described a framework for allocating new tasks discovered by robots in the missions. It proposes the use of heterogeneous robots organised in teams. The robot with the best computational resources is responsible for the allocation process. Thus, it could be said of that there is still a single point of failure within each team, so it is not exactly a decentralised solution like ours.

To the best of our knowledge, there are no studies using blockchain in the task allocation process either for multi-robot or multi-agent systems. In this way, we introduce as related work the use of blockchain in other applications.

Blockchain technology was first applied to the Bitcoin currency in 2008, but since then it was used in several other different applications. Ferrer's research [5] describes possibilities in the use of blockchain to improve three aspects related to a swarm robotic system: security through blockchain digital signature, public and private keys; distributed decision making where the blockchain can be applied to handle collective map building, obstacle avoidance and reach agreements; swarm control behavior differentiation considering linking several blockchains in a hierarchical manner, which would allow robotic swarm agents to act differently according to the blockchain being used.

In a different context, Lee [10] uses blockchain to control the manufacturer firmware version installed in its devices. The idea is to create a blockchain where all devices are connected through a peer-to-peer network. Through this blockchain each device can check its firmware version and, once identified the need for update, it requests to the node that has the most recent version.

As Lee proposes a blockchain change, focused on control and distribution of firmware version, Bogner's [2] research uses the blockchain applied in a Ethereum cryptocurrency. In his work the blockchain is used to handle device renting in Internet of Things context. For example if a user wants to rent a bike in a station, he just performs an operation transferring the rent value, and when the transfer is persisted in the blockchain, the station releases the bike.

In our research we identified that blockchain could be applied to solve problems in areas that are not directly related to currency. That motivates the current work in order to bring blockchain technology to handle the task allocation problem in the multi-agent context.

6 Conclusions

In this paper, we have presented an architecture for dynamic and decentralised allocation of tasks built on the idea of having communication and coordination in a multi-agent system through blockchain. The architecture was inspired by and is being developed for such an application in a multi-institution project funded by the government to address disaster response, in particular in case of flooding. Considering that real-world scenarios like flooding disasters typically require the use of heterogeneous robots and task fulfillment with different complexities and structures, our architecture takes into account the allocation of different types of tasks for heterogeneous robot teams, where robots can play different roles and carry out tasks according to their capabilities.

Our architecture takes advantage of the blockchain technology which is a promising way to deal with issues such as consistency, data integrity, resilience, security, decentralisation, transparency, and immutability. For example, using blockchain in our architecture allows all the participants to share the same knowledge about the task allocation process. Since all information about the allocation is stored in the blockchain, new robots can be added to the process at any time. The organisation (or some agent) can share the blockchain with the new robots, which will have access to the data previously stored in the blockchain. That allows the robots to synchronise their knowledge about the allocation process and so to participate in it. Security is also an important aspect for task allocation in flooding scenarios since the robots can be target of threats and attacks and that may impact in the search and rescue of victims. Blockchain uses an encryption scheme based on asymmetric cryptography which ensures the security to the information stored.

The use of blockchain as a technology to manage task allocation information is an innovative aspect of our architecture and seems to be useful also in other problems in multi-agent systems.

References

1. Basegio, T.L., Bordini, R.H.: An Algorithm for Allocating Structured Tasks in Multi-Robot Scenarios. In: Proceedings of the 11th International KES Conference, KES-AMSTA 2017. Vilamoura, Algarve, Portugal. To appear.
2. Bogner, A., Chanson, M., Meeuw, A.: A Decentralised Sharing App Running a Smart Contract on the Ethereum Blockchain. In: Proceedings of the 6th International Conference on the Internet of Things. pp 177–178 (2016)
3. Das, G.P., McGinnity, T.M., Coleman, S.A.: Simultaneous allocations of multiple tightly-coupled multi-robot tasks to coalitions of heterogeneous robots. In: Robotics and Biomimetics (ROBIO), 2014 IEEE Int. Conf. on. pp. 1198–1204 (2014)
4. Dias, M.B., Zlot, R., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE* 94(7), 1257–1270 (July 2006)
5. Ferrer, E.: The blockchain: a new framework for robotic swarm systems. {Online}, Available in <https://arxiv.org/pdf/1608.00695.pdf> (2016)
6. Gerkey, B., Mataric, M.: A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research* 23(9), 939–954 (2004)

7. Gernert, B., Schildt, S., Wolf, L., Zeise, B., Fritsche, P., Wagner, B., Fiosins, M., Manesh, R.S., Mller, J.P.: An interdisciplinary approach to autonomous team-based exploration in disaster scenarios. In: 2014 IEEE International Symposium on Safety, Security, and Rescue Robotics (2014). pp. 1–8 (Oct 2014)
8. Gunn, T., Anderson, J.: Effective task allocation for evolving multi-robot teams in dangerous environments. In: Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM Int. Joint Conf. on. vol. 2, pp. 231–238 (2013)
9. Hardjono, T., Smith, Ned.: Cloud-Based Commissioning of Constrained Devices Using Permissioned Blockchains. In: Proceedings of the 2Nd ACM International Workshop on IoT Privacy, Trust, and Security. pp. 29–36 (2016)
10. Lee, B., Lee, J.: Blockchain-based secure firmware update for embedded devices in an Internet of Things environment. In: The Journal of Supercomputing. pp 1–16 (2016)
11. Luo, L., Chakraborty, N., Sycara, K.: Provably-good distributed algorithm for constrained multi-robot task assignment for grouped tasks. *IEEE Transactions on Robotics* 31(1), 19–30 (2015)
12. Murphy, R.R.: *Disaster Robotics*. The MIT Press (2014)
13. Murphy, R.R., Tadokoro, S., Nardi, D., Jacoff, A., Fiorini, P., Choset, H., Erkmn, A.M.: *Search and Rescue Robotics*, pp. 1151–1173. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
14. Nakamoto, S.: *Bitcoin: A Peer-to-Peer Electronic Cash System*. {Online}, Available in <https://bitcoin.org/bitcoin.pdf> (2008)
15. Ramchurn, S.D., Huynh, T.D., Ikuno, Y., Flann, J., Wu, F., Moreau, L., Jennings, N.R., Fischer, J.E., Jiang, W., Rodden, T., Simpson, E., Reece, S., Roberts, S.J.: Hac-er: A disaster response system based on human-agent collectives. In: Int. Conf. on Aut. Agents and Multiagent Systems. pp. 533–541. AAMAS '15, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2015)
16. Rao, A.S., Georgeff, M.P.: Bdi agents: From theory to practice. In: Proceedings of the Int. Conference on Multi-Agent Systems (ICMAS-95). pp. 312–319 (1995)
17. Scerri, P.; Kannan, B.; Velagapudi, P.; Macarthur, K.; Stone, P.; Taylor, M.; Dolan, J.; Farinelli, A.; Chapman, A.; Dias, B.; and Kantor, G. 2012. Flood Disaster Mitigation: A Real-World Challenge Problem for Multi-agent Unmanned Surface Vehicles. Springer Berlin Heidelberg. 252–269.
18. Sleiman, M. D., Lauf, A. P., Yampolskiy, R.: Bitcoin Message: Data Insertion on a Proof-of-Work Cryptocurrency System. In: 2015 International Conference on Cyberworlds (CW). pp. 332–336 (2015)
19. Settini, A., Pallottino, L.: A subgradient based algorithm for distributed task assignment for heterogeneous mobile robots. In: 52nd IEEE Conference on Decision and Control. pp. 3665–3670 (2013)
20. Urakawa, K., Sugawara, T.: Task allocation method combining reorganization of agent networks and resource estimation in unknown environments. In: Innovative Computing Technology (INTECH), 2013 Third Int. Conf. on. pp. 383–388 (2013)
21. Yan, Z., Jouandeau, N., Cherif, A.A.: A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems* 10 (2013)
22. Zlot, R.M.: *An Auction-Based Approach to Complex Task Allocation for Multi-robot Teams*. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave (2006)
23. Watanabe, H., Fujimura, S., Nakadaira, A., Miyazaki, Y., Akutsu, A., Kishigami, J.: Blockchain contract: Securing a blockchain applied to smart contracts. In: 2016 IEEE Int. Conference on Consumer Electronics (ICCE). pp. 467–468 (2016)

A Case Study of the Development of an Agent-based Simulation in the Traffic Signal Control Domain using an MDD Approach

Fernando Santos¹², Ingrid Nunes¹³, and Ana L. C. Bazzan¹

¹ Instituto de Informática, UFRGS, Brazil,

² Departamento de Engenharia de Software, UDESC, Brazil

³ TU Dortmund, Germany

{fsantos,ingridnunes,bazzan}@inf.ufrgs.br

Abstract. Model-driven development (MDD) is an approach for supporting the development of software systems, in which high-level modeling artifacts drive the production of low-level, time and effort-consuming artifacts, such as source code. Previous work on its use showed that it significantly increases development productivity, given that the effort is focused on the business domain instead of technical issues. However, MDD was exploited in the context of agent-based development in a limited way, and previous work has not shown real evidences of the benefits that MDD promotes in this context. In this paper, we explore the use of MDD in agent-based modeling and simulation. We conducted a case study in the traffic signal control domain, in which autonomous agents are in charge of managing traffic light indicators to optimize traffic flow. We propose an MDD approach, composed of a modeling language, and model-to-code transformations for producing runnable simulations. An empirical study provides evidence that our MDD approach reduces the effort to develop agent-based simulations.

Keywords: Agent-based Modeling and Simulation, Model-driven Development, Development Effort, Traffic Signal Control

1 Introduction

Building simulations in which there are multiple interacting agents situated in an environment is a challenging task, which has been widely investigated in the context of agent-based modeling and simulation (ABMS). In ABMS, *modeling* refers to the task of specifying a model that represents a target system (e.g., a traffic system), while *simulation* is the execution of such model over a timeline, running agent behaviors and interactions repeatedly. During the design and implementation of agent-based simulations, different *roles* with distinct expertise must interact and communicate [7], and this is often a barrier to a successful development. Usually, the domain expertise is concentrated on the *thematician* and *modeler* roles, while the technical expertise (on ABMS and its simulation

platforms) is concentrated on the *computer scientist* and *programmer* roles. Researchers have already argued about the importance of tools and building blocks that enable domain experts themselves to build agent-based simulations [14].

Many alternatives have been proposed for the design and implementation of agent-based simulations. There are methodologies, languages, and programming platforms [12, 19] that focus exclusively on MAS aspects such as agents, interactions, and the environment. However, those MAS alternatives do not cover simulation aspects, such as the creation and initialization of entities and agents. In turn, agent-based simulation platforms—e.g., NetLogo [31]—are alternatives that consider such simulation aspects. These platforms, however, demand previous expertise in ABMS or in programming, thus hindering domain experts to build agent-based simulations themselves. The demand for technical expertise, such as programming, would significantly reduce with the provision of a solution that enables creating simulations by means of ABMS-related building blocks.

An approach towards this direction is *model-driven development* (MDD), whose goal is to express domain concepts effectively. MDD makes domain concepts (e.g., adaptation) available for modeling by means of a *domain-specific language* (DSL) [22]. Traditional software development approaches, in contrast, often only provide concepts from the solution space (e.g., programming statements and abstract types). Transformation engines and code generators reduce or suppress the development effort when using MDD [25]. There are MDD approaches focused on ABMS, such as metamodels [11, 18] and methodologies to identify the domain concerns that should be included in a modeling language [9, 15]. Nevertheless, they are limited to particular MDD aspects and are abstract in the sense of modeling only high-level ABMS-concepts, leaving much left to be developed in specific applications. Moreover, with such individual specific contributions, there is a lack of evidence of the real benefits that MDD approaches promote, covering the design and implementation of agent-based simulations.

In this paper, we address these issues by further exploring the use of MDD in the context of ABMS and empirically assessing the benefits it provides. We present a case study of the development of an MDD approach in the *adaptive traffic signal control* (ATSC) domain, in which we (i) performed a domain analysis; (ii) designed a metamodel and DSL; and (iii) developed model-to-code transformations. Moreover, we empirically assessed the gains obtained with our approach, in terms of development effort. Our approach is focused on a specific domain, as opposed to existing work that focused on ABMS in general, given that previous work on MDD showed that the more specific the application domain, the higher the chance of success [16]. We describe how the domain concepts were identified through a domain analysis process, which is the foundation of our ABMS metamodel. For the domain analysis, we considered existing simulations, which adopt either adaptation or reinforcement learning techniques, to keep the scope limited due to the aforementioned reasons. Because these techniques are often complex to develop, we provide a DSL that gives building blocks for modeling and automated transformations for code generation. Although we focused on a specific domain, we identified many abstractions, included in our

metamodel and DSL, that potentially can be adopted in other domains. We conducted an empirical study that evaluated the reduction of the effort required to develop agent-based simulations. We concluded that our approach reduces the development effort by 60-86%, in terms of produced software assets.

Specifically, our case study provides the following contributions: (i) a domain analysis method, which is applicable to other domains; (ii) a metamodel, DSL, and code generator, which support the development of applications in the investigated domain and potentially in similar domains; and (iii) an evaluation, which concretely demonstrates the benefits of MDD for ABMS.

2 Background on MDD

Software models are widely used in software development. While in traditional model-based software engineering they are used as guidance to source code implementation, in model-driven development (MDD) [25] models are equivalent to source code, because they are used to automatically generate it. Models are thus *first-class citizens*, and the development is driven by modeling artifacts [27].

Making domain abstractions available for modeling is fundamental in MDD. Previous work on the use of MDD in domains such as automotive manufacturing, mobile devices, and telecommunications, showed that productivity increases because the modeling effort is focused on *domain concerns* instead of programming statements [26]. Such effective MDD approaches are built upon a domain-specific language and its underlying metamodel; and source code generators [27].

A domain-specific language (DSL) is a modeling or programming language designed for a particular domain, trading generality for expressiveness [27]. An MDD approach relies on a DSL to build models [28]. The key element of a DSL—and of an MDD approach as a whole—is its underlying metamodel. Such metamodel defines meta-entities and relationships between them within a domain. The DSL simplifies the metamodel instantiation by providing building blocks for representing model elements and thus a more productive environment. In order to build a DSL, it is crucial to perform a domain analysis [28]. It produces as result a *domain model*, which describes domain concepts that should be provided by the modeling language so as to increase its expressiveness. Such domain model is the basis for creating the DSL metamodel. A DSL is described by its syntax, and semantics. The *abstract syntax* is specified in the DSL metamodel, while the *concrete syntax* specifies the notation used to represent instances of metamodel elements (e.g., textual, or graphical symbols). Finally, semantics specify the meaning of such symbols and their constraints.

Source code generators automate the creation of low-level software artifacts (e.g., lines of code). Manually creating such artifacts is an effort-consuming task and requires technical expertise. Moreover, pieces of code for recurrent structures and concepts are often implemented repeatedly. By putting these pieces of code into code generators, an MDD approach increases productivity in software development. Code generators are built upon production rules that describe what code statements are generated for each metamodel element.

3 MDD for ABMS

As previously stated, in this work we focus specifically on the adaptive traffic signal control (ATSC) domain. This is due to the trade-off to be made in MDD (generality *vs.* expressiveness), and there is evidence that the more specific the application domain, the higher the chance of success [16]. We next detail our case study, first describing the conducted domain analysis, then presenting the resulting metamodel, modeling language, and model-to-code transformations.

3.1 Domain Analysis

Any source of explicit or implicit domain knowledge can be considered in the domain analysis [22]. In this work, we essentially used *existing agent-based simulations* in the ATSC domain. Derived domain concepts were further validated using domain expert knowledge. Consequently, our domain analysis was performed using a proposed *bottom-up* approach, to reduce the bias of individual experts' views while identifying domain concepts. However, we consulted ABMS experts to select ATSC simulations for analysis. As result, we used as source work on self-organizing [5, 10] and reinforcement learning [21, 23, 30].

To guide the domain analysis, we considered existing work in this context, focused on MAS. Hassan et al. [15] proposed a process to guide the identification and formalization of domain concepts. A similar initiative was proposed by Garro and Russo [9]. In spite of providing valuable guidelines for identifying agents, interactions, and environmental entities, these processes do not provide support for identifying the simulation aspects of ABMS, such as temporal extent, initialization, and observation. To overcome this issue, we followed the Overview, Design concepts, and Details (ODD) protocol [13]. This protocol guides the identification and specification of most of the key characteristics of a simulation, such as its structure, agent capabilities (e.g., learning) and its underlying processes. Thus, our domain analysis method is composed of the following steps.

- Step 1* **Concept Preliminary List.** A list of MAS-related concepts is identified using the existing ATSC simulations (e.g., agents and the environment), following the steps of Hassan et al. [15] and Garro and Russo [9].
- Step 2* **ODD-based Refinement.** The ODD protocol is used to refine identified concepts, considering simulation aspects and additional agent capabilities such as learning.
- Step 3* **Concept Abstractions.** Identified concepts, already refined based on the ODD protocol, are analyzed in order to find the essence behind them. The analysis considers recurrent characteristics and behaviors. Similar concepts are abstracted as a single, essential concept, or generalized to a parent concept. During the analysis, a table of abstractions is built, containing the domain terminology and concepts, and generalizations.
- Step 4* **Domain Modeling.** The table of abstractions is used to build the domain model, which is described in the next section.

After performing the steps described above, which produced intermediate documentation¹, we identified different recurrent concepts in this domain, which are: environment, agents and their capabilities or perceptions, vehicles and demand, adaptation and learning. These are detailed as follows.

The *environment* of an ATSC simulation is a traffic network, which is composed of links and nodes that represent road lanes and intersections, respectively. Such traffic network is often provided as separated files, such as open street maps. A *traffic signal controller (TSC)* is an agent in charge of managing traffic light indicators (red, yellow, and green). TSC agents are created at each intersection and their perception is related to their incoming and outgoing lanes, such as the queue length and throughput. Additionally, it is assumed that TSC agents are able to perceive vehicle-related data, such as speed, and travel/waiting time.

The design of a TSC agent involves a set of concepts from the traffic control domain, which comprises our basic domain terminology and is shown in Figure 1. A *stage* describes a particular set of allowed traffic movements for vehicles in the lanes of the intersection. For each TSC, many stages are defined to regulate the traffic flow. A *phase* is a period of time in which the indicators of the corresponding stage are green, allowing the traffic flow. In addition to the green interval, a phase can specify a change interval (yellow) and a clearance interval (in which all the indicators of the intersection are red before activating the next phase). A *cycle* corresponds to a complete rotation through all the stages. Consequently, the duration of a cycle corresponds to the sum of its phase intervals. Finally, a *plan* is a set of phases plus the sequence in which they are activated. An offset can be defined for a plan and corresponds to a period of time in which the activation of the first phase is postponed. In order to evaluate the effectiveness of TSC agents, *vehicles* are created in the traffic network during the simulation according to a traffic demand.

Regarding adaptation, each existing simulation considered deals with it in its own particular way. Wiering [30], Oliveira and Bazzan [23], and Mannion et al. [21] described the use of reinforcement learning by TSC agents. Each work adopted a particular state representation and reward function, and the policy learned by TSC agents is related to stage, phase, or plan selection. The work on self-organizing traffic lights [5, 10] introduced a set of adaptive criteria, based on traffic conditions, which drive TSC agent decisions. These introduced adaptation approaches share a common characteristic: TSCs have designer-specified fixed plans, which are used as comparison baselines. Next, we describe how these observations and concepts were abstracted by following steps 3 and 4 of our domain analysis method, to derive the domain model.

3.2 Metamodel and Modeling Language

We next describe the key element of our MDD approach: a metamodel that is in accordance with the concepts identified in our domain model. The DSL concrete syntax and semantics of the modeling language are then detailed, and the semantics is only informally described, due to space restrictions.

¹<http://www.inf.ufrgs.br/prosoft/resources/dsl4abms/2017emas/>

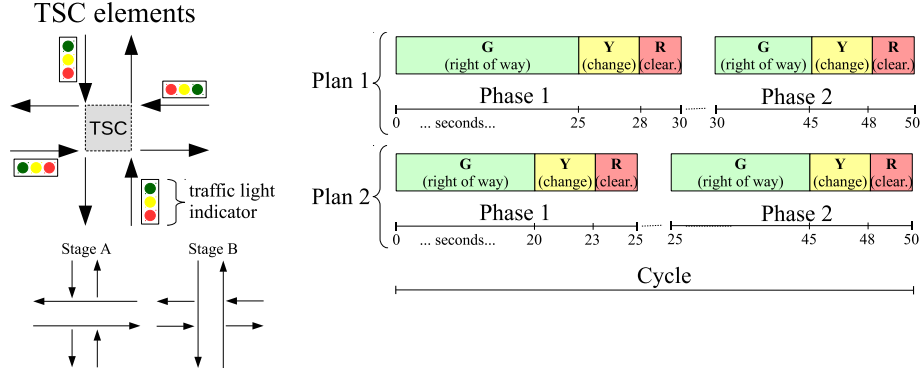


Fig. 1. ATSC Domain Terminology and Concepts.

Metamodel. The metamodel, partially presented in Figure 2, was built following the abstraction step of our domain analysis method. It is constructed upon the EMF Ecore² meta-metamodel. The basic elements of the metamodel are entity, agent, and attribute.³ An *entity* represents objects existing in a simulation (such as lanes and intersections) that has *attributes*. An *agent* is a particular kind of entity that has *agent capabilities*. The idea of agents as entities with attributes are in fact part of almost all agent-based metamodels, such as those presented by Bernon et al. [1]. However, given that we are following a bottom-up approach, existing metamodels are not used as a start point to avoid introduction of bias. Moreover, it can potentially lead to an overly complex metamodel. From the ABMS perspective, such bottom-up specification can provide valuable insights on building an effective MDD approach.

From the abstraction step, we observed that a TSC is, in its essence, an agent that has a *flow control capability* for regulating the flow of a set of streams. Consequently, it has a set of flow regulators for each known stream. These regulators can be seen as *actuators* of the agent. Regulators can be in certain states, such as green or red, open or closed. These states are abstracted to *actuator states*. The actuator state that is automatically activated when no other state is active is the default state. Actuators can be grouped into *actuator groups*, on which all activate the same state simultaneously. Consequently, a group is considered a single actuator because both are considered *actuatable* devices. Each actuator is identified by a number that relates the actuator to the corresponding stream (i.e., a regulator 0 is in charge of regulating the stream 0, and so on). Streams are represented as an agent attribute with cardinality greater than one (i.e., a collection of streams). Additionally, we consider actuators mutually exclusive: only one actuator or group can assume a non-default state at a given moment; all others remain in the default state. Figure 3 illustrates how domain concepts are abstracted to these metamodel elements. In the bottom, there are concepts

²<http://www.eclipse.org/modeling/emf/>

³Element names have the MM prefix in Figure 2 due to the technology used to specify them.

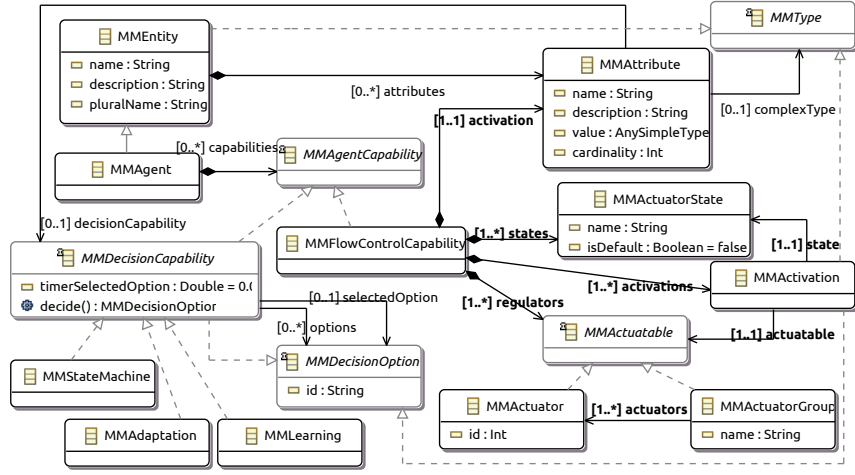


Fig. 2. ABMS Metamodel built based on Existing ATSC Simulations.

that were identified in steps 1 and 2 of the domain analysis. Dashed arrows point to the metamodel elements that abstract such concepts. As can be seen, a TSC agent is abstracted to an *agent* and a *flow control capability*. Each traffic signal indicator is an *actuator*, and red/yellow/green states are *actuator states*. Stages are abstracted to *actuator groups* given that the set of actuators that must activate simultaneously is obtained from stage definitions.

The behavior associated with a flow control capability is related to the management of its actuators. At each timestep, an agent must decide which pair (actuator, state) is selected for activation. Such pair is represented as an agent attribute whose type is *activation*. With this attribute, a decision capability must be associated, whose options are all the possible activation pairs. A *decision capability* represents a decision policy that must be chosen from many *decision options*. From the domain analysis, we identified three types of decision capabilities: state machines, adaptation, and learning. It is important to notice that decision capabilities can also be decision options for another decision capability. For example, a learning capability, whose decision options are state machines.

A *state machine* represents a fixed decision policy. It is composed of states, which are pairs of decision option and transitions. To represent state machines, we adopt a subset of the Unified Modeling Language (UML) Statemachines metamodel.⁴ An adaptation capability represents an adaptive decision policy. There is an *adaptation criterion* that describes which option should be selected among those available—the one that meets the criteria. Last, a learning capability allows an agent to learn a decision policy. We consider a reinforcement learning capability, with which agents learn through experience. As agents act on the

⁴<http://www.omg.org/spec/UML/2.5/>

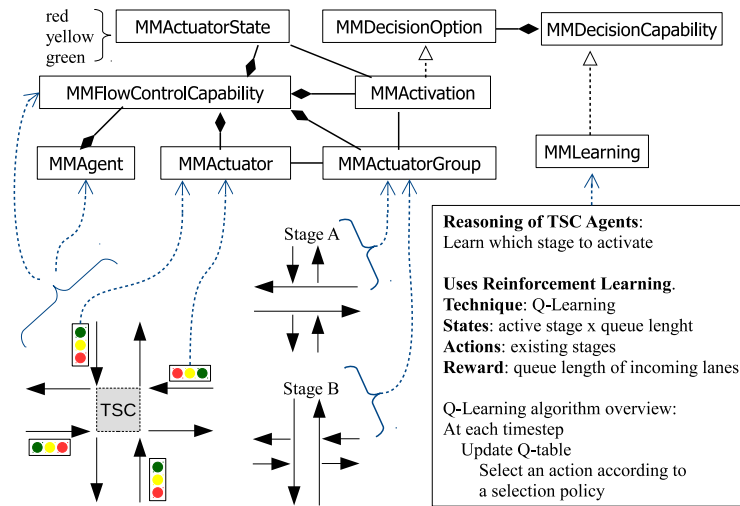


Fig. 3. Example of Concept Abstractions.

environment, they receive a reward signal based on the outcomes of previously states and actions. As illustration, Figure 3 also includes the abstraction of these learning concepts into a learning element. The reasoning of TSC agents is based on reinforcement learning, more specifically on the Q-Learning technique [29].

Language Concrete Syntax. The goal of our language is to ease the modeling of an agent-based simulation by providing building-blocks for its elements. We followed the method of Strembeck and Zdun [28] for building our DSL, whose metamodel was presented in the previous section. We adopted a graphical representation to reduce the effort required to identify model elements and their relationships. Figure 4 shows an overview of the concrete syntax of our language. To illustrate all the language features, the depicted model shows a (partial) combination of existing simulations: stages, phases, and plans as proposed by Oliveira and Bazzan [23]; and the learning technique adopted by Mannion et al. [21].

Entities and agents are represented using a box with at least three sections: the entity or agent name; how it is created; and its attributes. Such representation is inspired by the UML class diagram. The notation used to represent attributes consists of its name and cardinality, followed by the definition of how it is initialized and updated during the simulation. For example, the initialization of the *streams* attribute of the *Traffic Signal Controller* agent is by means of an *expression* that refers to the incoming links of the traffic node; while the updating of the *activation* attribute is related to a decision capability. The entity creation is specified in the second section of their boxes, by means of creational strategies. For the *Traffic Signal Controller* agent, a *designer defined* strategy is selected, in which the designer must specify the quantity of instances and their location. Expressions can be used to specify both of them. We assume

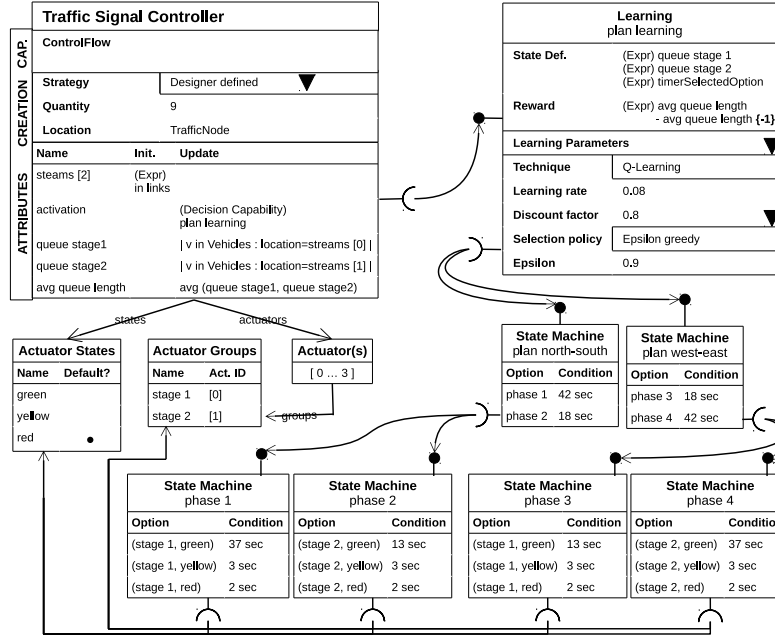


Fig. 4. Example of the Language Concrete Syntax.

that expressions follow a math notation, such as functional or set notations. Metaelements for other creational strategies (i.e., files), and other alternatives for initialization and updating, are included in the complete metamodel, which is not detailed in this paper due to space restrictions.

In addition to the previous sections, an agent box has an additional section to specify its capabilities. Each capability can have a set of predefined attributes that are incorporated into the agent. For example, *Flow Control Capability* incorporates the previously described attributes *streams* and *activation* into the agent. The additional specifications required by an agent capability are represented as elements connected to the agent by an arrow. For the *Flow Control Capability*, its actuator states and actuator groups are represented as tables. Each row in the *Actuator States* table represents a state, and the default state is the one whose *default* column is checked. Similarly, rows in the *Actuator Groups* table represents actuator groups. Each group shown in Figure 4 has only one actuator, because they are used specifically to exemplify group definitions. In such cases, the designer can specify no group and use the identifier of actuators to specify activations, given that one actuator is implicitly created for each stream.

Decision capabilities are represented as boxes whose content describes the elements required by each capability type. The top section of the box presents the corresponding capability type (*State Machine*, *Adaptation*, or *Learning*) and its name for further references. Each capability box includes two connectors, inspired by the UML component diagram. The connector with a semicircle rep-

resents that the decision capability requires a set of options, i.e. the decision input. The connector with a filled circle, in turn, represents the decision output, i.e. the selected option. The input provided to a decision capability is represented as a connection between its semicircle connector and any model element that represent decision options, which include actuators, their states and groups, and the output of other decision capabilities. In the model shown in Figure 4, the input of the state machine *phase 1* is the cartesian product of its input connections—actuator states and groups. The output of state machines *phase 1* and *phase 2* are the input of state machine *plan 1*.

For a state machine capability, states are represented as a list of *options* and their transition *conditions*. From all the options available, the designer can select only those that are relevant to the state machine. From the simulations considered in the domain analysis, we assume that the state machine states are activated sequentially. Consequently, each condition specifies the situation that triggers the transition to the state right below it—except for the last state, whose transition is to the first state. A transition condition is specified as an *expression*, which can specify a time interval, in addition to any agent attribute or the current state of the machine and its timer.

Regarding learning capability, its state definition and reward are represented separately from the learning parameters. The state is represented as a list of expressions. For example, the state definition of the *plan learning* capability consists of three expressions that refer to agent attributes. The reward is represented as a single expression. In the *plan learning* example, the reward is defined as the difference between the queue length in the current and previous timesteps. The ability to consider attribute values from previous timesteps in expressions is an additional abstraction incorporated in our metamodel. The adopted notation is to specify the absolute or relative timestep of the attribute enclosed in braces. The learning parameters section presents the selected reinforcement learning technique and their specific parameters. The value of these parameters can be specified statically, or they can refer to expressions or model parameters.

Finally, the representation of an adaptation capability is similar to those of learning and state machines. It introduces only a section for specifying the adaptation criteria as an expression. The model of Figure 4 does not include an adaptation capability; however, its representation is shown in the complete concrete syntax description available elsewhere.¹

3.3 Model-to-code Transformations

Our metamodel and DSL provide the support needed to model agent-based simulations. However, support for code generation is fundamental to reduce the effort to develop them. In order to show that it is possible to generate runnable simulations from our metamodel, and thus exploit the benefits of an MDD approach, we specified model-to-code transformations to generate code for NetLogo [31], a popular agent-based simulation platform.

Model-to-code transformations are performed through the use of production rules, which transform instantiated concepts of our metamodel to NetLogo code

Table 1. Production Rules (Partial View).

Production Rule	Transformation to NetLogo code
agent type	For each <i>MMAgent</i> → breed
agent attributes	For each <i>MMAttribute</i> → breed-own For each capability required data structure (e.g., Q-table) → breed-own
agent capabilities	For each <i>MMAgentCapability</i> → corresponding capability rule
rl capability	For each <i>MMReinforcementLearning</i> → qlearning init. qlearning init. qlearning reward def. qlearning updt. qtable qlearning decision
qlearning reward def.	For the <i>MMExpression</i> reward → reporter , which evaluates the reward expression and returns its value
qlearning init.	For each element of <i>MMLearningStateDefinition</i> → set statements for setting up states For each <i>MMDecisionOption</i> → set statements for setting up actions
qlearnig updt. qtable	executes reward reporter to compute its value set statements for updating the Q-table considering Q-Learning update rule: $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ where: <i>Q</i> is the Q-table <i>s, s'</i> are the current and resulting states <i>a, a'</i> are the current and resulting actions <i>r</i> is the reward α and γ are learning parameters.
qlearning decision	reporter , which selects and reports an action according to a selection policy. For a ϵ -greedy policy, it would be $\text{argmax}_a Q(s, a)$ with probability $1 - \epsilon$, and a random action with probability ϵ .

statements and blocks. Due to space restrictions, we are not able to present all production rules needed for our approach. However, Table 1 illustrates some of the rules for transformation of agents, their attributes and capabilities. Rules related to the reinforcement learning capability, more specifically for the Q-Learning [29] technique, are also shown. The production rule column indicates the rule name which, when applied, is transformed into the content presented in the transformation column. The meaning of NetLogo statements shown in this column is as follows: **breed** and **breed-own** statements are used to declare an agent type and their attributes, respectively; **reporter** is used to declare a procedure; and **set** is the assignment statement.

Production rules were implemented as templates using the Xpand⁵ template language. Each template describes source code that is generated for its corresponding metamodel element. Generated simulations were verified to assert that it produces the expected behavior.

4 Evaluation

We conducted an empirical study to evaluate the *effort* to develop an agent-based simulation using our approach. As discussed, existing MDD alternatives cover only conceptual ABMS-modeling and, consequently, models must be implemented using existing agent-based simulation platforms or general purpose programming languages to have a runnable simulation. We thus compared our approach to alternatives with which a runnable simulation can be developed.

To select the simulations for our study, we adopted the following criteria: source code availability; coverage of the decision capabilities of our metamodel; and use of NetLogo, if possible, given that our MDD approach generates code

⁵<http://www.eclipse.org/modeling/m2t/?project=xpand>

for this platform. As result, one simulation was selected for each decision capability: (i) fixed traffic signal plans, covering *state machine*; (ii) self-organizing traffic lights [10], covering the aspect of *adaptation*; and (iii) minimization of the number of stopped vehicles [23], covering *learning*. The first two simulations are available for NetLogo^{6,7}, while the last for the ITSUMO simulation platform⁸.

To measure the effort to develop the existing source code, we used the objective metrics proposed in a framework [4] for evaluating languages for multiagent systems. In this framework, the number of manual or automatically produced software artifacts is an objective metric of the development effort. Manual or generated lines of code (LoCs) are examples of these software artifacts [4]. This can be used to measure development effort, as evidenced by many software cost estimation models, such as COCOMO 2.0 [3], which uses size metrics (e.g. LoC) to estimate required person-months and calendar months. Our study considered only code dedicated to TSCs and their behaviors. Comments and code block delimiters were ignored.

To evaluate the effort to develop an agent-based simulation using our MDD approach, we modeled these existing simulations using our DSL and we generated its corresponding source code using our transformation engine. Given that our DSL uses a graphical representation, we used the *atomic model element* (AME) measure unit [2] for a fair comparison. An AME is a visual modeling element that is equivalent to a LoC. To classify a modeling element of our DSL as an AME, we adapted a generous estimation rule [2], based on the elements of UML diagrams. The following elements were counted as AMEs: model parameters; entity or agent boxes; attributes and their corresponding initialization and update; agent creation strategy and each of its parameters; agent capabilities, their options and parameters; actuator states and groups; and connectors of agent capabilities.

Obtained results are shown in Table 2, separated by decision capability. Columns indicate: (i) **AMEs**: the number of specified AMEs; (ii) **MLoCs**: the number of manually written lines of code; (iii) **Effort**: the sum of AMEs and MLoCs, which is the effort to develop the corresponding simulation; and (iv) **GLoCs**: the number of lines of code generated from AMEs.

As can be seen, our approach reduces the effort for all the three simulations. The development effort was reduced by **60.00%** (with 116 GLoCs), **81.93%** (103 GLoCs), and **85.60%** (297 GLoCs), respectively. For all simulations, TSC-related code was fully automatically generated. From the complete source code required to run a simulation, approximately 50% is automatically generated. Vehicle and visualization-related code, which are out of the scope of our approach, corresponds to the remaining 50%.

It is also possible to observe that the total amount of lines of code (MLoCs + GLoCs) produced by our MDD approach for TCSs and their behavior is greater than the total amount of lines of code in the existing NetLogo implementation, for all simulations. Given that our metamodel considers domain independent ab-

⁶<http://ccl.northwestern.edu/netlogo/models/TrafficGrid>

⁷<http://turing.iimas.unam.mx/~cgg/sos/SOTL/SOTL.html>

⁸<http://www.inf.ufrgs.br/maslab/traffic/itsumo/>

Table 2. Effort Comparison.

Decision Capability	Simulation	AMEs	MLoCs	Effort*	GLoCs
State Machine (Fixed Plan)	NetLogo	1	34	35	1
	Our MDD approach	14	0	14	116
Adaptation (Self-organizing Traffic Lights)	NetLogo	1	82	83	1
	Our MDD approach	15	0	15	103
Learning (Reinforcement Learning)	ITSUMO	0	257	257	0
	Our MDD approach	37	0	37	297

*Effort = AMEs + MLoCs

stractions (e.g., state machines and other decision capabilities), generated code contains additional statements to implement these abstractions. Existing implementations, in turn, are built upon application-specific abstractions that may not generalize to other domains. Nevertheless, GLoCs are not considered in the development effort metric, because no human effort is required to produce them.

Even if an MDD approach provided code generation for all the elements of a simulation model, it would provide little benefit if the effort to specify such model using the provided DSL is similar to or higher than implementing it from scratch. Based on the presented results, this is not the case of our MDD approach. Given that an AME is equivalent to a MLoC, the sum of AMEs and MLoCs produced when using our MDD approach is always lower than when using NetLogo. Although the number of AMEs produced using our MDD approach is the highest, it is lower than the number of MLoCs produced using NetLogo, leading to a reduction in the combined effort. Furthermore, the rule adopted for counting AMEs led to a fine-grained and platform independent evaluation. Therefore, a few elements of our DSL were counted as AMEs, but the effort to specify them is not equivalent to, but lower than, a line of code (e.g., parameters of a learning technique). As a consequence, the evaluation favored implemented simulations, giving us a stronger confidence of that our approach *reduces the development effort* of agent-based simulations in our domain.

5 Related Work

In this section, we discuss work related to our MDD approach. Kardas [17] reviewed a selection of model-driven approaches for MAS. Although there are MDD alternatives that provide modeling languages or source code generation, the author argues that their efficiency and practicability are under debate since the amount and quality of the automatically generated MAS components appear to be insufficient. In most situations, such code is generated only at the template level, and a significant amount of code needs to be completed manually. Additionally, most of the MDD approaches reviewed do not include empirical evaluation [17]. There is a lack of evaluations that go beyond demonstrating the use or feasibility of these approaches and thus lacking real evidences of their concrete benefits. Furthermore, it is important to notice that these alternatives are focused on multiagent models, and thus simulation aspects are uncovered.

Existing MDD approaches for ABMS focus on particular aspects of MDD. The AMASON [18] metamodel covers basic structures and dynamics of agent-based simulations. The MAIA [11] metamodel captures social concepts such as norms and roles. Ribino *et al.* [24] propose a conceptual metamodel to be used as a guideline and concept repository for designing simulations. Overall, these metamodels support only abstract ABMS-concepts, leaving much left to be developed in specific applications. In the IODA methodology [20], behaviors are encoded in a interaction matrix, which can be seen as a DSL for specifying the simulation dynamics. However, there are no available coarse-grained simulation building blocks and thus complex interactions must be specified from scratch. Our MDD approach, in turn, supports concrete concepts and their recurrent characteristics and capabilities (e.g., control flow and decision capabilities) identified using a domain analysis that considered existing simulations.

Transformations for code generation were considered in MDA4ABMS [8]. This work provides a light, task-based metamodel, but no DSL is provided. Instead, the use of UML activity diagrams is proposed for modeling, complemented by guidelines for transforming these diagrams into code artifacts. However, human intervention is required to drive the transformations (e.g., to specify which tasks should be executed at a simulation step). Our MDD approach provides automated code generation that produces runnable simulations.

Finally, MDD alternatives in the context of agent-based traffic simulations were already considered [6]. Their metamodel is based on general, agent-related concepts such as tasks, goals, and facts. To specify a sophisticated behavior (i.e., a learning capability), several elements must be instantiated for modeling it. In our MDD approach with its DSL, such sophisticated structures were reduced to their essence and incorporated in the metamodel as ready-to-use building blocks, reducing the burden of simulation development.

6 Conclusion

Developing agent-based simulations is a challenging task, because of a heterogeneous group of involved roles. Model-driven engineering allows focusing on domain concerns while hiding implementation details. Previous work adopted MDD in the context of agent-based modeling and simulation in a limited way, and lack real evidences of its promoted benefits.

In this paper, we explored the use of this approach in the ABMS context by means of a case study in the adaptive traffic signal control domain. Narrowing the domain is important because in MDD there is a trade-off between generality and expressiveness. As result, we provided a metamodel, domain-specific language and code generation for agent-based simulations in this investigated domain. The work is founded on a domain analysis performed in a disciplined way, using existing agent-based simulations. Steps of our domain analysis allowed us to identify the concepts added to our metamodel. These concepts were identified in our case study and, therefore, we have evidence that they are suitable to our particular investigated domain. Nevertheless, concepts, such as the

flow regulator agent, are present in other similar domains, and can be potentially reused. Examples of such domains are the distribution of provisions to relief centers in a disaster simulation, or the regulation of the throughput of links in a data network. Further studies must be performed to validate this.

Our evaluation showed that our approach reduces 60-86% of the development effort. These obtained results provide evidence that our approach gives helpful support for developing agent-based simulations in our domain and that the conducted domain analysis method is effective.

Our long term goal is to use MDD to allow people with little or no ABMS expertise to build agent-based simulations. Further work must be conducted towards this goal. First, experiments with humans must be conducted in order to evaluate subjective aspects, such as usability and comprehensibility. Moreover, specifically in our domain, other simulation techniques and aspects, left out of the scope such as alternative learning models and communication, should be incorporated to our metamodel.

Acknowledgments. This work is supported by funds from CNPq ref. 442582 and 305062, CAPES 7619-15-4, and Alexander von Humboldt BRA 1184533 HFSTCAPES-P.

References

1. C. Bernon, M. Cossentino, M.-P. Gleizes, P. Turci, and F. Zambonelli. A study of some multi-agent meta-models. In *AOSE*, pages 62–77, 2004.
2. J. Bettin. Measuring the potential of domain-specific modeling techniques. In *OOPSLA*, pages 39–44, Seattle, Washington, 2002.
3. B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future software life cycle processes: COCOMO 2.0. *Annals of Software Engineering*, 1(1):57–94, 1995.
4. M. Challenger, G. Kardas, and B. Tekinerdogan. A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, pages 1–41, 2015.
5. S.-B. Cools, C. Gershenson, and B. D’Hooghe. Self-organizing traffic lights: A realistic simulation. In *Advances in Applied Self-Organizing Systems*, pages 45–55. Springer, London, 2013.
6. A. Fernández-Isabel and R. Fuentes-Fernández. Analysis of intelligent transportation systems using model-driven simulations. *Sensors*, 15(6):14116–14141, 2015.
7. J. M. Galán, L. R. Izquierdo, S. S. Izquierdo, J. I. Santos, R. del Olmo, A. López-Paredes, and B. Edmonds. Errors and artefacts in agent-based modelling. *JASSS*, 12(1):1, 2009.
8. A. Garro, F. Parisi, and W. Russo. A process based on the model-driven architecture to enable the definition of platform-independent simulation models. In *SIMULTECH*, pages 113–129. 2013.
9. A. Garro and W. Russo. easyABMS: A domain-expert oriented methodology for agent-based modeling and simulation. *Simulation Modelling Practice and Theory*, 18(10):1453–1467, 2010.
10. C. Gershenson. Self-organizing traffic lights. *Complex Systems*, 16(1):29–53, 2005.

11. A. Ghorbani, P. Bots, V. Dignum, and G. Dijkema. Maia: a framework for developing agent-based social simulations. *Journal of Artificial Societies and Social Simulation*, 16(2):9, 2013.
12. J. J. Gómez-Sanz and R. Fuentes-Fernández. Understanding agent-oriented software engineering methodologies. *The Knowledge Engineering Review*, 30:375–393, 9 2015.
13. V. Grimm, U. Berger, D. L. DeAngelis, J. G. Polhill, J. Giske, and S. F. Railsback. The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768, 2010.
14. L. Hamill. Agent-based modelling: The next 15 years. *Journal of Artificial Societies and Social Simulation*, 13(4):7, 2010.
15. S. Hassan, R. Fuentes-Fernandez, J. M. Galan, A. Lopez-Paredes, and J. Pavon. Reducing the modeling gap: On the use of metamodels in agent-based simulation. In *ESSA*, pages 1–13, 2009.
16. J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *ICSI*, pages 633–642, 2011.
17. G. Kardas. Model-driven development of multiagent systems: a survey and evaluation. *The Knowledge Engineering Review*, 28(04):479–503, 2013.
18. F. Klügl and P. Davidsson. Amason: Abstract meta-model for agent-based simulation. In *MATES*, pages 101–114. 2013.
19. K. Kravari and N. Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015.
20. Y. Kubera, P. Mathieu, and S. Picault. Interaction-oriented agent simulations: From theory to implementation. In *ECAI*, pages 383–387, 2008.
21. P. Mannion, J. Duggan, and E. Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In *Autonomic Road Transport Support Systems*, pages 47–66. Springer, 2016.
22. M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
23. D. de Oliveira and A. L. C. Bazzan. Multiagent learning on traffic lights control: effects of using shared information. In *Multi-Agent Systems for Traffic and Transportation*, pages 307–321. IGI Global, 2009.
24. P. Ribino, V. Seidita, C. Lodato, S. Lopes, and M. Cossentino. Common and domain-specific metamodel elements for problem description in simulation problems. In *FedCSIS*, pages 1467–1476, 2014.
25. D. Schmidt. Model-driven engineering. *Computer-(IEEE Computer Society)*, 39(2):25–31, feb 2006.
26. J. Sprinkle, M. Mernik, J. P. Tolvanen, and D. Spinellis. Guest editors’ introduction: What kinds of nails need a domain-specific hammer? *IEEE Software*, 26(4):15–18, July 2009.
27. T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2006.
28. M. Strembeck and U. Zdun. An approach for the systematic development of domain-specific languages. *Software: Practice and Experience*, 39(15):1253–1292, 2009.
29. C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.
30. M. Wiering. Multi-agent reinforcement learning for traffic light control. In *ICML*, pages 1151–1158, 2000.
31. U. Wilensky. NetLogo, 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

Towards a MAS Product Line Engineering Approach

Dounia Boufedji¹³, Zahia Guessoum¹², Anarosa Brandão⁴, Tewfik Ziadi¹, and Aicha Mokhtari³

¹ LIP6, UPMC Paris 06 University, France

² CReSTIC, Reims Champagne Ardenne University, France

³ RIIMA, USTHB Sciences and Technology University, Algeria

⁴ Computing Engineering&Digital Systems Department, São Paulo University, Brazil

Abstract. It is our claim that the adoption of software engineering reuse techniques can leverage MAS development, mostly when we consider similar applications belonging to the same domain. MAS-Product Line (MAS-PL) raises as an interesting approach that uses Software Product Line Engineering (SPLE) techniques and AOSE to manage the commonalities (similarities) and variabilities (differences) of such MAS applications. Although MAS present specific characteristics that could be considered when describing the system variability, existing work on MAS-PL is devoted to deal with MAS variability considering only domain-specific issues. Moreover, the adoption of variability models such as feature models should be considered for describing both Generic and Specific MAS variability. We propose a MAS-PL approach to address the aforementioned issues by representing Generic MAS variability according to MAS concepts such as agents, environment, interaction and organization, and Specific MAS variability according to a specific application domain. We evaluate the approach by deriving a family of agents that perform jobs in the Multi-Agent Contest environment.

Keywords: Software Product Line Engineering, Software Engineering Reuse, Feature Model, Variability.

1 Introduction

Multi-Agent Systems (MAS) provide an interesting approach for developing software systems in several domains such as resource and information management, process control and simulation of complex systems. Nevertheless, since engineering MAS is a complex task that is not entirely controlled by a process which is accepted by the software industry, MAS are still not part of the mainstream of enterprise application development [18]. In this paper, we propose to introduce a new MAS-PL approach to improve the reuse during MAS development. AOSE provides several MAS templates and reuse patterns to facilitate MAS development [11, 19] and speed up the MAS adoption in software industry. However, most existing AOSE approaches are not suited to the development of similar

applications (a.k.a MAS families). These kinds of applications present similarities (i.e. commonalities) and differences (i.e. variabilities). Indeed, managing such applications remains a difficult task because even if the core architecture is reusable, the variability management is mainly achieved by making code changes.

As the facts mentioned above lead to a considerable waste of time, cost and effort, and make the MAS implementation a difficult task, it is important to provide a solution based on the idea of capitalizing the MAS implementation expertise. Thus, managing MAS variability in MAS families is a key solution to such a capitalization, and has become one of the main challenges in AOSE. SPLE comes out as an interesting solution to manage MAS variability at different levels such as in design models, implementation and so on. It provides a solution to speed up industrial adoption of MAS, and leverages such an adoption [24].

Several MAS-PL approaches have been proposed [4, 12, 22, 23] to apply SPL concepts to MAS development. Those MAS-PL approaches introduce the notion of variability for a reuse issue within a MAS family. They are often built as an extension of existing methods. So, they can be easily adopted by the users of those methods. However, most existing MAS-PL approaches introduce complex notations to specify variability at domain engineering. Moreover, the addressed variability concerns only specific domains.

We believe that a MAS-PL approach should address the variability not only concerning a specific application domain. It should also refer to a generic domain which concerns MAS concepts emerging from MAS meta-models and tools. We thus propose a new MAS-PL approach that follows the general SPLE Framework [1]. This approach relies on two types of features (resp. two types of reusable artifacts): the *Generic MAS* features (resp. artifacts) and the *Specific MAS* features (resp. artifacts). This will concretely serve to promote reuse in MAS families independently of a specific application domain. Those features result from a refinement process that is based on a variability analysis of distinct domains. *Generic MAS* variability analysis scopes the domain of existing MAS methods, meta-models, architectures and tools, while *Specific MAS* variability analysis scopes a specific application domain.

This paper is organized as follows: Section 2 presents a background, dealing with MAS-PL related work and motivations. Section 3 gives an overview of our MAS-PL approach; while Section 4 and Section 5 give more details about the approach and illustrate it with simple examples of the Multi-Agent Contest case study. Section 6 describes and discusses some results about the evaluation of our approach through the Multi-Agent Contest product line. Finally, Section 7 summarizes the contributions and proposes some perspectives for future work.

2 Background and Related Work

2.1 Software Product Line Engineering framework

SPLE represents one of the most interesting paradigm in software reuse and development. It reconciles both production and standardization with customization

in software engineering, and it considerably reduces development cost, time and effort. The general SPLE framework proposed by Apel et al. [1] is represented in Fig. 1 with its two levels: Domain and Application Engineering. The framework includes four activities: domain analysis, domain implementation, requirement analysis, and product derivation.

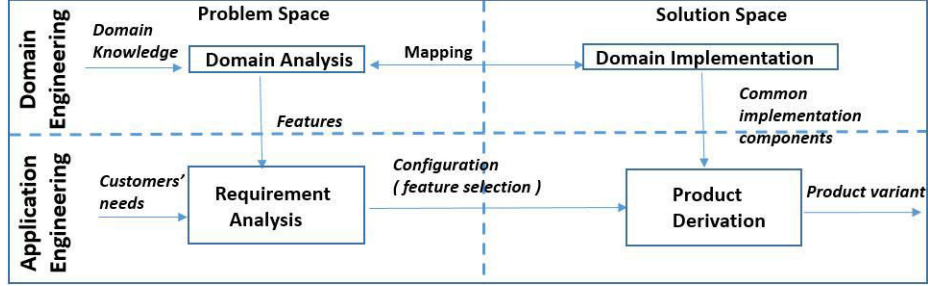


Fig. 1. SPLE Framework and its main activities. [1]

Domain Analysis allows to *scope* the domain (which products should be covered by the product line), and to specify the relevant features that should be implemented as reusable artifacts. The results of domain analysis are usually documented in a Feature Model (FM) [1]. The FM describes features' relationships (parent-child) in a hierarchical tree. Features express the commonality and variability among the products within a product line (see examples in Fig. 3).

Domain Implementation allows to develop reusable artifacts that correspond to identified features. There are many kinds of relevant artifacts in SPLE (including implementation, test, and documentation artifacts) [1].

Requirement Analysis considers a user's requirements to produce a customized configuration, by selecting the desired features.

Product Derivation aims at deriving the product according to the configuration provided by the requirement analysis.

2.2 Related Work

MAS-PL approaches emerged from the idea of applying SPLE approaches into AOSE ones. The first efforts on MAS-PL arose on the mid of the 2000's first decade. Peña et al. [24] argue that both AOSE and SPLE approaches are based on similar concepts in the first activities of domain engineering. For instance, both of them use models in the domain analysis activity: SPLE uses feature models and AOSE uses MAS meta-models. Unlike AOSE, SPLE covers common and variable features analysis of the software family. Most existing AOSE approaches do not consider implementation activity, while SPLE also relies on implementing reusable assets.

MAS-PL approaches differ according to the SPLE phases they cover: Domain and Application Engineering. Most approaches propose to extend AOSE methods by integrating SPLE concepts and techniques.

In this context, Dehlinger et al. [12] propose Gaia-PL to extend GAIA. They provide requirement specification pattern to capture changing design configuration (variation points) in agents and potential reuse of requirement specification. However they did not provide any detail about the domain implementation activity. Nunes et al. [22, 23] propose to extend PASSI [10], to cover the whole development process from requirements to code. They endow PASSI's UML models with stereotypes to model and document agent variability, and they propose implementation guidelines to help MAS developers. Moreover, they propose the modularization of the fine-grained variability (agent architecture) allowing a better specificity of the features [21]. For instance, they introduce decomposition of the goals that gives more specific plans. Although the decomposition of goals ensures alternative or optional features, reuse is only possible for a specific domain application.

Peña et al. [25] propose to enrich MaCMAS (Methodology for analyzing Complex Multi-Agent Systems) with software product lines to model and evolve MAS. They use UML to model a MAS-PL and focus on building the core architecture (common features). MaCMAS captures views of the system at different abstraction levels. The core architecture of the system is represented by a traceability model, and a set of role models. The model is evolved with variations and constraints. They specify the commonality and variability in a Feature Model.

MAS-PL approaches that cover the Application Engineering phase propose mainly MAS derivation approaches that extend derivation tools like in [9] where they propose to use multi-level models to support the configuration knowledge specification and automatic product derivation of MAS-PL.

Among MAS-PL approaches that cover both SPLE phases, SelfStarMAS [3] proposes a process for the development of self-adaptive agents in Internet of Things (IoT). This approach was extended by a Dynamic SPL based approach, that presents the advantage of behaviour agent adaptation at runtime [4]. The approach is interesting, but remains specific to IoT domain.

Even though many approaches have been proposed, some limitations can be clearly identified:

- The use of multiple complex and heavyweight notations to compose models prevent their adoption;
- Reusing feature models is unfeasible since the specified variability is domain-specific in all approaches;
- Most approaches focus on the domain analysis activity and do not give details on the domain implementation.

To overcome those limitations, we propose a new MAS-PL approach with two kinds of features: generic MAS features and application specific features. In the next section, we introduce our MAS-PL approach.

3 Overview of our MAS-PL approach

Our approach proposes to develop MAS families by following the general SPLE framework [1] and reusing MAS concepts (see Fig. 2).

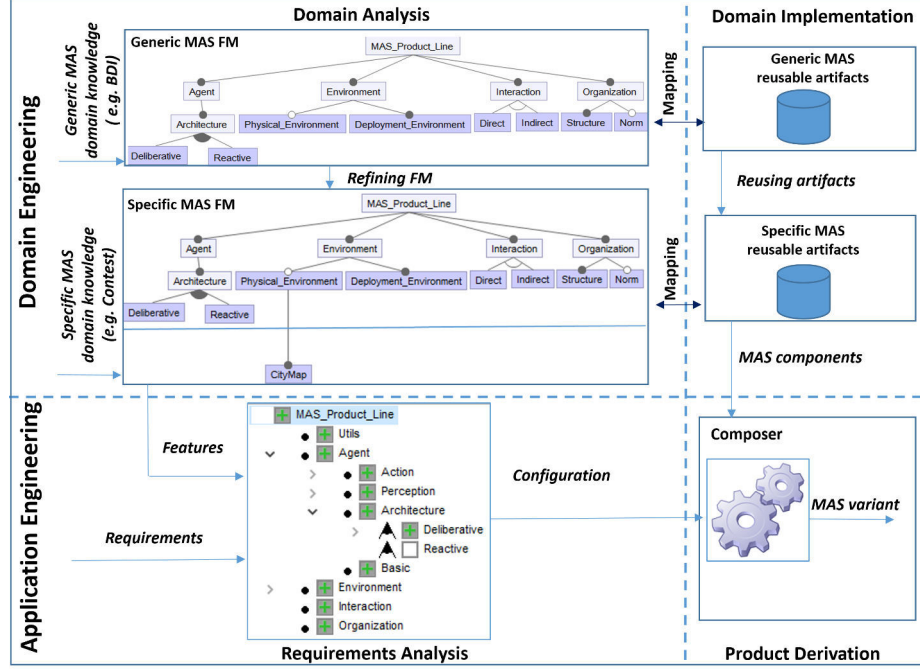


Fig. 2. Overview of our MAS-PL approach

Our approach splits the Domain Analysis (resp. Domain Implementation) activity of the domain engineering phase into two activities: 1) Generic MAS domain analysis (resp. Generic MAS domain implementation) and 2) Specific MAS domain analysis (resp. Specific MAS domain implementation). This distinction between *Generic MAS* domain and *Specific MAS* domain aims at capitalizing the expertise of using MAS methods, models and implementations, by reusing both *Generic MAS* features and *Domain specific* ones.

First, during the *Generic MAS* domain analysis activity, we analyze the Multi-Agent Systems knowledge in terms of generic MAS concepts which are involved in MAS approaches like GAIA [8], and PASSI [10]. Then, we represent *Generic MAS* concepts in terms of commonalities (similarities) and variabilities (differences) among MAS approaches, organized so that they refer to *Agent*, *Environment*, *Interaction* and *Organization* features. Our approach relies on AOSE methods, to enable reusing most common concepts and assets, and to

provide flexibility to MAS designers and developers by using the features they need. For instance, the *Role* concept that is provided by GAIA [8], PASSI [10] and AGR [15] can be placed as a child-feature component of the *Organization* feature. To illustrate the activity, we analyze the belief-desire-intention (BDI) model [17], some organizational models like Moise+ [20], and so on. This activity produces a *Generic MAS Feature Model (FM)*.

Second, *Specific MAS* domain analysis activity documents commonalities and variabilities between the members of a specific MAS family. We illustrate this activity with the Multi-Agent Contest example ¹. This activity produces a *Specific MAS FM*. In opposition to existing MAS-PL approaches that build it from scratch, our approach proposes to build it by reusing the *Generic MAS FM*.

Third, *Generic MAS* implementation activity involves implementing *Generic* reusable MAS artifacts, that do not rely on any specific MAS family. They implement *Generic MAS* features and are composed of Agent, Environment, Interaction and Organization artifacts.

Finally, *Specific MAS* implementation activity produces *Specific MAS* reusable artifacts. In existing MAS-PL approaches, these reusable artifacts are implemented from scratch. In our approach, *Specific MAS* implementation relies on *Generic MAS* implementation. It can adapt the corresponding artifacts to the specific MAS family.

Application engineering activities concern both MAS requirement analysis and derivation. According to the MAS requirements, MAS variants are specified by selecting a valid configuration from the *Specific MAS FM*. The specified product that represents the MAS application variant is then derived. As we are interested in *Composition-Based* implementation approaches [1] instead of *Annotation-Based* ones that use heavyweight notations, we need a *Composer* to derive the product. This derivation activity is done by FeatureHouse composer [2] that generates MAS variants automatically by composing reusable artifacts. We illustrate our approach by Contest Agent Variants based on configurations produced during the Application Engineering phase.

4 MAS Domain Engineering

This section describes the proposed domain engineering phase based on activities and their outputs.

4.1 Generic MAS domain analysis

In this first activity, we analyze MAS models' commonalities and variabilities to build the *Generic MAS FM* which is a compact representation of MAS concepts. We therefore define the features that are often shared by MAS applications.

Our idea is to produce a common FM (see Fig.3) based on that generic representation and to use the Vowels paradigm [13] to organize these features.

¹ <https://multiagentcontest.org/>

The Vowels paradigm considers that MAS are composed of (1) Agents (Vowel A), which refers to the description of internal architectures of the system processing entities; (2) Environment (Vowel E), which refers to domain-dependent elements for structuring external interaction among the system entities; (3) Interaction (Vowel I), which refers to elements for structuring internal interaction among the system entities; and (4) Organization (Vowel O), which refers to elements for structuring entities within the MAS. We do not include the User (Vowel U) dimension that is considered in the Vowels extension to explicitly take into account the user.

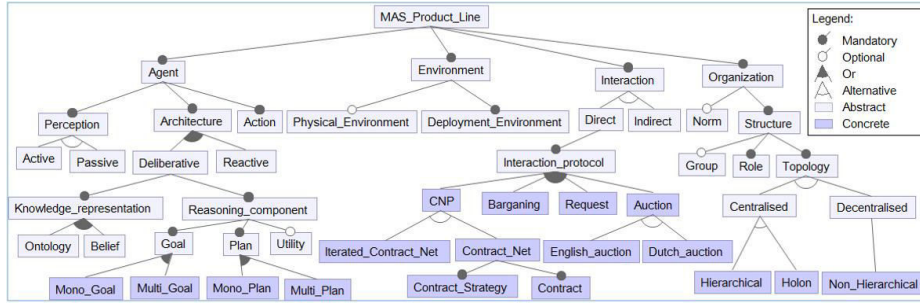


Fig. 3. Generic MAS Feature Model

Agent features: Agent architectures provide solutions to structure agents and define their functionalities in order to enable them to act and to interact in a dynamic environment. Most existing architectures follow the perception-action loop. We thus propose three categories of features: *Perception*, *Architecture* (Internal Architecture) and *Action*. For the Internal Architecture, we consider both categories: *Reactive* and *Deliberative* [14]. While deliberative agents follow Perception-Deliberation-Action cycle, reactive agents follow Perception-Stimulus-”Re” action one. *Hybrid* architectures can also be defined by combining the previous ones (selecting both *Deliberative* and *Reactive features*).

Deliberative agents need a knowledge model to provide them a representation about their environment, and their own knowledge. The *Knowledge_representation* feature represents knowledge representation such as *Belief* and *Ontology*. The BDI model [17] includes three mental attitudes: Beliefs, Desires and Intentions. Whenever the agent has a BDI architecture, all of *Belief*, *Goal*, and *Plan* features are mandatory. The *Utility* feature of our *Generic MAS FM* represents an option used to endow Goal-Based agents with an utility measure for evaluating the level of success when reaching the goal, to obtain Utility-Based agents [26].

Environment features: Agents are situated in an environment that used to be domain dependent and generally spatial. That environment represents many aspects that conceptually do not belong to agents themselves such as in a software infrastructure on which the MAS is deployed, or in a representation

of physical environment. City maps used for situated agents can be mentioned as an example. We represent the agent Environment product line by two main environment features: the *Deployment_Environment* which is mandatory and the optional *Physical_Environment* since the physical world is not always represented in MAS.

Interaction features: Interaction provides a way to ensure coordination of agents' activities. Agents' interaction can be *Direct* or *Indirect*. In direct interaction, agents exchange messages to coordinate their behaviour and achieve the global goal. In indirect interaction, agents use the environment to share information and coordinate their actions. For example, ants use the pheromone to coordinate. In this paper, we focus only on direct interaction that is regulated by interaction protocols. The *Interaction_protocol* feature is therefore mandatory. Interaction protocols were introduced into MAS to facilitate the specification and the implementation of interaction between agents. According to FIPA² definition, an interaction protocol is a common pattern of communication (a predefined sequence of messages). Thus the specification and the implementation of the Protocol could be independent of the scope and of the agent internal architecture. Several interaction protocols have been proposed: request protocol, bargaining, auction and Contract Net Protocol (CNP), among others. Our *Generic MAS FM* includes some of them with a possible feature selection. The *CNP* feature has two-child features which are alternative: *Contract_Net* and *Iterated_Contract_Net*, representing two possible versions of the protocol.

Organization features: In MOISE+ model [20], the organization is seen under three points of view: Structural, Functional, and Normative. We propose to model the MAS Organization product line by the root *Organization* feature that includes two-child features: the *norm* optional feature ; and the mandatory *structure* feature to represent all possible organization structures. Ferber et al. proposed the first organizational model Agent-Group-Role (AGR) [15], to highlight the importance of organizational concepts like 'groups'.

The *Role* concept is used in most existing MAS organizational meta-models while the *Group* concept is used only in some of them such as AGR and AGRE [16]. So, the *Group* feature is optional while the *Role* feature is mandatory. The last mandatory feature that concerns the *structure* is the *Topology*, that has two alternative child features: *Centralized* and *Decentralized* organizations described with other child features.

4.2 Specific MAS domain analysis

During this activity, we analyze MAS commonalities and variabilities of a specific MAS family (e.g. Multi-Agent Contest) to produce a *Specific MAS FM* to refine the *Generic MAS FM*. The refinement process is achieved by adding Specific features to solve the problem. These specific features are placed hierarchically under the generic features they are specializing. Since our specific domain is

² <http://fipa.org/>

chosen, from now, we will refer to *Specific MAS* features as *Contest MAS* features. Contest agents' teams move around the streets of a realistic city, having the goal of earning money by completing jobs. Teams should then decide how to navigate the city map, and where to get the resources to assembly, buy and deliver items considering targets like shops, warehouses, charging stations, and storage facilities. Tournament points are distributed according to the amount of money a team owns at the end of the simulation.

Fig.4 depicts an example of the *Contest MAS FM*, where a line separates the two categories of features. For example, the Contest mandatory feature *cityMap* *child of Physical_Environment* refines the *Environment* feature and corresponds to the provided Contest environment. The specific *BuyItem_Goal* feature refines the *Mono_Goal* one, and is used to specify specific Contest agents for the items acquisition goal. This first version of the *Specific MAS FM* represents the simplest one and considers only agents that achieve one goal. But, as our approach is incremental, this simplest version of the *Specific MAS FM* can be also refined at its turn. The left side of Fig. 5 depicts another version of the *Specific MAS FM* which refines the simplest one. This refinement includes adding more features to support other versions of agents. For example, both of *DeliverItem_Goal* and *Charging_Goal* Contest specific optional Features refine the generic *Multi_Goal* feature by specifying agents that have to achieve items delivering and charging goals. In this example, we consider only the Agent variability. The environment dimension is provided by the Contest. When considering other generic variabilities such as Interaction, the process remains the same.

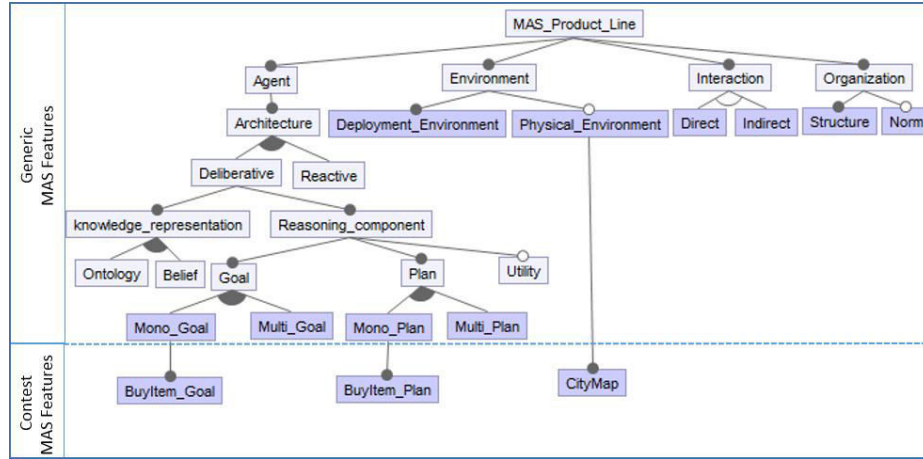


Fig. 4. An example of the Contest Specific MAS FM that refines the Generic MAS FM

4.3 Generic MAS Implementation

The following activity concerns the *Generic MAS reusable artifacts* implementation. The MAS implementation is often based on existing frameworks. Artifacts correspond to components provided by those frameworks. In this paper, we consider APLTK (A Toolkit for Agent-Oriented Programming) [5] to illustrate the feasibility of the approach in the implementation side. The set of reusable artifacts can be then enriched by considering other frameworks such as JaCaMo [7].

Agent artifacts: Agent reusable artifacts implement *Agent* features. Although we could not raise variability concerns related to *Belief*, both *Goal* and *Plan* can present variability depending on the BDI choosing algorithms. The three algorithms we use to illustrate our approach are those proposed by Wooldridge [17]. The variability among these algorithms lies on the cardinality of the set of B, D and I. The first algorithm represents the simplest version. It corresponds to mono-Goal agents. While the second algorithm concerns agents that have to achieve multiple goals. The last algorithm proposes to enrich the library plan during the execution. *Goal* and *Plan* variabilities can then be expressed by the following features: *Mono_Goal*, *Multi_Goal*, *Mono_Plan* and *Multi_Plan*.

Most BDI model implementations use *brf* (belief revision function), *ogf* (option generation function), *filter* and *asf* (action selection function). However, the algorithm variability has an impact on these functions. For instance, if we consider BDI *Mono_Goal* implementation, agents use neither the *ogf* function nor the *filter* one. *Mono_Plan* agents do not need a function to select a plan.

We implement the *Mono_Goal* feature with reusable artifacts, composed of the functions proposed in the simplest algorithm of Wooldridge [17] : (i) *getting-Percepts()*: to execute the get-next percept, (ii) *createBeliefsFromPercept()*: to create and update the agent beliefs, it represents the *brf()* function; (iii) *checkAll-BeliefsForInsertGoal()*: for the agent deliberation to correspond to the *deliberate()* function, and (iv) *performActionGoal()*: to select a plan and execute it.

Environment artifacts: The general view of the environment considers that agents are part of the environment, which can provide for example means or resources for agent communication. However, environment standardization should be done by separating both concepts. Behrens et al. [6] proposed a generic approach for connecting agents to environment by considering reusable environment artifacts, that are as much independent of a specific environment structure as possible. For example, we reuse EIS (Environment Interface Standard) API [6], that represents possible reusable environment artifacts, which are mapped to the *Physical_Environment* feature. EIS reduces the implementation effort for connecting to the agents' environments (e.g. Unreal Tournament UT3 and UT2004 gaming environments, and the Multi-Agent Contest).

Interaction artifacts: The reusable Interaction artifacts concern mainly the Interaction protocols. FIPA standard Interaction artifacts are available to the programmer through abstractions to develop FIPA-compliant MAS. Some reusable FIPA implementations are provided by MAS frameworks such as the CNP implementation in JADE . For example, we reuse the CNP-Initiator and CNP-Participant roles that are mapped to the *CNP* feature.

Organization artifacts: Some concepts of the *Generic MAS FM* which are linked to the organization such as *Roles* may be implemented by reusable artifacts at the implementation level. For instance, in JADE API, the classes implementing the behaviours can represent roles that are reusable.

4.4 Specific MAS Implementation

During the last activity of Domain Engineering, *Specific MAS reusable artifacts* implement *Specific MAS features* by reusing *Generic MAS artifacts*.

Fig. 5 summarizes the four activities of the Domain Engineering. It illustrates a *Contest MAS* reusable artifact which is at the right side bottom of the figure. This Contest artifact implements the *BuyItem_Goal* Contest feature, by reusing the *PerformActionGoal()* function that is mapped to the *Mono_Goal* feature.

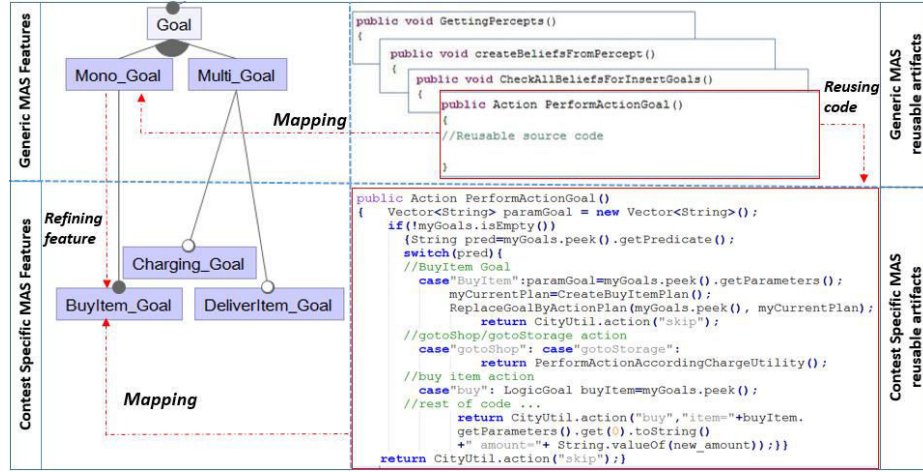


Fig. 5. An example of a Contest MAS reusable artifact obtained through the four Domain Engineering activities

5 MAS Application Engineering

This section will present application engineering activities, and illustrate them with Contest Agent configuration and derivation.

5.1 Requirement Analysis

During this activity, features are selected from the *Specific MAS FM* to create a configuration according to the application requirements. Before achieving this

feature selection, the target MAS variant is chosen. After that the MAS developer selects the features that fill those requirements.

Suppose we want to obtain a customized MAS application which contains a Car Contest Agent (CCA) that has to achieve one goal, which is to execute an acquisition job (buying items in a shop). The agent has no charge utility concerning its battery, and has to execute one Plan (find a shop, move to it, and buy the items). Such requirements (needs) are fulfilled by the given configuration presented on (a) in Fig. 6. All selected features are represented on (b) through a set of literals.

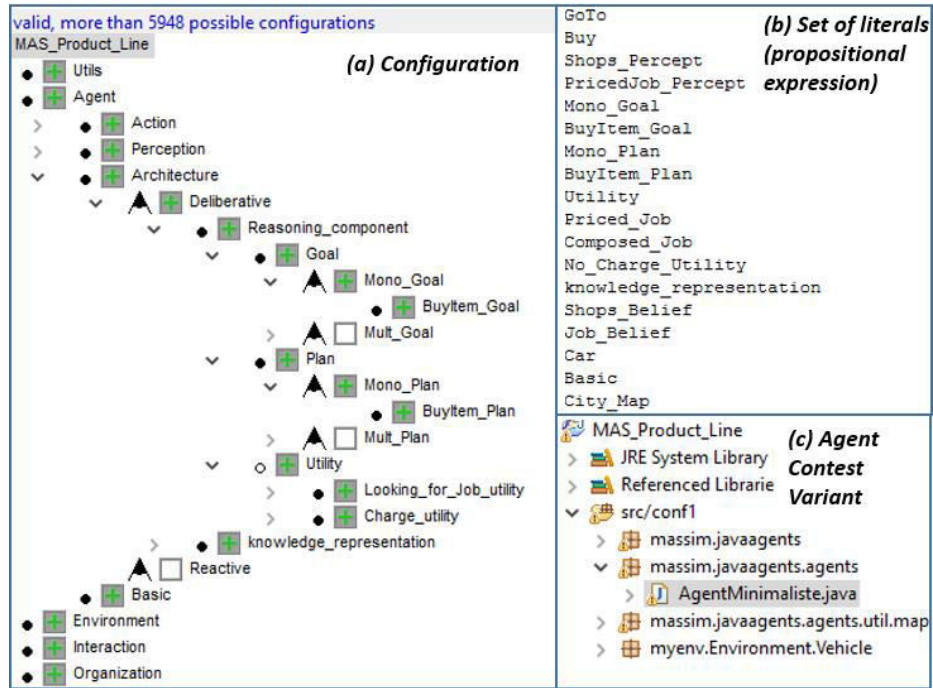


Fig. 6. An example of (a) a Contest Agent Configuration (b) its propositional expression and (c) the derived Contest Agent Variant

5.2 Product Derivation

For a valid configuration, the MAS variant derivation activity is automatically achieved by a composer, such as FeatureHouse. Fig. 6 gives on (c) the derived code relative to the configuration on (a). For more details, interested readers can access our link ³.

³ <http://www-desir.lip6.fr/~boufedji/emas17.html>

6 Evaluation

In order to evaluate the proposed approach, we derived several Contest Agent Variants and deployed them in the MAS Contest environment. The main objective of this evaluation is to highlight, through some pertinent metrics, the advantages our approach brings. The implementation of our approach adopted the Feature IDE tool ⁴ to specify the feature models, and to create agent configurations. FeatureHouse composer was used to derive Contest Agent Variants. Since our product line does not involve interaction nor organization aspects of the team, we consider only agents that have to complete their priced jobs. More details are available on the Contest website under the Contest 2016 example ⁵.

The case study offers more than 5948 possible configurations, as shown on the upper left side in Fig.6. We present among them ten (10) of the derived and simulated Contest Agent Variants. The variants are labeled from CAV1 (Contest Agent Variant) to CAV10. Table 1 represents the relative configurations of each variant that includes both Generic and Contest MAS features.

CAV1 and CAV2 have the minimal feature configurations that involve a total of six mandatory features. CAV1 corresponds to the simplest agent (see Fig.6). CAV3 to CAV10 represent variants with a maximum number of eight possible selected features. These variants follow the second BDI algorithm [17]; and differ on their charging utilities. According to those variabilities, more or less methods and lines of code are derived. We calculate them for each variant, by using eclipse Metrics⁶. The metrics show the advantage of using software product lines in our approach to develop MAS. There is a development time saving of about 354 lines of code, and 25 methods from CAV1 to CAV10. We can also compare variants according to their strategies using utilities. For example, through simulations, we could distinguish between the best and the worst charging utilities. We could also make the agents variants play together.

The main advantage of our MAS-PL approach is the fact that compared to existing MAS-PL approaches, MAS designers and developers will not build feature models nor develop the source code from scratch. Another positive advantage of our approach lies in the use of known notations covering both design and implementation aspects, what would increase its chance to be adopted.

Moreover, when using our approach, we can compare Contest agents performances easily since the agents are derived automatically and can be deployed faster.

However, one limit of our approach concerns the hypothesis made to simplify the study. We did not consider interaction and organization aspects. Another limit is that most variability is specific to Contest. But, we prospect to enrich our work with a more refined MAS variability. Currently, we are studying organizational aspects, and different selecting-job utilities linked to interaction

⁴ <https://marketplace.eclipse.org/content/featureide>

⁵ <http://multiagentcontest.org>

⁶ <http://eclipse-metrics.sourceforge.net>

In this paper, we illustrated the different activities of our approach by simple examples issued from the Multi-Agent Contest 2016. We showed how configurations can fill MAS requirements to derive customized agents.

During our research work, in order to evaluate our MAS-PL approach, we derived a Contest product line of agents that include variants that had been simulated in the Multi-Agent Contest environment. We compared those variants according to some metrics. The result of our work shows that our approach is promising.

As for further on-going research work, we are concentrating on an interesting perspective which would allow us to obtain an Interaction and Organizational variability to enrich both *Generic MAS* FM and artifacts. We intend, for instance, to introduce interaction mechanisms such as ant-based algorithms.

Another perspective is to suggest to researchers and MAS developers to use our approach when implementing Multi-Agent System Product Lines in different domains. The feedback would help us improve our work to serve the technological development and advances.

References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines. Springer (2013)
2. Apel, S., Kastner, C., Lengauer, C.: Language-independent and automated software composition: The FeatureHouse experience. *Software Engineering, IEEE Transactions on* 39(1), 63–79 (2013)
3. Ayala, I., Amor, M., Fuentes, L., Troya, J.M.: A software product line process to develop agents for the iot. *Sensors* 15(7), 15640–15660 (2015)
4. Ayala, I., Horcas, J.M., Amor, M., Fuentes, L.: Using models at runtime to adapt self-managed agents for the iot. In: *German Conference on Multiagent System Technologies*. pp. 155–173. Springer (2016)
5. Behrens, T.: Towards Building Blocks for Agent-Oriented Programming. Ph.D. thesis, Clausthal University of Technology (2012)
6. Behrens, T.M., Hindriks, K.V., Dix, J.: Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence* 61(4), 261–295 (2011)
7. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. *Science of Computer Programming* 78(6), 747–761 (2013)
8. Cernuzzi, L., Juan, T., Sterling, L., Zambonelli, F.: The gaia methodology. In: *Methodologies and Software Engineering for Agent Systems*, pp. 69–88. Springer (2004)
9. Cirilo, E., Nunes, I., Kulesza, U., Lucena, C.: Automating the product derivation process of multi-agent systems product lines. *Journal of Systems and Software* 85(2), 258–276 (2012)
10. Cossentino, M.: From requirements to code with the passi methodology. *Agent-oriented methodologies* 3690, 79–106 (2005)
11. Cossentino, M., Burrafato, P., Lombardo, S., Sabatucci, L.: Introducing pattern reuse in the design of multi-agent systems. In: *Net. ObjectDays: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World*. pp. 107–120. Springer (2002)

12. Dehlinger, J., Lutz, R.R.: Gaia-PL: A product line engineering approach for efficiently designing multiagent systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20(4), 17 (2011)
13. Demazeau, Y.: From interactions to collective behaviour in agent-based systems. In: *Proceedings of the 1st. European Conference on Cognitive Science*. Saint-Malo. Citeseer (1995)
14. Ferber, J.: *Multi-agent systems: an introduction to distributed artificial intelligence*, vol. 1. Addison-Wesley Reading (1999)
15. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: *Agent-Oriented Software Engineering IV*, pp. 214–230. Springer (2004)
16. Ferber, J., Michel, F., Báez, J.: AGRE: Integrating environments with organizations. In: *Environments for multi-agent systems*, pp. 48–56. Springer (2005)
17. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: *International Workshop on Agent Theories, Architectures, and Languages*. pp. 1–10. Springer (1998)
18. Guessoum, Z., Cossentino, M., Pavón, J.: Roadmap of agent-oriented software engineering. In: *Methodologies and software engineering for agent systems*, pp. 431–450. Springer (2004)
19. Hara, H., Fujita, S., Sugawara, K.: Reusable software components based on an agent model. In: *Parallel and Distributed Systems: Workshops, Seventh International Conference on*, 2000. pp. 447–452. IEEE (2000)
20. Hübner, J.F., Sichman, J.S., Boissier, O.: Moise+: towards a structural, functional, and deontic model for mas organization. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. pp. 501–502. ACM (2002)
21. Nunes, I., Cowan, D., Cirilo, E., De Lucena, C.J.: A case for new directions in agent-oriented software engineering. In: *International Workshop on Agent-Oriented Software Engineering*. pp. 37–61. Springer (2010)
22. Nunes, I., De Lucena, C.J., Cowan, D., Kulesza, U., Alencar, P., Nunes, C.: Developing multi-agent system product lines: from requirements to code. *International Journal of Agent-Oriented Software Engineering* 4(4), 353–389 (2011)
23. Nunes, I., Kulesza, U., Nunes, C., Cirilo, E., Lucena, C.: Extending PASSI to model multi-agent systems product lines. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. pp. 729–730. ACM (2009)
24. Peña, J., Hinchey, M.G., Ruiz-Cortés, A.: Multi-agent system product lines: challenges and benefits. *Communications of the ACM* 49(12), 82–84 (2006)
25. Peña, J., Hinchey, M.G., Ruiz-Cortés, A., Trinidad, P.: Building the core architecture of a nasa multiagent system product line. In: *Agent-Oriented Software Engineering VII*, pp. 208–224. Springer (2007)
26. Russell, S., Norvig, P., Intelligence, A.: *A modern approach. Artificial Intelligence*. Prentice-Hall, Egnlewood Cliffs 25, 27 (1995)

Towards the Engineering of Agent-Based Augmented Worlds

Angelo Croatti and Alessandro Ricci

DISI, University of Bologna, Via Sacchi 3 – Cesena, Italy
emails: {a.croatti|a.ricci}@unibo.it

Abstract. Recent technological advances in wearable computing and augmented reality technologies bring new opportunities into agent research, in particular to explore agents’ role in understanding, designing and developing *augmented worlds*. With the term “augmented worlds” we mean a broad spectrum of systems that go from human/personal augmentation – related to wearable and eyewear computing – to environment and reality augmentation – related to ubiquitous computing and augmented reality. In this position paper, we frame the role that agents can have for augmented worlds, as an interesting and challenging application domain fostering the research and development of new personal assistant technologies, and we sketch a research agenda for engineering these agent-based systems.

1 Introduction

In next decade, *augmentation* technologies and systems are going to play a mayor role in the ICT panorama, allowing us to design full-fledged *augmented worlds*.

After more than 30 years of research in wearable computing [24,43,44], wearable technologies – and in particular eyewear ones like smart-glasses [5] – are almost ready for the prime time, to be adopted by people engaged in daily activities—either professional or not. The key point of these technologies compared to e.g. mobile ones such as tablet or smartphone is the possibility to realise *hands-free* systems or *on-the-go* systems, that make it possible to perceive and interact with the system without necessarily stopping or distracting the user from what she is physically doing. Conceptually, these devices allow to extend people cognitive capabilities, improving the way in which they perform their tasks [44]. This can be interpreted as a form of *human augmentation*, in a way that recalls the Augmented System conceptual framework introduced by the computer science pioneer Doug Engelbart more than 50 years ago [15].

Wearable computing is a key enabling technology for Augmented Reality (AR) and Mixed Reality (MR). AR can be defined as a medium in which digital information is added (superimposed) to the physical world *in registration* with the world itself and displayed to a user dependently on its location and its perspective [1,25,10]. Such a registration may occur in different ways, e.g. by means of fiducial markers placed in the environment, perceived and processed through the camera(s) mounted on smart-glasses or devices (or even on smartphones), or directly exploiting the spatial information obtained by sensors mounted on the AR visor. Mixed reality (MR) refers more generally to the merging of real and virtual worlds to produce new environments and visualisations

where physical and digital objects co-exist and interact in real time [9]. As defined by P. Milgram and F. Kishino, it is “anywhere between the extrema of the virtuality continuum” [25], that extends from the completely real through to the completely virtual environment with augmented reality (AR) and augmented virtuality ranging between.

Nowadays Mobile AR (MAR) technologies [19] allow for easily creating outdoor applications where, e.g., physical places and spaces are decorated by dynamic virtual information that can be perceived by users simply using a smartphone or smart-glasses, depending on their position (obtained through GPS). The recent planetary success of a game like Pokemon Go¹ – featuring just basic features of MAR – is an example of the level of engagement and attraction that this kind of application can generate. State-of-the-art AR technologies – main examples are Microsoft Hololens², Magic Leap³, Meta⁴, Google Tango⁵ – allows for augmenting indoor physical environments with fairly sophisticated holograms, that users can interact with.

These technologies can have a huge impact from an application point view, allowing for rethinking the way in which people work, interact and collaborate—escaping from the limits that current mobile devices such as smartphones and tablets enforce. Given the progress of HW technologies, in the short future the level of sophistication and functionality provided by these augmentation technologies will largely depend on the smart software that will be capable to properly exploit them, so that we can talk about *software-based* augmentation [51].

If wearable computing and AR are a direct example of augmentation technologies, ubiquitous/pervasive computing [49,40,21,33] provides a further and related form of augmentation of the physical world. In this case, the augmentation is given by open ecosystems of connected and invisible computing devices, embedded in physical objects and spread in the physical environment, equipped with different sensors and actuators. Data generated by these devices is typically collected and managed in the cloud, and accessed by mobile/cloud applications. This embedded computing layer *augments* the functionalities of the physical things that people use everyday and, again, are going to have a strong impact on how people work, interact and collaborate [32]. This view about environment augmentation strongly recalls the idea of *computer-augmented environments* by P. Wellner et al. [50], in which the digital (cyber) world is merged with the physical one.

From a conceptual level, these augmentation technologies, spanning from human augmentation to reality and environment augmentation, can be considered as key enablers for designing and developing a broad class of applications that we refer here as *augmented worlds* (AWs), as systems in which the digital and the physical layers are deeply interconnected and merged. In this position paper we aim at providing a broad picture about the role that agents can play for augmented worlds, identifying research opportunities. This is related to a wide spectrum of existing research works and fields in literature, where agents has been already effectively adopted for modelling

¹ <http://http://www.pokemongo.com/>

² <https://www.microsoft.com/microsoft-hololens/en-us>

³ <https://www.magicleap.com>

⁴ <https://www.metavision.com/>

⁵ <http://get.google.com/tango/>

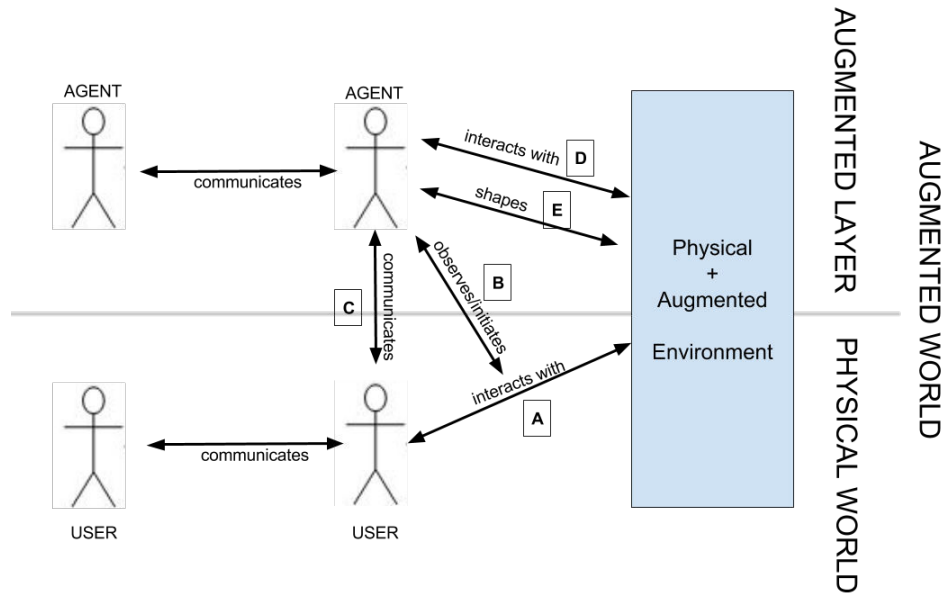


Fig. 1: Main Elements of Agent-Based Augmented Worlds

and engineering, e.g., AR systems [6], Ambient Intelligence (AmI) systems [39]. The augmented world view adopted in this paper allows for defining a unifying perspective which is useful to both relate issues that have been developed in separate disciplines, and to identify further interesting research opportunities for agents research, spanning from envisioning a new generation of personal assistant agents to new kind of agent-based AmI environments.

The remainder of the paper is organised as follows. In Section 2 we provide a high-level comprehensive discussion framing the role of agents in AWs. Then, in Section 3 we focus more on the engineering of agent-based AW, proposing a conceptual model which could be useful to drive the design and implementation of agent-based AWs. Finally, in Section 4 we conclude the paper by sketching a research agenda for future works in agent-based AW engineering.

2 Agents for Augmented Worlds (and Back)

The reference context of an AW is a physical environment – either indoor or outdoor – inhabited by some (human) users engaged in some physical activity, that could require mobility, physical actions (using hands), etc. Activities could be individual or cooperative. The augmentation layer – running on humans and on the environment – aims at enriching/extending with a set of functionalities supporting/helping human actions.

The first main point is: what agents could be useful for and what can they add to the state-of-the-art in eyewear computing and augmented/mixed reality. The main

contribution is providing a principled way to think and develop *pro-active* applications, beyond current *reactive* approaches. That is, exploiting agent models, architectures and theory for devising the next generation of smart augmentation software, framing a new level of *assistance*.

The diagram shown in Fig. 1 aims at capturing the broad range of scenarios, providing an overview about the role of agents in AW. It is an extension of the one about *interface agents* in [23]. An augmented world is made by an augmented layer on top of a physical world. Agents living in the augmented layer can be deployed at two different levels:

- agents augmenting the users – running on user personal devices (e.g., smart-glasses and smartphones)
- agents augmenting the physical environment – running on the smart environment infrastructures.

The former level will be discussed in Subsection 2.1, while the latter in Subsection 2.2.

2.1 Towards a New Generation of Personal Assistant Agents

Agents augmenting the users allow to conceive a new generation of *software personal assistant agents* (SPAs) [26,23]. Personal assistants are a well known application of software agents [29]. Existing proposal and technologies have been developed for different purposes and capabilities, such as scheduling joint activities and calendaring [27,48,42,2], task and time management [28,8,47], managing personal information [17], assisting in negotiation decision support [22].

Generally speaking, with SPAs the user is engaged in a cooperative process in which human and computer agents both initiate communication, monitor events, and perform tasks [23]. This is based on the metaphor that of a *personal assistant* who is *collaborating with the user* in the same work environment [23].

The AW introduces a richer context that allows for further interesting functionalities to explore.

“I see what you see” The main interesting feature of a SPA agents in AW – not fully explored in traditional PA – is the capability to “see what the user sees” – by means of the camera on the eyewear device – and, more generally, to know what the user is perceiving about her context. This is represented by the relationship labelled with B on the diagram in Fig. 1 – that is, an agent is capable to observe and initiate user interaction with the environment (which is meant here to include also perceptions). This allows to frame new kind of a pro-active assistance in which SPAs reason not only about the context of the user, but about what the user is perceiving – and not perceiving – from that context, what she is looking at, etc.

The definition and development of SPA agents in AW introduces interesting new research challenges for the agent community, partially tackled in related research context such as eyewear computing [5], cognition-aware computing [4], activity-based computing [46]. These agents are meant to build dynamically a model about what the user is perceiving, and use this knowledge along with the information about user’s goals, the state of ongoing activity, the actual state of the physical environment, to provide

a pro-active and smart assistance, possibly anticipating and notifying problems and suggesting actions to do. This is strongly related to existing research on plan/activity recognition [45], that can be further characterised and enriched here to consider augmented worlds and augmentation technologies.

This scenario can be further enhanced by considering SPAs assisting teams of users immersed in the same augmented world, possibly sharing the same physical context or being in different places. In that case, different SPAs can communicate and share information (through the network) in order to understand *what the team sees* overall, and use this knowledge to help both the action of the individual user and the coordination of the team.

Agents as an Augmentation of the Self Traditional SPAs are typically deployed on desktop computers or mobile devices such as smartphone. The shift from mobile to wearable/eyewear devices has a strong impact on HCI interface and then on how a SPA and the human user can communicate/interact. Handsfree systems and *on-the-go* interfaces should be designed to avoid distracting the user from what she is doing – if not desired – providing an information and suggestion flow which is seamlessly perceived and exploited by the user in her activity. Metaphorically speaking, an agent in this case is not really perceived as an entity to whom delegate something, but more as a guardian angel monitoring and suggesting. Or, it disappears, being perceived as *an extension of the self*, whose perceptions, beliefs, and possibly goals could be thought to be an extension of the user's ones. This perspective goes toward the design of interface agents that – in spite of agent autonomy – should make the user always “feel in control” [14]. This perspective calls for new interesting issues to be explored in the research on interface agents and virtual agents, starting from existing related research work developed in the context of wearable computing about specific applications—such as the wearable remembrance agent [34].

Agents as Mediators for Augmented Realities So far we have considered users and their SPAs immersed in purely physical environments. Further interesting issues arise as soon as we introduce augmented reality and, more generally, augmented environments (see Fig. 1). In this case, the kind of assistance of SPA can include also the possibility to interact with/manipulate the augmented entities enriching the physical space, possibly shared with other users (and SPAs) situated in the same space. For instance, a SPA could automatically annotate with virtual notes the physical environment where the user is working, on physical objects, as a memo for the user herself or useful for implicitly coordinating with other users, perceiving those virtual signs. Differently from a MAR browser, an SPA could show only those augmented entities that the agent knows to be relevant according to user's goals and needs for current activity, so functioning as a smart filter when moving in complex AW. Augmented entities composing the augmented environment could provide an interface – like in the case of Web of Things [18] – that SPAs can exploit to act on them, changing their state – prefiguring the vision of a *Web of Augmented Things* that can be perceivable by humans and interactable by agents [12].

These scenarios call for investigating the role of mediators and indirect interfaces for SPAs in AW, and the type of *delegation* that can be assigned to them.

2.2 Toward a New Kind of Agent-Based Smart Environments

Besides being personal user assistants, a main role that agents can play in AW is to be the building block for shaping next generation of pro-active/smart augmented/mixed reality applications, where augmented entities are not limited to messages or information superimposed on the physical reality, but full-fledged stateful, dynamic, and possibly autonomous entities. This is the case, for instance, of AR and MAR games (e.g., [31]). Recently, we have witnessed the level of engagement that AR games can generate by the case of Pokemon Go, a location-based game featuring some elements of (M)AR applications.

Besides games, this direction is epitomised by *hologram-based* applications which is possible to build today using enabling technologies such as Microsoft HoloLens, Quantum Leap, or Meta2, that allow for creating full-fledge holograms, possibly featuring autonomous and articulated behaviours, interacting with users.

Generally speaking, agents can be naturally used to model and develop those augmented entities which are part of the augmented layer featuring a pro-active, goal-oriented behaviour, perceiving and acting in the augmented environment (label E in Fig. 1), encapsulating the pro-active functionalities of such novel kind of smart environments. Agents can have themselves an augmented representation that makes them perceivable by humans (by means of, e.g., smart-glasses), similar to virtual agents, but situated in the physical environment. Or, the agents can be just “soul” situated in the that augmented environment, perceiving the users inside and acting accordingly. In that case they are similar to agents in ubiquitous/pervasive computing and context-aware contexts, but capable of perceiving/acting on the virtual environment.

Agents shaping Augmented Realities In literature, a comprehensive account about the role of agents in Mixed and Augmented Reality contexts has been already given with the concept of MiRa (Mixed Reality Agents) [20] and AuRA (Augmented Reality Agents) [6].

A MiRA is the broadest concept, used to delineate “an agent embodied in a mixed reality environment”, blending the notion of agency with embodiment [13] and Milgram’s reality–virtuality continuum [25]. AuRAs identify those MiRAs that can both sense and act in the virtual component of their reality but can only sense in the physical [6]. This specialisation helps distinguish AuRAs from mixed reality agents with a primarily physical component, such as robots operating in an AR environment.

Given this taxonomy, [6] identifies two main purposes in developing AR applications: AuRAs as *interface paradigm* – modelling interface agents living in AR environments and directly interacting with human users – and as *design paradigm*, i.e., an effective software development solution tackling main complexities of the real-world environments characterising AR applications.

Related to applications, recent works have emphasized the value of (serious) mixed-reality games as a platform to explore scenarios in the real world that are typically hard to study in realistic settings, such as disaster response, to study the joint activities of human-agent collectives [16]. Similarly, mixed-reality testbeds have been deployed for the incremental development of human-agent robot applications [7].

Towards Augmented Societies The vision of augmented worlds discussed in this paper can be useful to trigger further explorations beyond existing work. In particular, the wider notion of augmentation brought by AW allows for conceiving a new generation of agent-based AR and MR systems based on a *bidirectional augmentation* of the physical and digital matter, the physical and the virtual reality. “Bidirectional” in the sense that for software agents shaping and living in AWs, the physical environments can be conceived as an augmentation of their (virtual) environment, including the augmented entities that are perceived by human users (see Fig. 1). As humans perceive and act upon virtual augmented entities in their physical reality, agents living in the augmented layer (and controlling the augmented entities) can perceive and act upon physical entities, by means of a properly designed agent environment coupled to the physical worlds by means of pervasive computing technologies. This view is being explored in Mirror Worlds [36,38].

Following this view, AWs can be framed as virtual cities/environments where an open society of intelligent agents live, layered on top of physical cities/environments where people live, providing different levels of high-level augmentations – from cognitive augmentation to social augmentation. This view triggers a wide range of research issues and challenges that have not been explored yet about this kind of augmented, *blended* societies—in terms of models (including organizational models), infrastructures, as well as the problems and risks that such a vision about augmented societies can bring, and consequently devising the proper ways to avoid or deal with them.

3 Engineering Augmented Worlds: a First Conceptual Model

After discussing the role of agents for AWs, in this section we make a step towards their engineering by discussing a first concrete conceptual model – whose main elements are shown in Fig. 2. This can be considered a generalisation of the model introduced in [35], about Mirror Worlds.

In this model, the degree of augmentation introduced by the AW concept to the physical reality is given by full-fledged computational objects located in the space that users and agents can perceive and interact with. We refer to these computational objects as *Augmented Entities* (AEs).

Generally speaking, an AE can be totally virtual, as holograms that are perceived by humans in the physical world in some specific location, and that are observed and controlled by agents in the digital space. Or, they can be coupled to some concrete physical object, representing a kind of virtual extension *or mirror* of the object in the physical world. This includes also *human users*, enabling agents to observe users’ actions and movements.

As depicted in Fig. 2, agents can observe the AW at two different levels. An agent can observe/track specific AEs, perceiving their observable state and related changes. Or, agents can track *regions*, as zones of the physical/geographical space—where one or multiple augmented entities can be located.

Finally, to allow augmented entities to be perceived in the physical world by humans equipped with suitable mobile/wearable devices – the concept of *hologram* is introduced, defining the geometry and representation of the entity in the real world, like in classical AR applications. This geometry can be described, e.g., as a set of 2D/3D points defined

Discovery and Observability The spatiality of augmented entities can be exploited to enrich the ways in which these computational objects are discovered, observed and more generally how they interact with each other. Inside an augmented world – whose execution is intended to be supported by an ad-hoc *AW runtime* – the reference to an AE could be retrieved by *lookup* functions provided by runtime, specifying the region of the world to be queried or the entity identifier. Vice versa, an AE could also be retrieved as soon as it becomes *observable*, given its position with respect to the position of the *observing* entity (either an agent or an human user). In order to support this modality, the augmented entity abstraction has a observation-related property, referred in the simplest way as *observability radius* and *observation radius*.

Physical Embedding The notion of augmentation in augmented worlds eventually means also the extension of the functionalities of existing physical objects. Such extension can be useful, first of all, for human users interacting with the physical objects, extending the object affordances by means of virtual interfaces to control or inspect the state of the physical object. Secondly, it allows the possibility to enrich physical objects functionalities to exploit the computing capabilities given by the augmented layer. This kind of augmentation requires in general some kind of *coupling* between the physical objects and the corresponding augmented entities providing their extension, such that changes to the physical state of the objects are tracked and notified in the augmented level and vice versa.

User Modelling and Interaction User interaction is a main aspect of augmented worlds, being them applications where human users are engaged in some kind activity in some physical environment. An augmented world is typically a *multi-user* application, where multiple users perceive, share and interact within the same augmented entities.

Many specific applications may require to explicitly model users themselves inside an augmented world, in order to make them observable and trackable by agents. Given the AW conceptual model, this can be accomplished by specific augmented entities representing users' *augmented body*, featuring an observable state coupled to users' physical state (reporting e.g. where the user is looking at). From the one hand, by exploiting the observation-related features, it is possible to determine which entities of the world the user is perceiving and their observable state. From the other hand, an agent can *see what the user sees* because the agent is able to observe (and react to) the state of the augmented entity that represents the human in the AW encapsulating the current "state" of the human, from its position to its activities and perceptions.

3.1 Agents Frameworks for Augmented Worlds

The proposed conceptual model can be useful to drive the definition of concrete computational and programming models based on the agent-oriented programming abstractions of existing agent platforms.

As an example following the work in [35], we are exploring the use of JaCaMo platform [3] for that purpose. We consider this platform particularly effective for this job since it allows to explicitly model and implement agent environments, as defined

in the A&A (Agents and Artifacts) [30] meta-model, as a first-order abstraction aside to agents to engineer multi-agent systems. Accordingly, augmented entities belonging to an augmented world can be designed and implemented as *artifacts* composing the agent environment. In A&A an artifact can be used to model and design any kind of resources and tools used and possibly shared by agents to do their job [37]. Artifacts have an *observable state* that agents can perceive and a set of *operations*, that agents can request as *actions* to affect the world. So, in the context of AWs, agents can perceive, observe and manipulate augmented entities by dynamically creating, observing and using artifacts.

4 Setting Up a Research Agenda for Engineering Agent-Based AWs

In this position paper we gave our viewpoint about the role that agents can play for engineering AWs. We identified two broad related directions. The first one is about a new generation of personal assistant agents, fully exploiting wearable technologies to provide smart pro-active support to human users – either individual or teams – engaged in activities in some physical and work environment. The second one is about the definition of models and architectures for agents shaping AWs and living and reasoning in AWs, as well as platforms and infrastructures to build and play with agent-based AW—embedding enabling AR technologies.

We proposed then a conceptual model, expressive enough – from our point of view – to capture the main elements that characterise agent-based AWs. This is a generalisation of the specific model that we are exploring with Mirror Worlds, implemented on top of JaCaMo [35,38].

Actually these can be considered just the first steps of a more comprehensive research agenda that can be devised about the engineering of agent-based AWs. In that research agenda we see as main points:

- Identifying a common conceptual framework for agent-based AW, integrating the points discussed in this paper with existing agent AR-frameworks such as AuRA [6] and MiRA [20].
- Defining a rigorous specification of the conceptual model, as a common meta-model for developing and running agent-based AWs possibly based on heterogeneous agent technologies and platforms.
- Deepening the investigation about what does it mean designing AWs from a methodological point of view, integrating research works developed in different fields: from Human Computer Interaction and Augmented Reality, down to Agent-Oriented Software Engineering.
- Identifying in more detail the main technical challenges that are put forth by the engineering agent-based AWs, including existing results and contributions in other fields that are important for tackling them. A main recent one is *Eyewear computing* [5], which appears essential for the development of personal agents living in AWs.
- Identifying some case studies – both toy examples and real-world ones – to be used as common reference by researchers to explore and compare different approaches in

engineering agent-based AWs. A main one that we are investigating is in the health-care domain, about the development of personal medical assistant agents for medics engaged in rescue activities, working in hospital augmented environments [11].

To conclude, nowadays Augmented Reality is considered one of the pillars of Industry 4.0 [41], along with other ones such as Internet of Things and Autonomous Robots. We believe that the augmented world perspective discussed in this paper – widening the notion of augmentation beyond AR – provides a reference framework to locate these pillars, using agents as an effective high-level paradigm to design and understands these systems on top of enabling technologies.

References

1. Azuma, R.T.: A survey of augmented reality. *Presence: Teleoper. Virtual Environ.* 6(4), 355–385 (Aug 1997)
2. Berry, P.M., Gervasio, M., Peintner, B., Yorke-Smith, N.: Ptime: Personalized assistance for calendaring. *ACM Trans. Intell. Syst. Technol.* 2(4), 40:1–40:22 (Jul 2011)
3. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with JaCaMo. *Science of Computer Programming* 78(6), 747–761 (2013)
4. Bulling, A., Zander, T.O.: Cognition-aware computing. *IEEE Pervasive Computing* 13(3), 80–83 (July 2014)
5. Bulling, A., Cakmakci, O., Kunze, K., Reh, J.M.: Eyewear Computing – Augmenting the Human with Head-mounted Wearable Assistants (Dagstuhl Seminar 16042). *Dagstuhl Reports* 6(1), 160–206 (2016), <http://drops.dagstuhl.de/opus/volltexte/2016/5820>
6. Campbell, A.G., Stafford, J.W., Holz, T., O’Hare, G.M.: Why, when and how to use augmented reality agents (auras). *Virtual Reality* 18(2), 139–159 (2014)
7. Cap, M., Pechoucek, M., Jakob, M., Novak, P., Vanek, O.: Mixed-reality testbeds for incremental development of hart applications. *IEEE Intelligent Systems* 27(2), 19–25 (2012)
8. Chalupsky, H., Gil, Y., Knoblock, C.A., Lerman, K., Oh, J., Pynadath, D.V., Russ, T.A., Tambe, M.: Electric elves: Applying agent technology to support human organizations. In: *Proceedings of the Thirteenth Conference on Innovative Applications of Artificial Intelligence Conference*. pp. 51–58. AAAI Press (2001)
9. Costanza, E., Kunz, A., Fjeld, M.: *Human Machine Interaction*. chap. Mixed Reality: A Survey, pp. 47–68. Springer-Verlag, Berlin, Heidelberg (2009)
10. Craig, A.B.: *Understanding Augmented Reality*. Morgan Kaufmann (2013)
11. Croatti, A., Montagna, S., Ricci, A.: A personal medical digital assistant agent for supporting human operators in emergency scenarios. In: *X Int. Workshop on Agents Applied in Health Care (A2HC 2017) held at AAMAS 2017*. Sao Paolo, Brazil (2017)
12. Croatti, A., Ricci, A.: Towards the web of augmented things. In: *International Workshop on Engineering IoT Systems: Architectures, Services, Applications, and Platforms*. Gothenburg, Sweden (2017)
13. Dautenhahn, K.: Computation for metaphors, analogy, and agents. chap. Embodiment and Interaction in Socially Intelligent Life-like Agents, pp. 102–141. Springer-Verlag, Berlin, Heidelberg (1999)
14. Don, A., Brennan, S., Laurel, B., Shneiderman, B.: Anthropomorphism: From eliza to terminator 2. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. pp. 67–70. CHI ’92, ACM, New York, NY, USA (1992)
15. Engelbart, D.C., English, W.K.: A research center for augmenting human intellect. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. pp. 395–410. AFIPS ’68 (Fall, part I), ACM, New York, NY, USA (1968)

16. Fischer, J., Jiang, W., Kerne, A., Greenhalgh, C., Ramchurn, S.D., Reece, S., Pantidi, N., Rodden, T.: Supporting team coordination on the ground: Requirements from a mixed reality game. In: 11th Int. Conference on the Design of Cooperative Systems (COOP '14) (2014)
17. Freed, M., Carbonell, J., Gordon, G., Hayes, J., Myers, B.A., Siewiorek, D., Smith, S., Steinfeld, A., Tomasic, A.: Radar: A personal assistant that learns to reduce email overload. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence. pp. 1287–1293 (July 2008)
18. Guinard, D., Trifa, V., Wilde, E.: A resource oriented architecture for the web of things. In: Internet of Things (IOT), 2010. pp. 1–8 (Nov 2010)
19. Höllerer, T., Feiner, S.: Mobile augmented reality. Telegeoinformatics: Location-Based Computing and Services. Taylor and Francis Books Ltd., London, UK 21 (2004)
20. Holz, T., Campbell, A.G., O'Hare, G.M.P., Stafford, J.W., Martin, A., Dragone, M.: Mixed reality agents. Int. J. Hum.-Comput. Stud. 69(4), 251–268 (Apr 2011)
21. Kurkovsky, S.: Pervasive computing: Past, present and future. 5th IEEE International Conference on Information and Communications Technology (ICICT) (2007)
22. Li, C., Giampapa, J.A., Sycara, K.P.: Bilateral negotiation decisions with uncertain dynamic outside options. IEEE Trans. Systems, Man, and Cybernetics, Part C 36(1), 31–44 (2006)
23. Maes, P.: Agents that reduce work and information overload. Commun. ACM 37(7), 30–40 (Jul 1994)
24. Mann, S.: Wearable Computing as Means for Personal Empowerment. IEEE Computer Society Press, Fairfax, VA (May 1998), <http://www.eyetap.org/wearcam/icwc98/keynote.html>
25. Milgram, P., Kishino, F.: A taxonomy of mixed reality visual displays. IEICE Trans. Information Systems E77-D(12), 1321–1329 (Dec 1994)
26. Mitchell, T.M., Caruana, R., Freitag, D., McDermott, J., Zabowski, D.: Experience with a learning personal assistant. Commun. ACM 37(7), 80–91 (Jul 1994)
27. Modi, P.J., Veloso, M., Smith, S.F., Oh, J.: CMRadar: A personal assistant agent for calendar management. In: Proceedings of the 6th International Conference on Agent-Oriented Information Systems II. pp. 169–181. AOIS'04, Springer-Verlag, Berlin, Heidelberg (2005)
28. Myers, K.L., Berry, P., Blythe, J., Conley, K., Gervasio, M.T., McGuinness, D.L., Morley, D.N., Pfeffer, A., Pollack, M.E., Tambe, M.: An intelligent personal assistant for task and time management. AI Magazine 28(2), 47–61 (2007)
29. Okamoto, S., Scerri, P., Sycara, K.: Toward an understanding of the impact of software personal assistants on human organizations. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 630–637. AAMAS '06, ACM, New York, NY, USA (2006)
30. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems 17(3), 432–456 (2008)
31. Piekarski, W., Thomas, B.: ARQuake: The outdoor augmented reality gaming system. Commun. ACM 45(1), 36–38 (Jan 2002)
32. Poovendran, R.: Cyber-physical systems: close encounters between two parallel worlds. Proceedings of the IEEE 98(8) (2010)
33. Reeves, S.: Envisioning ubiquitous computing. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 1573–1582. CHI '12, ACM (2012)
34. Rhodes, B.J.: The wearable remembrance agent: A system for augmented memory. Personal Technologies 1(4), 218–224 (1997)
35. Ricci, A., Croatti, A., Brunetti, P., Viroli, M.: Programming Mirror Worlds: An Agent-Oriented Programming Perspective, pp. 191–211. Springer International Publishing (2015)
36. Ricci, A., Piunti, M., Tummolini, L., Castelfranchi, C.: The mirror world: Preparing for mixed-reality living. IEEE Pervasive Computing 14(2), 60–63 (2015)

37. Ricci, A., Pianti, M., Viroli, M.: Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems* 23(2), 158–192 (Sep 2011)
38. Ricci, A., Tummolini, L., Pianti, M., Boissier, O., Castelfranchi, C.: Mirror worlds as agent societies situated in mixed reality environments. In: *Revised Selected Papers of the International Workshops on Coordination, Organizations, Institutions, and Norms in Agent Systems X - Volume 9372*. pp. 197–212. Springer-Verlag New York, Inc., New York, NY, USA (2015)
39. Sadri, F.: Ambient intelligence: A survey. *ACM Comput. Surv.* 43(4), 36:1–36:66 (Oct 2011)
40. Satyanarayanan, M.: Pervasive computing: Vision and challenges. *IEEE Personal Communications* 8, 10–17 (2001)
41. Schwab, K.: *The Fourth Industrial Revolution Hardcover*. Crown Business
42. Shintani, T., Ito, T., Sycara, K.: Multiple negotiations among agents for a distributed meeting scheduler. In: *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'00)*. pp. 435–436. IEEE Computer Society, Washington, DC, USA (2000)
43. Starner, T.: The challenges of wearable computing: Part 1. *IEEE Micro* 21(4), 44–52 (Jul 2001)
44. Starner, T.: Project Glass: An Extension of the Self. *IEEE Pervasive Computing* 12(2), 14–16 (2013)
45. Sukthankar, G., Sycara, K.P.: Activity recognition for dynamic multi-agent teams. *ACM TIST* 3(1), 18 (2011)
46. Sukthankar, R., Davies, N., Siewiorek, D.P.: Activity-based computing. *IEEE Pervasive Computing* 7(undefined), 20–21 (2008)
47. Tambe, M.: Electric elves: What went wrong and why. *AI Magazine* 29(2), 23–27 (2008)
48. Wagner, T., Phelps, J., Guralnik, V., VanRiper, R.: Coordinators: Coordination managers for first responders. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*. pp. 1140–1147. AAMAS '04, IEEE Computer Society, Washington, DC, USA (2004), <http://dx.doi.org/10.1109/AAMAS.2004.97>
49. Weiser, M.: The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.* 3, 3–11 (July 1999)
50. Wellner, P., Mackay, W., Gold, R.: Computer-augmented environments: back to the real world. *Communications of the ACM* 36(7) (1993)
51. Xia, C., Maes, P.: The design of artifacts for augmenting intellect. In: *Proceedings of the 4th Augmented Human International Conference*. pp. 154–161. ACM (2013)

Distributed Transparency in Endogenous Environments: the JaCaMo Case

Xavier Limón¹, Alejandro Guerra-Hernández¹, Alessandro Ricci²

¹ Universidad Veracruzana, Departamento de Inteligencia Artificial, Sebastián Camacho No 5, Xalapa, Ver., México 91000
xavier120@hotmail.com, aguerra@uv.mx

² DISI, Università di Bologna, Via Sacchi, 3, Cesena, Italia 46100
a.ricci@unibo.it

Abstract. This paper deals with distribution aspects of endogenous environments, in this case, distribution refers to the deployment in several machines across a network. A recognized challenge is the achievement of distributed transparency, a mechanism that allows the agent working in a distributed environment to maintain the same level of abstraction as in local contexts. In this way, agents do not have to deal with details about network connections, which hinders their abstraction level, and the way they work in comparison with locally focused environments, reducing flexibility. This work proposes a model that creates a distinctive layer for environment distribution, which the agents do not manage directly but can exploit as part of infrastructure services. The proposal is in the context of JaCaMo, the Multi-Agent Programming framework that combines the Jason, CArtAgO, and MOISE technologies, specially focusing on CArtAgO, which provides the means to program the environment. The proposal makes an extensive use of the concept of workspace to organize the environment and transparently manage different distributed sites.

Keywords: Distributed environments, Endogenous environments, Environment Programming, JaCaMo framework.

1 Introduction

Traditionally, agents are conceived as entities situated in an environment, which they can perceive and modify through actions, also reacting to changes in it accordingly [16]. Not only that, but the agents' goal is to achieve an environment desired state. This conception of environment, as the locus of agent perception, action, reaction, interaction, and goals, stays true in current MAS development.

Two general perspectives are adopted when defined the concept of environment in MAS: exogenous, and endogenous [14]. The exogenous perspective is rooted in Artificial Intelligence, conceiving the environment as the external world, separated to the actual MAS, which can be only perceived and acted upon by agents. An example of this conception can be found in EIS [1]. In contrast,

the endogenous perspective, grown in the context of Agent-Oriented Software Engineering (AOSE) [10], conceives the environment as a first class abstraction for MAS engineering [17], that not only can be perceived and acted upon, but it can also provide services and tools for the agents to aid them in their tasks, and as such, it is designed and implemented as part of the whole system. An example of this conception is found in CArtaGO [13].

From a Software Engineering point of view, environments can also be of two types: local, and distributed. In the local case, the entirety of the environment is centralized in a single process, being the easiest case for implementation. Distributed environments, on the other hand, entail multiple processes, possibly across a network, to contain the environment, and presents various challenges in the implementation and conceptualization side. In this work, we expose the idea that, from the agents point of view, there should not be any difference between local and distributed environments, the right level of abstraction should be encouraged instead, recognizing this by the term Distributed Transparency.

Distribution is not a new topic in MAS, multiple MAS technologies such as JADE [2], have a mature support for it, but it is mostly centered in agent communication and interaction, not on the endogenous conception of environment which this works entails. In this regard, JaCaMo [4] is a better example, as it supports endogenous distributed environments. This support is practical, but it lacks distributed transparency as there is a clear conceptual distinction between local and distributed environments, so agents, and agent plan programmers, need to handle the difference, reducing in this way the flexibility of the system, and forcing the programmer to change the level of abstraction in each case.

Scalability and fault tolerance are also issues when dealing with distribution, a flexible configuration is required in order to deploy the system in different settings, allowing it to grow or change as network problems arise. A good example of a distributed deployment system for JADE is [6]. Returning to the case of JaCaMo, there is no support for fault tolerance, and it lacks proper configuration facilities for distributed deployment.

This work proposes an extension to the Agents & Artifacts model of JaCaMo for modeling distributed transparent environments, while giving insights of how to address distributed deployment and fault tolerance. The outcome is an implemented JaCaMo-oriented infrastructure and Agent API that gives support to the mentioned requirements, while extending the dynamics and possibilities of MAS programming in general.

The paper is organized as follows. Section 2 briefly introduces the JaCaMo framework, addressing its distributed model. Section 3 introduces the problems present in the JaCaMo distributed model, presenting a proposal to solve them, first in an intuitive and informal manner, and then formally. Section 4 features different aspects of the implementation, such as the general architecture, and configuration and deployment. Section 5 discusses a case study that shows how the new JaCaMo-oriented implementation approach compares to current JaCaMo, giving code examples. Being a work in progress, section 6 discusses vari-

ous topics regarding future work, including proper evaluation and fault tolerance implementation. Finally, section 7 closes this paper with a conclusion.

2 Background

Although the discussion here is about endogenous environments in general, we adopt the JaCaMo [4] framework to implement our proposed model and guide our discussion, this is due the fact that, to the best of our knowledge, it has the most mature implementation of endogenous environments for MAS. As such, a brief introduction of JaCaMo is presented in this section. JaCaMo is the result of the orthogonal composition of three technologies for MAS: Jason [5] (taken as a proper name inspired by Greek mythology), CArtaGo [13] (Common AR-Tifact infrastructure for AGents Open environments), and MOISE [9] (Model of Organisation for multi-agent SystEms).

Jason provides the means for programming autonomous agents. It is an agent oriented programming language that entails the Belief-Desire-Intention (BDI) approach, it is based on the abstract language *AgentSpeak(L)* [12]. Apart from its solid BDI theoretical foundations, the language offers several facilities for programming Java powered, communicative MAS. Communication in Jason is based on Speech Acts, as defined in KQML [7].

CArtaGo provides the means to program the environment, following an endogenous approach where the environment is part of the programmable system. In CArtaGo terms, the aspects that characterize a model for environment programming are the following [14]: 1) Action model: how to perform actions in the environment. 2) Perception model: how to retrieve information from the environment. 3) Environment computational model: how to represent the environment in computational terms. 4) Environment data model: how to share data between the agent and environment level to allow interoperability. 5) Environment distributed model: how to allow computational distributed environments. Aspects 1-3 are directly supported by artifacts [11], which are dynamical sets of computational entities that compose the environment and encapsulate services and tools for the agents. Artifacts are organized and situated in workspaces, which essentially are logical places (local or remote) where agents center their attention and work. Aspect 5 is supported by workspaces, but also partially by artifacts, as artifact actions can be executed remotely. Aspect 4, on the other hand, depends on the underlying agent programming language used and is not directly related to artifacts or workspaces.

MOISE provides the means to create agent organizations, which have the aim to control and direct agent autonomy in a general purpose system. To this end, it is possible to specify tree aspects: i) Structural, consisting on the different agent groups and roles that take part in the organization; ii) Functional, defined by social schemes, missions, and goals which direct the agent behaviour toward organization ends; and finally iii) Normative, defined through norms that bind roles to missions, constraining agent's behaviour when entering a group and playing a certain role.

2.1 JaCaMo and CArtAgO Distribution Model

As mentioned earlier, environment programming in JaCaMo is provided by means of CArtAgO, considering distribution in its model. At the higher level, distribution is achieved through workspaces, which serve as logical places where agents may center their attention, and where artifacts are situated. Agents can create, join, and quit workspaces. If an agent is in a workspace, it can use the artifacts situated there.

At the low level, nodes enable distribution. A node is a CArtAgO process that can be remote, where workspaces can be spawned. When a JaCaMo MAS is deployed, it is contained in a default node, that node is also the default for agents, which consider it as its local context, so workspaces created in that node are also local workspaces, but workspaces created in different nodes are considered remote workspaces. The distinction between remote and local workspace is not only conceptual, but also syntactical, requiring IP and port information at the agent level to manipulate remote workspaces. Figure 1 depicts the current CArtAgO environment model from the workspaces and nodes perspective. From the figure, it is apparent the fact that there is no connection between nodes, and in consequence between workspaces in different nodes, needing to explicitly know the IP address and port of each node.

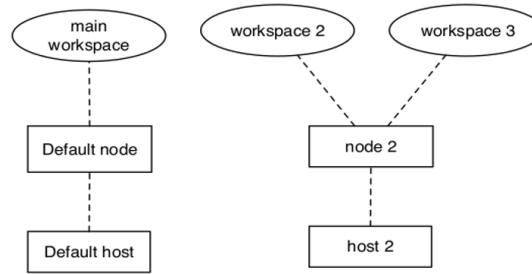


Fig. 1. Current CArtAgO environment model depicting multiple nodes and workspaces deployed.

More concretely, the following code snippet shows the difference in the JaCaMo API for the local and remote versions of join workspace, taking as a basis figure 1 where *default node* represents the local node, and *node2* a remote one:

```
1 | joinWorkspace("main", WspId1);
2 | joinRemoteWorkspace("workspace2", "192.168.0.2:8091", WspId2);
```

3 Proposal

Environment programming in JaCaMo comes with various shortcomings regarding distributed settings, being the most important the fact that local and remote workspaces are defined and treated differently, which derives in the following problems: i) There is not distributed transparency for agents, being forced to directly manipulate network information, making network distinctions between workspaces. ii) The underlying environment topology is difficult to represent and exploit by the agents as it does not follow any structure or workspace relations beyond the sharing of the same node. All of these problems have the consequence of reducing the abstraction level in which agents work, impacting flexibility and environment exploitation as well.

Another problem is the lack of proper configuration facilities to allow the inclusion of remote workspaces information at deployment time, meaning that host information for remote workspace spawning need to be hard-coded on the agent programs or externally supported. To spawn a remote workspace, a CArtaGO node needs to be running on the destination host, and there is not any integrated facility to manage them automatically when needed. Furthermore, the current distributed implementation does not exhibit any degree of fault tolerance, this is specially important for possible network connection problems that may arise in a distributed system.

In this section, a proposal to solve the identified problems is presented. A separation between environment and infrastructure is suggested. The environment is represented as a hierarchical tree structure, which represents the topology. In this tree, each node is a workspace which actual physical placement on the distributed system is irrelevant. Workspaces may be deployed in different places, but for the agents point of view, it only matters their placement in the topology. A workspace may be the logical parent of another one, multiple workspaces can be in the same physical place, and there is no restriction about how the topology may be organized, e.g.; workspaces on the same physical place may be on different branches. This allows to organize environments as it is usually done in CArtaGO, but in a more structured way, also supporting remote workspaces transparently.

In a practical sense, each workspace in the tree is represented by a path starting at the root workspace, these paths brings the notion of logical placement that agents require to organize and exploit their environment. We adopt a Unix-like path format to represent this placement, but using a "." instead of a "/", following Jason syntax. These paths are used by the agents to execute environment related actions, such as creating new workspaces or joining one. From the proposed JaCaMo API, there is no difference between local and remote actions related to workspaces. For example, returning to the code snippet presented in section 2.1 for joining local and remote workspaces, which it is related to figure 1; with the proposal, a workspace topology would be created, a possibility is to have *workspace2* and *workspace3* as direct descendants of the root workspace *main*, with this setting the associated code snippet is as follows:

```

1 |   joinWorkspace("main", WspId1);
2 |   joinWorkspace("main.workspace2", WspId2);

```

As in current CArtAgO, agents may work in multiple workspaces at the same time, but the concept of current workspace is dropped since in actuality all the joined workspaces should be considered the current context of working. Nevertheless, agent may specify the target workspace for an action. A new introduced concept is the home workspace of an agent, which it is the workspace where the agent is initially deployed, serving as a relative reference to other places in the topology, providing a default place for the agent, and also serving as the default workspace to execute actions when a target workspace is not specified.

On regard of the infrastructure, a layer is added to manage distribution, this layer provides the required services for the agents to exploit their distributed environment. These services include: i) Workspace management, so agents can create, join, and quit workspaces no matter their physical placement; ii) Topology inspection, so agents can reason about the current topology organization and do searches concerning workspaces; iii) Workspace dynamics observation, so agents can know when other agents manage workspaces, or when workspaces disconnect and reconnect after a network problem; iv) Disconnection and fault tolerance to manage and recuperate from network problems, which it is currently left as future work, but initially designed as presented in section 6.2 . We believe that the set of mentioned services do not only bring distributed support, but also enhance the dynamics of MAS in general, extending its possibilities.

3.1 Formal description

JaCaMo assumes an endogeneous approach to MAS, i.e., the environment is an explicit part of the system:

Definition 1. *A MAS is composed by a set of agents (Ags), their environment (Env), and an infrastructure (Infr) running both of them:*

$$MAS = \{Ags, Infr, Env\}$$

The set of agents is composed by $n \geq 1$ agents:

$$Ags = \{a_1, \dots, a_n\}$$

Each agent, as usual, is composed by beliefs, actions, and other elements equal to:

$$a_i = \{bels, acts, \dots\}$$

By default, when created, an agent includes minimally:

$$a_i = \{joined(home)\}, \{join, quit, create\}, \dots\}$$

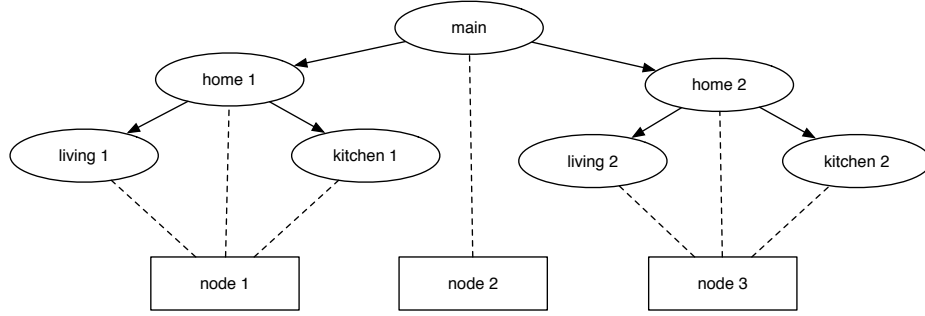


Fig. 2. The intended view of an endogeneous environment.

which means that every agent believes he has joined a home workspace, and has actions to join, quit, and create workspaces; and update the information about the environment.

Figure 2 illustrates the intended view of the environment in this proposal. First, the environment, properly speaking, is a tree of workspaces, expressing a kind of spatial relation among workspaces, e.g., the kitchen 1 is at the home 1. Second, nodes and hosts are not part of the environment, but are defined as part of the infrastructure of the MAS, nevertheless, workspaces keep information about its corresponding physical node.

The infrastructure is a layer hidden to the agents, that gives the low level support to distribution, formally defined as:

$$Infr = \{Nodes, Hosts\}$$

where:

- $Nodes = \{node_1, \dots, node_k\}$ is a set of CArTAgo nodes, i.e.; processes, possibly remote, where workspaces can be created. Each $node_i$ is a tuple $\langle ni, SWsps, hi, port \rangle$, where ni is a unique identifier for the node; $SWsps \subseteq W$ is the set of spawned workspaces in the node, containing at least a default workspace for the node; hi is an identifier of the host computer where the node exists; and $port$ is the host port used by the node process.
- $Hosts = \{host_1, \dots, host_p\}$ is the set of available computer devices on the distributed system. Each $host_i$ is a tuple $\langle hi, HNodes, ip \rangle$, where hi is a host unique identifier, $HNodes \subseteq Nodes$ is a set of hosted nodes, and ip is the IP address of the computer.

Formally, the environment Env is defined as a graph:

$$Env = \{W, E\}$$

where:

- $W = \{w_1, \dots, w_i\}$ is a finite, non-empty set of $i \geq 1$ workspaces that contain artifacts. Each $w_i = \langle idW, name, ni \rangle$, where idW is a unique identifier for the workspace, $name$ is a logical name, and ni is a reference to the CArTAgo node in *Infr* that contains w_i . The *node* element establishes a connection between the environment and the infrastructure, in order to forward agent actions to the destined physical place.
- $E \subset W^2$ is a set of edges over the workspaces, such that Env is minimally connected, i.e., it is a rooted tree that represents the environment topology.

For instance, following Figure 2, $Env = \{W, E\}$, and considering for simplicity only the name of each w_i , such that:

- $W = \{main, home1, home2, living1, kitchen1, living2, kitchen2\}$
- $E = \{\{main, home1\}, \{main, home2\}, \{home1, living1\}, \dots\}$

The expression $w_1.w_2 \dots w_n$ denotes a path on Env , if:

- $w_i \in W$ for $i = 1, \dots, n$;
- $\{w_{i-1}, w_i\} \in E$ for $i = 2, \dots, n$.

Abusing a little bit of the notation, we can write $w_1 \dots w_n \in Env$. For instance, $main.home1.living1 \in Env$. Some useful operations over paths, include:

- $last(w_1.w_2 \dots w_n) = w_n$
- $butlast(w_1.w_2 \dots w_{n-1}.w_n) = w_1.w_2 \dots w_{n-1}$
- $add(w, w_1.w_2 \dots w_n, Env) = w_1.w_2 \dots w_n.w$. This involves adding w to W , and $\{w_n, w\}$ to E in Env .
- $del(w, w_1.w_2 \dots w_n.w, Env) = w_1.w_2 \dots w_n$. This involves deleting w from W , and $\{w_n, w\}$ from E in Env .

In what follows, the transition rules related to environment agent actions are described, workspaces denote paths in the environment.

Joining a workspace An agent can ask himself about the workspaces he has currently joined: $ag_{bels} \models joined(w)$, if and only if, w is a workspace currently joined by the agent. Recall that by default $ag_{bels} \models joined(home)$. An agent can join different workspaces concurrently, so that $ag_{bels} \models joined(Ws)$ unifies Ws with a list of the workspaces joined by the agent. Two transition rules define the behavior of the action *join*. First, an agent can join a workspace w , if and only if w is a path in the environment Env and it is not believed to be already joined:

$$\begin{array}{c}
 \text{(join}_1\text{)} \quad \frac{join(w) \mid w \in Env \wedge ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag', Env \rangle} \\
 s.t. \ ag'_{bels} = ag_{bels} \cup \{joined(w)\}
 \end{array}$$

Second, nothing happens if an agent tries to join a previously joined workspace:

$$\text{(join}_2\text{)} \quad \frac{join(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

Any other use of *join* fails.

Quitting workspaces An agent can quit the workspace w if he believes he had joined w . The agent forgets such belief.

$$(\mathbf{quit}_1) \quad \frac{quit(w) \mid ag_{bels} \models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag', Env \rangle} \\ s.t. \ ag'_{bels} = ag_{bels} \setminus \{joined(w)\}$$

If the agent tries to quit a workspace he has not joined yet, nothing happens:

$$(\mathbf{quit}_2) \quad \frac{quit(w) \mid ag_{bels} \not\models joined(w)}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

Creating workspaces An agent can create a workspace w , if it is not a path in the environment, but $butlast(w)$ is one:

$$(\mathbf{create}_1) \quad \frac{create(w) \mid w \notin Env \wedge butlast(w) \in Env}{\langle ag, Env \rangle \rightarrow \langle ag, Env' \rangle} \\ s.t. \ Env' = add(last(w), butlast(w), Env)$$

Observe that the result of creating a workspace must be propagated to the rest of the agents in the MAS. This could be done by the infrastructure, or broadcasting the *add* operation. The actual node where the workspace is going to be created is decided by the infrastructure following a policy, by default the infrastructure spawns the workspace on the same node where its parent workspace is running.

Trying to create an existing workspace does nothing:

$$(\mathbf{create}_2) \quad \frac{create(w) \mid w \in Env}{\langle ag, Env \rangle \rightarrow \langle ag, Env \rangle}$$

4 Implementation

The model introduced on section 3 is open enough to allow different implementations. This section presents a practical possibility, intended to be integrated with JaCaMo. The core implementation and main design choices are related to the general architecture, and configuration and deployment.

4.1 General architecture

The architecture to support agent services is based on the concept of Node, which refers to the *Nodes* element in *Infr*. Nodes represent independent CArtaGo processes, possibly remote, running on a given host (*Hosts* element in *Infr*),

and associated to a port. Nodes are the main abstraction to manage workspaces (W element in Env), and as such, they provide all the necessary tools to create, join, and quit workspaces, as well as the means to communicate with other nodes in order to maintain a consistent workspace topology, and properly dispatch topology related events. The workspace topology corresponds to the E element in Env . A NodeArtifact is the gateway used by an agent to interact with the node services and to observe the distributed environment. There is a NodeArtifact in each workspace, and every agent has access to one of them, which one depends on its *home* workspace, which in turn it is intended to be on the same node as the agent process.

Nodes communicate between each other following a centralized approach: one node is designated as the central node, this is usually the node deployed by default by JaCaMo, so every change on the topology is inspected and approved by a single node, and the associated actions and events are dispatched from there. This centralized approach makes possible to maintain a consistent topology, avoiding run conditions. To exemplify node communication, the workflow for creating a new workspace is the following:

- An agent that wants to create a workspace issues the action to its corresponding NodeArtifact, passing a tree path.
- The artifact checks if the tree path is consistent with the topology tree, if it is, it sends a request to the central node.
- The central node issues a request to the end node where the workspace is actually going to exist. By default, it chooses as end node the same as the parent workspace from the path given.
- The end node creates the workspace and returns control to the central node.
- The central node makes the corresponding changes to the workspace topology and communicates the success to the original requesting node. It also dispatches a create and tree change event to the other nodes.

As the node model is centralized, there exists the concern of a single point of failure, that is why all nodes maintain redundant information about the topology, so it is possible to recuperate from a central node dropping, as any node can take the role of central node. The topology structure is also lightweight, which speeds up the tree synchronization among nodes, this synchronization is only required when there is a change in the topology. This redundancy also allows to boost the efficiency of operations such as joining and quitting workspaces, since those operations only need to read from the topology, so the local copy is used in those cases. Communication with the central node is only required in cases where a change in the topology is required. We believe that in traditional environment management, it is more common for the agents to join and quit workspaces than to create new ones.

4.2 MAS configuration and deployment

To ease the deployment of the distributed infrastructure is a goal of our overall proposal, this means to be able to configure and launch the desired hosts, nodes,

and workspaces that will take part in the MAS from the start. It is also possible to manually add new nodes after deployment. The idea is to extend the deployment of JaCaMo, where only workspaces are considered. JaCaMo uses a text file known as the JCM file to configure the deployment of the MAS. The intention is to further include fields in this file to also configure host, and nodes for distributed systems; and add the facilities to automatically launch CArtaGo nodes in remote machines through a daemon service.

The changes to the JCM file include:

- Host configuration: assign a logical name and IP address to each host.
- Node configuration: assign a logical name for the node, i.e.; the name of the default workspace; the related host name; and optionally a port number.
- Workspaces configuration: relate each workspace to a specific node.
- Lib file: the path to a jar file containing all the necessary files to launch CArtaGo nodes. This includes custom artifacts binaries, third party libraries, custom classes binaries, and any configuration file. This file is intended to be shared among all nodes.

5 Case study

In order to show how to exploit the proposed distributed model and implementation, and how it compares to the current version of JaCaMo, a small case study is presented in this section. This case study pretends to show the new proposed agent API, and the flexibility of the proposed model, focusing mainly on workload distribution, which is one of the aspects that can be enhanced, but other aspects such as fault tolerance, reactivity on environment changes, and more complex agent organizations and dynamics are also possible.

The case study consists on constantly fetching current weather information for every city in a country, and with that information, construct weather prediction models for each city, so the models could be consulted by end users through a user interface. The construction of each model is an online learning [3] task that can be heavy on the computational side, so work distribution is desirable. To simplify the distributed setting, assume that the cities of the country are divided in west-cities and east-cities, one computer is in charge of the models from the west-cities, and another one from the models of the east-cities; furthermore, information fetching need to be quick and constant, and also the end user searching service, so one computer takes care of both. The described setting yields a total of three different computers in the distributed system.

Workflow is modeled through 3 agent programs: *fetcher*, which fetches weather information and forwards it to a destination depending the type of city; *learner*, which task consist on creating online weather prediction models for each city with the data provided by the *fetcher* agent; and finally, a *searcher* agent, which attends end user requests to retrieve information from the learned models.

When implementing this case study in current JaCaMo some problems will arise: the IP addresses of the different computers should be hard-coded in the agent programs; every CArtaGo node representing a remote workspace should

be manually started in every computer on the distributed setting; startup plans should be implemented in order to focus the attention of each agent in its designated place; when the searcher agent has to collect information about learned models to attend a petition, it necessarily has to ask learner agents about its location, not only considering the workspace but also the ip address where the workspace is, making also necessary to distinguish between local and remote workspaces when the agent intend to return the result to the end user. A better solution using our new approach that solves all the mentioned problems, and better exploits the environment is presented next.

- A possible JACAMO configuration file for this example, following the idea from section 4.2, is the following. For the sake of clarity, artifact related configuration, and MOISE related organization is not shown.

```

1  mas weather {
2      host c1 {
3          ip: 192.168.0.2
4      }
5      host c2 {
6          ip: 192.168.0.3
7      }
8      node west {
9          host: c1
10         port: 8080
11     }
12     node east {
13         host: c2
14         port: 8080
15     }
16     agent fetcher : fetcher.asl {
17         home: main
18     }
19     agent west_learner : learner.asl {
20         home: main.west
21     }
22     agent east_learner : learner.asl {
23         home: main.east
24     }
25     agent searcher : searcher.asl {
26         join: main.central
27     }
28     lib_file : /home/system/wheather.jar
29 }

```

Note that workspaces *main*, *main.west*, and *main.east* are implicitly created as they are the default workspaces for the nodes, e.g.; *main* is the node deployed by default by JaCaMo.

In our proposed model, agents refer to workspaces only through their path in the topology, giving in this way a more structured sense of placement. As in UNIX file system paths, tree paths can be considered absolute or relative. A path is relative from the *home* workspace of the agent, and should start with a single dot, any other path is considered absolute and must begin at the root workspace (by default *main*).

A simplified version of the agent programs is presented next.

– fetcher:

```
1 //creations and initializations omitted
2 +!fetch : true <-
3   getData(Data);
4   category(Data, Cat);
5   if(Cat == "west") {
6     .send(west_learner, tell, add(Data));
7   }
8   else {
9     .send(east_learner, tell, add(Data));
10  };
11  !fetch.
```

– learner:

```
1 //initialization of agent omitted
2 +add(Data): true <-
3   getCity(Data, City);
4   .concat(".", City, Path); //from home
5   createWorkspace(Path); //does nothing if already present
6   joinWorkspace(Path); // does nothing if already joined
7   //creations and initializations omitted
8   addData(Data)[wsp(Path)]; //route action to wsp
9   induceModel[wsp(Path)].
```

– searcher:

```
1 +search(Query) : true <-
2   //Query is just the name of the city
3   .concat(".*", Query, RegExp);
4   searchPaths("main", RegExp, [H | T]);
5   .length(List, Len);
6   if(Len > 0) {
7     joinWorkspace(H);
8     getForecast(Forecast)[wsp(H)];
9     quitWorkspace(H);
10    sendReply(Forecast);
11  }.
```

Some of the actions from the agent programs correspond to the API of the proposed model, such actions are described as follows.

- *joinWorkspace(+WspPath, –WspId)*: the agent adds a specified workspace to its joined workspaces list, obtaining a workspace ID.
- *quitWorkspace(+WspPath)*: removes a specified workspace from its joined workspaces.
- *createWorkspace(+WspPath)*: creates a new workspace on the specified path. By default, the workspace will actually be spawned on the same CArtaGo node as the parent workspace derived from *WspPath*, this allows workload management for workspaces.
- *searchPaths(+PointPath, +RegExp, –WspPathList)*: returns a list with the workspaces that follow a certain regular expression, the search is restricted to the subtree given by *PointPath*. This action exemplifies how the topology organization may be exploited.

It is worth mentioning that the actual API is more extensive, including perception, events, and more actions related to the environment. For example, it is possible for the agents to react to the creation of a new workspace through the event *created(WspPath)*, or analyze and exploit the current topology organization through the perception *topology_tree(List)* where *List* is of the form: $[main, [subNode_1, [subsubNode_1, [...], subsubNode_m]], \dots, [subNode_n]]$.

As the example shows, agents do not concern about computer distribution, they simply work with workspaces: learner agents organize their work in workspaces that can be on different computers; and the searcher agent can reach any workspace directly, not relying on agent communication, but directly acting through the knowledge of the topology. Deployment is also greatly enhanced since the distributed setting is properly configured beforehand, and the launching of nodes is automatic.

6 Discussion and future work

As a work in progress, our proposal still lacks a proper treatment of different aspects such as evaluation, and fault tolerance. This sections briefly discusses and outlines these topics, which are considered as immediate future work.

6.1 Evaluation

Our proposed model, while introduced in some formal way, still needs a proper formal evaluation, for this end, the adoption of a more formal representation such as the ones proposed in [8], and [15] seem to be required.

Concerning performance evaluation, with the adopted centralized model, it is required to asses scalability issues that may arise as nodes are added to the MAS. In case of finding such problems, we can still improve some of the required subprocess, as the synchronization of the topology among all nodes, and event propagation, which could be distributed.

6.2 Fault tolerance

Given the proposed architecture, connection loss is the same as node dropping, and as such it directly impacts the topology tree structure used by agents as all the corresponding workspaces are also lost. An intuitive idea of how fault tolerance could be implemented following our design choices is described next.

Following the overall node organization introduced in section 4.1, all nodes maintain a keepalive connection with the central node, and a ordered list of connected nodes. If a node losses connection, then the central node issues the corresponding dropping event to the rest of the nodes, and modifies the topology tree structure accordingly. The disconnected node tries to establish a connection with the rest of the nodes, following the order of the connected nodes list, this being useful on the case that several nodes lost connection or the central node dropped. With the available nodes (it may be only one), the node in the upper

position on the connected nodes list is designated as the new central node, issuing the corresponding disconnection events and creating a new tree node structure where every default workspace from the nodes is on the same level. The new central node keeps trying to reconnect to the original central node for a period of time.

When successfully reconnecting, the original central node will try to return the topology to the way it was before the problem, but sometimes that would not be possible, e.g.; when one of the nodes keep missed. It is strongly recommended that every default workspace corresponding to a node is mounted on the same upper tree level of the tree, that way when reconnecting, the tree structure will keep consistent with the way it was before, otherwise the tree topology may vary in an unexpected manner, which can be problematic on certain applications. After the node tree structure is recreated, the reconnecting nodes return to work with the original central node, and the central node triggers the corresponding reconnection events.

7 Conclusion

The introduced model is a step forward to improve environment programming for MAS, it addresses issues related to distribution, which are important for a wide variety of applications. We see distributed transparency as the most important contribution of this work, as Multi-Agent Systems are intended to raise the level of abstraction in software development, as compared with other industry established programming paradigms such as POO. Proper abstraction levels for aspects such as concurrency management are already accomplished, but distributed computing is still an important topic to improve.

With a solid foundation for distributed environment programming, it is possible to address new challenges like MAS interoperability, which refers to the integration and collaboration of independent Multi-Agent Systems. An extension to the proposed model and implementation is envisaged to support MAS interoperability features, such as MAS composition where two different MAS can fuse together to extend the scope of their work; and also MAS attachment, where a mobile MAS can temporally join a MAS in order to exploit services. These features bring new possibilities to the dynamics of MAS in general, and are also interesting from the software engineering point of view as they allows an upper level of flexibility and scalability.

References

1. Tristan M Behrens, Koen V Hindriks, and Jürgen Dix. Towards an environment interface standard for agent platforms. *Annals of Mathematics and Artificial Intelligence*, 61(4):261–295, 2011.
2. Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade—a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

3. Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11(May):1601–1604, 2010.
4. Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.
5. Rafael H. Bordini, Jomi F. Hübner, and Mike Wooldridge. *Programming Multi-Agent Systems in Agent-Speak using Jason*. John Wiley & Sons Ltd, 2007.
6. Lars Braubach, Alexander Pokahr, Dirk Bade, Karl-Heinz Krempels, and Winfried Lamersdorf. Deployment of distributed multi-agent systems. In *International Workshop on Engineering Societies in the Agents World*, pages 261–276. Springer, 2004.
7. T. Finin et al. An overview of KQML: A knowledge query and manipulation language. Technical report, University of Maryland, CS Department, 1992.
8. Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.
9. Jomi F Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.
10. James J Odell, H Van Dyke Parunak, Mitch Fleischer, and Sven Brueckner. Modeling agents and their environment. In *International Workshop on Agent-Oriented Software Engineering*, pages 16–31. Springer, 2002.
11. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.
12. A.S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.
13. A. Ricci, M. Viroli, and A. Omicini. Construenda est cartago: Toward an infrastructure for artifacts in MAS. *Cybernetics and systems*, 2:569–574, 2006.
14. Alessandro Ricci, Michele Pionti, and Mirko Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, 2011.
15. Alessandro Ricci, Mirko Viroli, and Maurizio Cimadamore. Prototyping concurrent systems with agents and artifacts: Framework and core calculus. *Electronic Notes in Theoretical Computer Science*, 194(4):111–132, 2008.
16. Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.
17. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.

Give Agents Some REST: A Resource-Oriented Abstraction Layer for Internet-scale Agent Environments

Andrei Ciortea¹, Olivier Boissier¹, Antoine Zimmermann¹, and
Adina Magda Florea²

¹ Univ. Lyon, MINES Saint-Étienne, CNRS,
Lab Hubert Curien UMR 5516, F-42023 Saint-Étienne, France
{andrei.ciortea,olivier.boissier,antoine.zimmermann}@emse.fr
² Department of Computer Science, University “Politehnica” of Bucharest,
313 Splaiul Independentei, 060042 Bucharest, Romania
adina.florea@cs.pub.ro

Abstract. To keep up with current technological developments, the engineering of multi-agent systems (MAS) has to provide solutions to: (i) support large scale systems, (ii) cope with open systems, and (iii) support humans in the loop. In this paper, we claim that the World Wide Web provides a suitable middleware for engineering MAS that address these challenges in an integrated manner. Even though approaches to engineer Web-based MAS have already been explored in the MAS community, existing proposals do not achieve a complete integration with the Web architecture. We approach this problem from a new angle: we design the *agent environment* as a *hypermedia application*. We apply REST, the architectural style of the Web, to introduce a resource-oriented abstraction layer for agent environments that decouples the *application environment* from its *deployment context*. Higher-level environment abstractions can then be implemented on top of this lower-level abstraction layer. To demonstrate our approach, we implemented a multi-agent application for the Internet of Things in which software agents can seamlessly navigate, use and cooperate in an environment deployed over multiple Web services (e.g., Facebook, Twitter) and constrained devices.

Keywords: Agent environments, service-oriented computing, REST, Web architecture, Web of Things, Internet of Things

1 Introduction

Over the last decade, the agent environment has gained broad recognition as a *first-class abstraction* in multi-agent systems (MAS) [39]: it is a key component designed and programmed with clear-cut responsibilities, such as mediating interaction among agents and access to the deployment context (e.g., physical devices, digital services). The increased emphasis on agent environments in MAS also raises new challenges to be addressed. Three research topics of growing

importance are to design agent environments that [38]: (i) support *large scale systems*, (ii) can cope with *open systems* in which components are deployed and evolve at runtime, and (iii) *support humans in the loop*.

If we are to examine existing software systems based on the above criteria, the World Wide Web is arguably the most scalable, versatile, and human-centric software system deployed on the Internet. In fact, the Web was specifically designed to be an Internet-scale and long-lived system in which components can be deployed and can evolve independently from one another at runtime [13]. More recently, on account of its architectural properties, the Web is emerging as the application layer for the Internet of Things (IoT), an initiative known as the Web of Things (WoT) [23,40]: physical devices are integrated into the Web such that software clients can access them in a uniform manner using Web standards. The WoT vision is rapidly being implemented through combined standardization efforts of the W3C WoT Working Group³, the IETF Constrained RESTful Environments Working Group⁴, and the IRTF Thing-to-Thing Research Group⁵. The World Wide Web is turning into a middleware for most systems envisioned on the Internet, and its huge success comes from its carefully designed architectural properties.

In this paper, we take a deep look into the rationale behind the modern Web architecture. Our claim is that we can apply the same rationale to address the three above-mentioned research topics in an integrated manner. The novelty of our approach is to design the *agent environment* as a *hypermedia application*. We apply REST [14], the architectural style of the Web, to introduce a resource-oriented abstraction layer for agent environments that decouples the *application environment*⁶ from its *deployment context*. This abstraction layer is based on *socio-technical networks (STNs)* [2], that is dynamic networks of humans, software agents and artifacts interrelated in a meaningful manner via typed relations. We use STNs to address HATEOAS, one of the architectural constraints that is really central to REST and the modern Web architecture.⁷ To the best knowledge of the authors, this proposal is the first approach to engineer MAS that are completely aligned with the Web architecture.

This paper is structured as follows. Section 2 presents the REST architectural style and discusses related work on engineering Web-based MAS. Section 3 presents our approach in further detail. Section 4 presents an implementation of this approach to develop a multi-agent application for the IoT in which software agents can seamlessly navigate and use an agent environment deployed over Facebook, Twitter, and multiple constrained devices.

³ <http://www.w3.org/WoT/WG/>, last accessed: 08.02.2017.

⁴ <http://datatracker.ietf.org/wg/core/>, last accessed: 08.02.2017.

⁵ <http://datatracker.ietf.org/rg/t2trg/>, last accessed: 08.02.2017.

⁶ That is to say, the part of the environment designed and programmed for the application at hand [39].

⁷ In previous publications, STNs have been applied as a means to bring the multi-agent paradigm to IoT application development [4], and to enhance discoverability across otherwise siloed IoT environments [3].

2 Background and Related Work

In Section 2.1, we present the key principles behind REST and the architectural properties they induce. Then, in Section 2.2 we focus our attention on existing approaches to engineer Web-based MAS. We conclude that none of the existing approaches are completely aligned with the REST principles.

2.1 The REST Architectural Style

Representational State Transfer (REST) is an architectural style for distributed hypermedia systems [14]. REST was designed to meet the needs of the Web as an *Internet-scale system* in which the overall performance is dominated by network communication rather than computation-intensive tasks [13]: component interactions consist of large-grain data objects transferred over high-latency networks that span across geographical and organizational boundaries. To this end, REST was designed to minimize latency and network communication, and at the same time maximize the independence of components and the scalability and visibility of component interactions [14,13]. These combined characteristics transformed the Web into an *Internet-scale, open, and long-lived* system.

REST in a Nutshell REST is defined as a *coordinated set of architectural constraints* [14]. When applied to a software architecture, REST constrains only those portions of the architecture considered to be essential for *Internet-scalability* and *openness* [13]. In particular, REST focuses on the *semantics of component interactions*, in contrast to other architectural styles that focus on the *semantics of components*. REST can thus be used in conjunction with component-based specifications, such as the FIPA Abstract Architecture [15] or the reference model for agent environments proposed in [39].

Interaction in a REST-style system follows the *request-response* pattern: a *client* (e.g., a Web browser) issues a request, a *server* processes the request and returns a response. REST components (i.e., origin clients, intermediaries, origin servers) interact by transferring *representations of resources*. A *resource* is the key abstraction of information in REST, where “any information that can be named is a resource” [14]. The interaction is *stateless*, which improves the scalability of server implementations, and performed through a *uniform interface* that hides the heterogeneity of component implementations. When combined, the *stateless interaction* and *uniform interface* constraints enable the use of *intermediaries* that can interpret requests almost as well as their intended recipients. Intermediaries can then be deployed along the request-response path, for instance, to cache and reuse interactions, to distribute the workload across multiple servers, to enforce security or to encapsulate legacy systems [14], which are all important concerns in an *Internet-scale, open, and long-lived* system.

Hypermedia-driven Interaction To achieve a *uniform interface* between components, interaction in REST is *driven by hypermedia*, a constraint a.k.a.

Hypermedia As the Engine of Application State (HATEOAS) [14]. Without hypermedia-driven interaction, browsers would be tightly coupled to origin servers and unable to seamlessly navigate an *open* Web.

To best illustrate hypermedia-driven interaction, consider a typical Web application composed of multiple hyperlinked Web pages. This application is a finite state machine: each page represents a state and hyperlinks between pages represent transitions between states. Given the URI of one of the pages, a client can dereference the URI to retrieve an *HTML representation* of that page. This action thus triggers a transition to a new application state, and if the transition is completed successfully the client can now choose from a new set of Web pages (i.e., new reachable states). Both the *next reachable states* and the *knowledge required* to transition to those states are conveyed to the client *through hypermedia* retrieved from the origin server (e.g., via HTML forms).

In other words, given a URI as an *entry point* into a Web application, *standard Web transfer protocols* and *representation formats* (a.k.a. *media types*), the client should be able to discover new resources and how to use those resources at runtime. The client is thus loosely coupled to the origin server via standardized knowledge: it does not hard-code URIs or individual requests. This is an important difference from how most existing Web services work today.

HATEOAS is essential to achieve a uniform interface, which in turn is a central feature in REST [14]. The uniform interface allows components to be deployed and to evolve independently from one another, which is an important feature in an *Internet-scale*, *open*, and *long-lived* system.

2.2 Engineering Web-based MAS

In this section, we discuss approaches to use the Web as an infrastructure for distributed MAS. These approaches fall broadly in one of two categories: they use the Web either as a *transport layer*, or as an *application layer*. To the best knowledge of the authors, none of the existing approaches fully complies with the REST principles presented in the previous section.

The Web as a Transport Layer Systems that use the Web as a transport layer make limited use of its architectural properties, existing infrastructure and future extensions (see Section 6.5.3 in [14] for a detailed discussion).

The *Foundation for Intelligent Physical Agents (FIPA)* investigated the integration of software agents with Web services [21]. One of the main Web-related results stemming out of the standardization efforts is a FIPA specification for using HTTP as a transport protocol for messages exchanged among agents [16]. FIPA-compliant platforms that implement this specification include JADE[12], SPADE [22] and SEAGENT [11].

WS- Web services* also use the Web as a transport layer [32]. Given their widespread adoption in the early 2000s, the WS-* standards have had a strong influence on multi-agent research in service-oriented computing [25,37,24,18]. A number of MAS platforms support direct integration with WS-* Web services

(e.g., [33,5,11]), while others rely on a gateway component to mediate interactions between agent services and Web services (e.g., [20,31,36,29]).

The Web as an Application Layer More recent approaches to engineer Web-based MAS are based on REST-like Web services (commonly referred to as “RESTful”), which typically use HTTP as an *application protocol* (see [32]), but do not use hypermedia and thus do not support hypermedia-driven interaction. As a result, clients and origin servers are *tightly coupled* to one another, which is an important limitation when engineering Internet-scale, open systems.

In [9], the authors propose an approach to automatically translate agent services to Web services (and vice-versa), with the aim to support both the WS-* standards and REST. The generated REST-like Web services, however, are not resource-oriented and do not support hypermedia-driven interaction. In other proposals, REST-like Web services are used for agent communication. In [1], agents implement interaction protocols (e.g., FIPA Contract Net Protocol [17]) by creating and manipulating resources on a Web server. A more general approach to REST-inspired agent communication is taken in [35]: agents use *message repositories* to create *graphs of messages* following predefined interaction protocols. Note that the links among messages can help address the HATEOAS constraint, but details on the APIs exposed by the message repositories are not available (e.g., if they are hypermedia-driven).

Radigost [28] implements the FIPA abstract architecture and exposes most of the system functionality through a REST-like Web service. A gateway component allows agents hosted on other platforms to exchange messages with Radigost agents. However, the HATEOAS constraint is not addressed.

REST-A [19] provides a REST-like approach to design *agent environments*: agents perform actions on *resources* in their environment using a set of CRUD (create, read, update, delete) operations over HTTP. However, the HATEOAS constraint is not addressed.

To conclude, to the best knowledge of the authors, existing approaches to engineer Web-based MAS are either agnostic to REST, or do not support hypermedia-driven interaction. In both cases, this implies that they cannot fully inherit the architectural properties of the Web, such as *Internet-scalability* and *openness*.

3 A Resource-oriented Foundation for Agent Environments

In this section, we propose a resource-oriented abstraction layer for agent environments that goes beyond the state-of-the-art by fully conforming to REST. As illustrated in Figure 1, the purpose of this layer is to decouple the *application environment* from its *deployment context*. Agents can then interact with their environment while being agnostic to the underlying infrastructure. We present our approach in Section 3.1. Then, in Section 3.2 we focus our discussion on

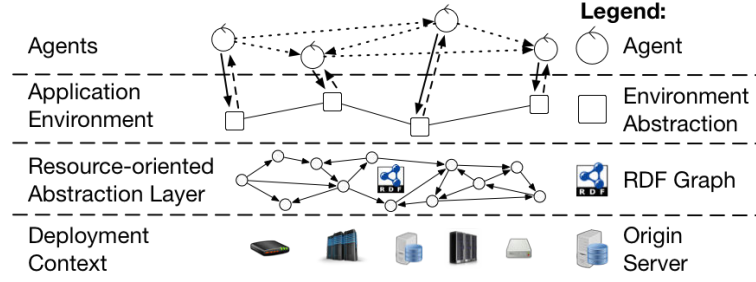


Fig. 1. Layers of an agent environment: the resource-oriented abstraction layer decouples the *application environment* from its *deployment context*.

achieving a *uniform interface* between the programmable part of an agent environment, which would typically run on a MAS platform, and its deployment context (e.g., physical devices, digital services).

3.1 Resource-oriented Agent Environments

Our approach to design the proposed resource-oriented abstraction layer for agent environments relies on *socio-technical networks (STNs)*, which were defined formally in [2]. STNs are grounded in multi-agent research, based on semantic Web technologies, and aligned with the Web architecture.

In the following, we first present the core abstractions introduced by the STN model [2]. Then, we show how they map to Web resources, and finally discuss the development of application environments on top of these abstractions.

Socio-technical Networks *Socio-technical networks (STNs)* are dynamic networks of humans and things (e.g., physical devices, digital services) interrelated in a meaningful manner via *typed relations* (e.g., friendship, ownership, provenance, colocation). STNs are situated in environments that span across the physical-digital space. Humans and things that are actively trying to influence the environment are modeled as *agents*, and things that passively augment the environment with new capabilities as *artifacts*. Agents can enter or leave STNs and “rewire” their networks in pursuit of their goals.

The *agent* and *artifact* abstractions have their roots in the *Agents & Artifacts meta-model* [30] and are motivated by the separation of concerns principle. First, these abstractions separate exhibited behavior from the actual entities, which allows developers and end-users to conceive of humans and heterogeneous things in a uniform manner. Second, agents and artifacts simplify the design of multi-agent applications by providing a clear separation between the logic that manipulates the environment and the logic that augments the environment. Third, agents and artifacts provide a modular approach to multi-agent application development when designed and implemented as loosely coupled components meant to be deployed and to evolve independently from one another at runtime.

Typed relations enable the *dynamic discovery* of agents and artifacts at run-time. For instance, say David owns a smart TV that can aggregate movie recommendations from other smart TVs owned by David’s friends. If David becomes friends with Bob, an explicit representation of this relation (e.g., via Facebook or Twitter) enables David’s smart TV to discover Bob and any smart TVs he might own.

From STNs to Web Resources An STN is reflected in the digital world by means of *digital artifacts* hosted by *STN platforms*. For instance, a lightbulb is a physical artifact that can have a digital counterpart hosted by an STN platform. The *digital lightbulb* represents the *physical lightbulb* and reflects its state. A software agent can then turn on or off the physical lightbulb by updating the state of the digital lightbulb. Similarly, if a human agent turns on or off the physical lightbulb, its state is reflected by the digital lightbulb and can be perceived by software agents.

Human and software agents can also be represented by digital artifacts, such as the *user accounts* they may hold on various STN platforms. User accounts are the agents’ proxies in STNs: it is assumed that the entity acting through a user account is acting on behalf of the agent who holds the account.

To deploy STNs on the Web, we map all entities in an STN (e.g., agents, physical artifacts, digital artifacts, STN platforms) to *Web resources* described in the *Resource Description Framework (RDF)* [10]. RDF is the data model of the Semantic Web and a natural choice for our purposes. We provide a Web ontology⁸, called hereafter the *STN ontology*, that developers can use to describe the various entities and relations among them in conformance with the proposed STN model [2]. Developers can further extend the STN ontology with domain- and application-specific knowledge.

STN-based Application Environments A key feature that STNs bring to agent environments is the *dynamic discovery* of agents and artifacts via crawling. This feature is central to *Internet-scale* and *open* MAS that can evolve over time. The state of a MAS is reflected in the resource-oriented abstraction layer, for instance whenever an agent joins or leaves the system, or whenever an artifact is created or deleted. Agents can navigate and act on the STN-based layer via environment abstractions (see Figure 1), which would typically run on a MAS platform. For MAS platforms that do not support environment abstractions, the manipulation of the STN-based layer can simply be hidden behind the agents.

Note that crawling STNs at scale would be inefficient, but STN-based application environments can provide agents with more efficient discovery mechanisms built on top of the proposed abstraction layer. By analogy, the Web enables the discovery of Web pages, but manually browsing the Web to search for a specific Web page is inefficient. Most people would generally start with a search engine to focus their search.

⁸ <http://w3id.org/stn/>

Another point worth emphasizing is that an STN-based application environment is not limited to its underlying STNs or even the Web: it can contain entities that are not reflected in its underlying STNs, and it can integrate non-Web components. For instance, the application environment we present in Section 4 includes an artifact that connects to a cloud-based MQTT broker⁹ in order to retrieve sensor readings from a Texas Instruments SensorTag¹⁰. The MQTT-based artifact is hidden behind the agent using it and not represented in the STN.

3.2 A Uniform Interface for STN-based Agent Environments

To decouple the application environment from its deployment context, it is necessary to achieve a *uniform interface* between the *origin clients* (e.g., MAS platforms, Web browsers) that access and manipulate *digital artifacts*, and the *STN platforms* that host the *digital artifacts*. The “hallmark” of achieving this uniform interface is that the resource-oriented abstraction layer is *driven by hypermedia* (see Section 2.1 for details): given the URI of a Web resource as an *entry point* into this layer, the origin client should then be able to seamlessly navigate and manipulate STNs using standard Web transfer protocols and media types, while being agnostic to the underlying STN platforms.

To achieve this uniform interface, we apply REST’s interface constraints¹¹ to STN platforms. We reformulate these constraints as follows:

- *Identification of digital artifacts*: digital artifacts hosted by an STN platform are always identified via URIs such that they can be referenced globally and independent of context.
- *Manipulation of digital artifacts via representations*: clients interact with STN platforms by exchanging *representations* of digital artifacts, where the representation of a digital artifact is an RDF serialization of the artifact’s current or intended state.
- Messages exchanged between clients, STN platforms, and any intermediaries in-between are *self-descriptive*.
- Interaction between clients and STN platforms is *driven by hypermedia*.

We further detail the last two interface constraints in what follows.

Self-descriptive messages In REST, components exchange representations of resources via *self-descriptive messages*: messages are self-contained¹², message semantics are defined by standard *methods* (e.g., the ones defined by HTTP or CoAP) and *representations formats* (a.k.a. *media types*), responses provide

⁹ Message Queuing Telemetry Transport (MQTT) is an application-level protocol for the IoT.

¹⁰ <http://www.ti.com/sensortag>, last accessed: 08.02.2017.

¹¹ See [13] for more details on the interface constraints in REST.

¹² Note that interaction is stateless between requests, so a component does not need to go beyond a single message to understand it.

explicit cache directives etc. Components must be able to reliably interpret exchanged messages using standardized knowledge. Any implementation-specific details about raw resources must remain hidden behind the uniform interface.

When it comes to choosing representation formats for digital artifacts, developers would generally have three options: (i) if applicable, reuse an existing media type¹³, (ii) define a new media type, or (iii) use standard RDF serialization formats, such as Turtle or JSON-LD, with ontologies for describing the states of digital artifacts.

The first option, reusing existing media types, promotes interoperability. The second option, defining new media types, can be particularly useful to meet domain-specific requirements, but the new media types have to be adopted to achieve interoperability. For instance, in WoT applications it can be useful to define concise representation formats for lightweight processing on constrained devices. Lastly, using standard RDF serialization formats might be preferable for general purpose applications, but this approach assumes that clients understand the ontologies used to describe the states of digital artifacts.

Hypermedia-driven interaction To support hypermedia-driven interaction, an STN platform must expose a *hypermedia-driven API*. There are multiple solutions already available for this purpose. The STN ontology provides two modules that developers can use to expose hypermedia-driven APIs for STN platforms (see [2] for more details). Other similar solutions include Hydra [27] and the Thing Description¹⁴.

STNs provide several elements to enhance hypermedia-driven interaction both *within* and *across* STN platforms (see [3] for a detailed discussion):

- the *social network metaphor* enables the discovery of agents;
- the `stn:holds`¹⁵ relation, defined in the STN Ontology, enables the discovery of *user accounts* held by *agents* in various STNs;
- the `stn:hostedBy` relation enables the discovery of STN platforms and any hypermedia-driven APIs they may expose.

However, most existing Web services, such as Facebook and Twitter, expose non-hypermedia APIs. Nevertheless, both Facebook and Twitter can bring significant value to STN-based agent environments, as we show in Section 4. To support our approach, we thus have to provide solutions to integrate non-hypermedia APIs into the hypermedia-driven STN-based layer. We addressed this problem in [3].

To conclude, in this section we introduced a resource-oriented abstraction layer for agent environments. This layer is based on STNs, which are used to

¹³ A list with all registered media types is available at IANA: <http://www.iana.org/assignments/media-types/media-types.xhtml>, last accessed: 08.02.2017.

¹⁴ <http://w3c.github.io/wot/current-practices/wot-practices.html>, last accessed: 08.02.2017.

¹⁵ We use `stn:` as a prefix bound to the URI: <http://w3id.org/stn/core#>.

bring *humans in the loop* and to tackle the HATEOAS constraint. By fully conforming to REST, the STN-based abstraction layer inherits architectural properties such as *Internet-scalability* and *openness*. Therefore, the proposed abstraction layer addresses in an integrated manner all three research challenges in the engineering of agents environments that motivate our work.

4 Implementation and Evaluation

To demonstrate our approach, we implemented an STN-based agent environment for the IoT that is deployed over multiple *heterogeneous* Web services and constrained devices. Software agents running in *independent* application environments are able to seamlessly navigate and act on the underlying distributed STN in order to discover and interact with one another and with human agents.

This section is structured as follows. First, we present the application scenario in Section 4.1. Then, in Section 4.2 we present the implemented system. Finally, in Section 4.3 we discuss the workings and limitations of this proof of concept.

4.1 Application Scenario

David owns multiple smart things, such as a wristband, a mattress cover, light bulbs and curtains.¹⁶ These things are able to produce, share and consume contextual information about David and his bedroom. For instance, when David falls asleep, both the wristband and the mattress cover can produce this information with various certainty levels and share it with the other things. David also uses a digital calendar service to keep track of important events. If there is an upcoming event scheduled and the calendar *knows* that David is still asleep, it cooperates with other things discovered at runtime in attempts to wake him up (e.g., via vibration alarms on his wristband, opening the curtains to allow natural light to enter the room). David’s things are also context-aware: it makes little sense to open the curtains if the outside light level is below 100 lux (S.I.), which is the equivalent of an overcast day¹⁷. If all attempts fail and the scheduled event has high priority (e.g., a morning flight), the calendar crawls David’s online distributed social graph in order to discover and contact friends that can wake him up. To avoid unnecessary contact attempts, the calendar must first discover which of David’s friends are already awake.

In this scenario, human and software agents work together towards a common goal: to wake up David. The calendar is able to discover and interact with other agents in an *open system*. In doing so, the calendar has to navigate and use an agent environment deployed over multiple *heterogeneous platforms* (e.g., Facebook, Twitter) and constrained devices (see Figure 2), a requirement that is essential in *Internet-scale systems*. If all software agents’ attempts to wake

¹⁶ All connected objects used in our application scenario resemble products already available to end-users.

¹⁷ http://www.engineeringtoolbox.com/light-level-rooms-d_708.html, last accessed: 08.02.2017.

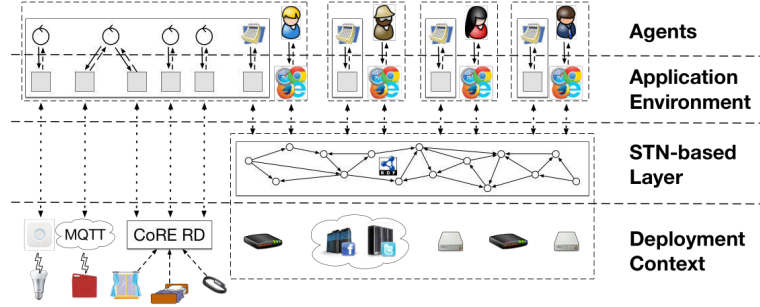


Fig. 2. Experiment setup: an STN-based agent environment deployed over multiple *heterogeneous* platforms and constrained devices.

up David fail, the calendar delegates this goal to human agents, who are thus *brought into the loop* at runtime and become part of the system. Therefore, this application scenario emphasizes all three research topics that motivate our work.

4.2 System Overview

To evaluate the above scenario, we implemented the system in Figure 2. Multiple *application environments* developed using the *JaCaMo platform* [7] are “glued” together via an STN distributed across Facebook, Twitter, and multiple instances of our own implementation of an *STN platform*. Software agents running in these application environments use the STN platforms to communicate with one another, and use Twitter to communicate with humans. We modeled each of David’s things and the calendars of each of David’s friends as BDI agents developed in *Jason* [8].

In what follows, we first present in further detail the *deployment context*, then the *STN-based layer*, and finally the *application environments* (see Figure 2).

Deployment Context The deployment context is *distributed* across multiple *heterogeneous* platforms and constrained devices.

We used Facebook and Twitter to deploy David’s online social graphs. These platforms expose non-hypermedia APIs. The clients used in our implementation are preconfigured with the authorization tokens required to access these APIs.

To host STNs for David, each of his friends, and their things, we used multiple instances of an *STN platform*. These platforms are used to store and manage relations and as message brokers. The STN platforms expose hypermedia-driven APIs (see [4] for more technical details).

We implemented David’s lights using a Philips Hue lightbulb. The lightbulb communicates with a bridge via ZigBee Light Link, and the bridge exposes a non-hypermedia HTTP API for controlling the lightbulb.¹⁸

¹⁸ For more details: <http://www.developers.meethue.com/documentation/how-hue-works>, last accessed: 08.02.2017.

We implemented the outside light sensor using a Texas Instruments (TI) SensorTag. The TI SensorTag communicates with a smartphone via Bluetooth Low Energy, which in turn pushes the sensor readings to a cloud-based MQTT broker¹⁹. MQTT clients can then subscribe to this broker to receive real time sensor readings (see details in Section 4.2).

To implement the wristband, mattress cover, and window curtains, we emulated CoAP devices using the Californium (Cf) framework [26]. Following the typical setup used in *constrained RESTful environments (CoRE)*, these devices register their resources with a CoRE Resource Directory²⁰, which allows them to be discovered dynamically at runtime. We used Cf-RD²¹ to deploy the resource directory.

STN-based Layer In our experiment setup, David has an `stn:connectedTo` social relation to a friend on his STN platform, two other friends on Facebook, and follows two users on Twitter. We used Facebook’s test harness and regular Twitter accounts to set up David’s social graphs on these platforms.

The description of David’s user account on his STN platform advertises the user accounts he `stn:holds` on Facebook and Twitter, which makes them discoverable and allows agents to *navigate into* these networks. To enable agents to *navigate out* of Facebook and Twitter, David’s friends advertise their URIs as their *personal websites*, an attribute field available on most social platforms.

Once all data is lifted to RDF (see [3] for technical details), software agents have a uniform RDF-based view of the distributed STN and are *decoupled* from its underlying *heterogeneous platforms* (see Figure 2).

Application Environments The application environments programmed for this scenario provide agents with *CARtAgO artifacts* [34] that they can use to manipulate STNs and to interact with devices.

The *STN artifacts* are environment abstractions that encapsulate parsers for interpreting RDF descriptions of STN platforms, and HTTP clients used to issue requests constructed *at runtime* based on these descriptions. The platform descriptions are *discovered at runtime* via the `stn:hostedBy` relation and provide all the knowledge required to interface with the platforms, which includes metadata about the platform’s API (e.g., supported authentication protocols and media types) and descriptions of supported STN operations.

The *device artifacts* provide agents with wrappers for devices in their deployment context. The Philips Hue artifact is a wrapper over an HTTP client that accesses the Philips Hue bridge in the local network, whereas the TI SensorTag artifact is a wrapper over an MQTT client that communicates with an MQTT broker in the cloud. Similarly, the artifacts for CoAP devices are wrappers over CoAP clients that access the emulated devices. These artifacts are

¹⁹ In our implementation, we used HiveMQ: <http://www.hivemq.com/>, last accessed: 08.02.2017.

²⁰ <https://tools.ietf.org/html/draft-ietf-core-resource-directory-09>

²¹ <https://github.com/eclipse/californium.tools>, last accessed: 08.02.2017.

created dynamically at runtime based on the *resource types* discovered via the CoRE resource directory.

4.3 Proof of Concept

The application scenario runs in multiple stages. First, David’s agents get bootstrapped into the STN layer via preprogrammed behavior. Their *entry point* is David’s URI, which they use to declare David as their owner by creating `stn:ownedBy` relations. In doing so, the agents are acting on the distributed STN in an *autonomous* and *reliable* manner in order to *make themselves discoverable*. Then, they can crawl the STN to discover and subscribe to all other agents `stn:ownedBy` David.

Once interconnected, agents interact using the FIPA Contract Net Protocol [17]: the *calendar* agent publishes a call for proposals to wake up David via his STN platform, and the *wristband*, *lights* and *window curtains* agents reply with proposals. The agents use a shared communication language (in our implementation, the one provided by Jason) and a shared vocabulary for describing the state of the environment. The window curtains agent decides to join the interaction based on the readings of the outside light sensor. All wake-up attempts eventually fail in our implementation.

Next, the calendar agent crawls David’s online distributed social graph to discover friends that also `stn:owns` calendar agents, where the *entry point* in the distributed STN is once again David’s URI. We described this crawling process in detail for a similar scenario in [3]. Once David’s calendar discovers the other calendar agents in our system (see Figure 2), it asks each of them if their owner is asleep. For each friend that is awake, the calendar then searches for an `stn:UserAccount` they `stn:hold` on an `stn:Platform` that implements the `stn-ops:SendMessage` operation. The only friend who satisfies this criteria is Mike, who is awake and holds an account on Twitter, which implements the required operation. The calendar sends Mike a direct message on Twitter to wake up David.

This proof of concept application demonstrates that we can successfully apply our approach to engineer agent environments that address the three research topics that motivate our work in an integrated manner. Note that the agent environment used in this application integrates two of the most used online social platforms in a seamless manner.

The *STN artifacts* are decoupled from the platforms that host the distributed STN in our experiment. The *device artifacts* are still tightly coupled to the deployment context (see Figure 2), but they are hidden behind agents. However, these artifacts can also be decoupled using the same approach. First, they have to be linked in the distributed STN to become discoverable, and then they have to translate the higher-level semantics of operations defined by a domain-specific model (e.g., a general model for light bulbs) to the lower-level semantics of a WoT protocol used by the devices’ APIs.

5 Conclusions

The World Wide Web is arguably the most scalable and versatile software system deployed on the Internet [13], and most other software systems today revolve around the Web. This huge success comes from the architectural properties of Web, which was specifically designed as an Internet-scale and long-lived system in which components can be deployed and can evolve independently from one another. Precisely because of its design goals, key architectural principles and ubiquity in people's lives, in this paper we claimed that the Web provides a suitable middleware for agent environments that (i) support large scale systems, (ii) can cope with open systems, and (iii) support humans in the loop, three important and current research topics in the engineering of agent environments [38].

Our approach is to apply the REST architectural style to design a resource-oriented abstraction layer for agent environments that decouples the *application environment* from its *deployment context*. Higher-level environment abstractions can then be implemented on top of this lower-level abstraction layer. The novelty of our approach is the use of *socio-technical networks (STNs)* as a means to address the HATEOAS constraint, which is one of the core tenets of the modern Web architecture. By analogy with how the Web enables the discovery of Web pages, STNs enable the discovery of agents in agent environments on the Web.

We consider the impact of the contribution presented in this paper to be twofold. On the one hand, it addresses important research topics in engineering MAS. On the other hand, it enables the transfer of MAS technology to the development of Web-based systems in a manner that is completely aligned with the Web architecture. We consider the latter to be an important step towards achieving the vision of a *Semantic Web*, as originally described in [6].

References

1. Abdullah Althagafi. Designing a Framework for RESTful Multi-Agent Systems. Master's thesis, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada, 2012.
2. Andrei Ciortea, Antoine Zimmerman, Olivier Boissier, Adina Magda Florea. Towards a Social and Ubiquitous Web: A Model for Socio-technical Networks. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 461–468. IEEE, 2015.
3. Andrei Ciortea, Antoine Zimmerman, Olivier Boissier, Adina Magda Florea. Hypermedia-driven Socio-technical Networks for Goal-driven Discovery in the Web of Things. In *Proceedings of the 7th International Workshop on the Web of Things (WoT)*. ACM, 2016.
4. Andrei Ciortea, Olivier Boissier, Antoine Zimmerman, Adina Magda Florea. Responsive Decentralized Composition of Service Mashups for the Internet of Things. In *Proceedings of the 6th International Conference on the Internet of Things (IoT)*. ACM, 2016.
5. Estefania Argente, Vicente Botti, Carlos Carrascosa, Adriana Giret, Vicente Julian, and Miguel Rebollo. An abstract architecture for virtual organizations: The thomas approach. *Knowledge and Information Systems*, 29(2):379–403, 2011.

6. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.
7. Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.
8. Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007.
9. Lars Braubach and Alexander Pokahr. Conceptual Integration of Agents with WSDL and RESTful Web Services. In *International Workshop on Programming Multi-Agent Systems*, pages 17–34. Springer, 2012.
10. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation 25 February 2014. W3C Recommendation, World Wide Web Consortium (W3C), February 25 2014.
11. Oguz Dikenelli. SEAGENT MAS platform development environment. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pages 1671–1672. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
12. Jose Exposito, Joan Ametller, and Sergi Robles. Configuring the JADE HTTP MTP. <http://jade.tilab.com/documentation/tutorials-guides/configuring-the-jade-http-mtp/>, 2010. Accessed: 15.11.2016.
13. Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.
14. Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
15. Foundation for Intelligent Physical Agents. FIPA Abstract Architecture Specification. <http://www.fipa.org/specs/fipa00001/SC00001L.html>, 2002. Document number: SC00001L.
16. Foundation for Intelligent Physical Agents. FIPA Agent Message Transport Protocol for HTTP Specification. <http://www.fipa.org/specs/fipa00084/SC00084F.html>, 2002. Document number: SC00084F.
17. Foundation for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification. <http://www.fipa.org/specs/fipa00029/SC00029H.html>, 2002. Document number: SC00029H.
18. Nicholas Gibbins, Stephen Harris, and Nigel Shadbolt. Agent-based semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):141–154, 2004.
19. Abdelkader Gouaïch and Michael Bergeret. Rest-a: An agent virtual machine based on rest framework. In *Advances in Practical Applications of Agents and Multiagent Systems*, pages 103–112. Springer, 2010.
20. Dominic Greenwood and Monique Calisti. Engineering web service-agent integration. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, pages 1918–1925. IEEE, 2004.
21. Dominic Greenwood, Margaret Lyell, Ashok Mallya, and Hiroki Suguri. The ieee fipa approach to integrating software agents and web services. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 276. ACM, 2007.
22. Miguel Escrivá Gregori, Javier Palanca Cámara, and Gustavo Aranda Bada. A jabber-based multi-agent system platform. In *Proceedings of the fifth international*

- joint conference on Autonomous agents and multiagent systems*, pages 1282–1284. ACM, 2006.
23. Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.
 24. Michael N Huhns. Agents as web services. *IEEE Internet computing*, 6(4):93, 2002.
 25. Michael N Huhns and Munindar P Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet computing*, 9(1):75–81, 2005.
 26. Matthias Kovatsch, Martin Lanter, and Zach Shelby. Californium: Scalable cloud services for the internet of things with coap. In *Internet of Things (IOT), 2014 International Conference on the*, pages 1–6. IEEE, 2014.
 27. Markus Lanthaler and Christian Gütl. Hydra: A vocabulary for hypermedia-driven web apis. *Proceedings of the 6th Workshop on Linked Data on the Web*, 996, 2013.
 28. Dejan Mitrović, Mirjana Ivanović, Zoran Budimac, and Milan Vidaković. Radigost: Interoperable web-based multi-agent platform. *Journal of Systems and Software*, 90:167–178, 2014.
 29. Xuan Thang Nguyen and Ryszard Kowalczyk. Ws2jade: Integrating web service with jade agents. In *International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 147–159. Springer, 2007.
 30. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.
 31. Benno J Overeinder and Frances MT Brazier. Scalable middleware environment for agent-based internet applications. In *International Workshop on Applied Parallel Computing*, pages 675–679. Springer, 2004.
 32. Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. “big” web services: Making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, pages 805–814. ACM, 2008.
 33. Alessandro Ricci, Enrico Denti, and Michele Piunti. A platform for developing soa/ws applications as open and heterogeneous multi-agent systems. *Multiagent and Grid Systems*, 6(2):105–132, 2010.
 34. Alessandro Ricci, Mirko Viroli, and Andrea Omicini. CArtAgO: A framework for prototyping artifact-based environments in MAS. In *Environments for Multi-Agent Systems III*, pages 67–86. Springer, 2007.
 35. Michael Rovatsos, Dimitrios Diochnos, and Matei Craciun. Agent protocols for social computation. In *Advances in Social Computing and Multiagent Systems*, pages 94–111. Springer, 2015.
 36. M Omair Shafiq, Ying Ding, and Dieter Fensel. Bridging multi agent systems and web services: towards interoperability between software agents and semantic web services. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC’06)*, pages 85–96. IEEE, 2006.
 37. Munindar P Singh and Michael N Huhns. *Service-oriented computing: semantics, processes, agents*. John Wiley & Sons, 2006.
 38. Danny Weyns, Fabien Michel, et al. Agent environments for multi-agent systems – a research roadmap. In *Lecture Notes in Computer Science*, volume 9068, pages 3–21. Springer, 2015.
 39. Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.
 40. Erik Wilde. Putting things to rest. *School of Information, UC Berkeley*, 2007.

Modeling for openness in MAS

Daniela Maria Uez and Jomi Fred Hübner

Department of Automation and Systems Engineering
Federal University of Santa Catarina
CP 476 - CEP 88040-900 - Florianópolis - SC - Brazil
{daniela.uez@posgrad.ufsc.br, jomi.hubner@ufsc.br}

Abstract. An open MAS allows any of its dimensions (e.g. agents, environment, interaction, organization) being changed at runtime, with the inclusion or exclusion of components that are unknown at design time. Openness makes the design of systems a complex task since one or more dimensions are unknown and cannot be specified. Despite the existence of many methods to design MAS, for different reasons they do not deal properly with openness. In this paper, we introduce a method to design open MAS that aims to help the design team to deal with openness issues. In our proposed method, dimensions are specified independently, using their own concepts and abstractions, without relying on other, and a set of linking concepts that is used to allow these dimensions to work together. Linking concepts are grouped into two categories: related concepts that represent the same abstraction from two points of views, and border concepts that provide, at design time, information about how dimensions are linked at runtime. It is important to notice that since our work is still in progress, this is a preliminary set of concepts that may be changed along next steps.

Keywords: Multiagent Systems, Openness, Method, MAS Dimensions, Linking Concepts, Related Concepts, Border Concepts

1 Introduction

A MAS may be based on four main dimensions, each one concerned with a specific system aspect [1,11]: agents, environment, organization and interaction. For each MAS a different arrangement of these dimensions can be done, according to the systems requirements. For example, a scholar system may be composed of agents, representing professors and students; organization, defining norms that specify the desired behavior of professors and students; and a set of interactions protocols that allows the communication between students and professors. The environment dimension may not be specified since in that case, its artifacts are not crucial for the system. On the other hand, a system that simulates the car traffic in a town may be composed only of the agent dimension, representing cars and walkers, and environment dimension, representing road signals, without including organizational and interaction dimensions that are not relevant to the simulation.

A MAS is commonly called open when the agent dimension can be changed at runtime, with agents entering or leaving the system freely [2,8,12,18]. It means the agents that will take part in the system at runtime may not be known at design time. For example, in the scholar system, we can easily define roles, norms, and protocols of interactions. However, the agents can be unspecified in the case where each student and professor may develop its own agent.

Such definition of openness is focused on the possibility of agent dimension be changed at runtime and do not take account of the openness in other dimensions. Despite that, environment, organization and interaction dimensions may also be changed at runtime. It means that in a specific MAS new artifacts may be created in the environment, norms and roles may be included in the organization, and the set of protocols of interactions can be increased at runtime. Thus, in this paper, we consider an open MAS as the one that allows any dimension being changed at runtime. Components of any dimension may be included even if it is not previously known, as well existent components may be excluded while the system is running.

Hence, as detailed in section 2, openness implies some issues that make the design of this kind of MAS a complex task and current methods do not deal properly with them. We can highlight two reasons for this inadequacy: the interdependence of dimensions and the lack of concepts to link them at runtime. In order to deal with these issues, in section 3 we describe a preliminary method to analysis and design open MAS. Since the methodological aspects of our proposal are not the focus of this paper, we present only some few characteristics, mainly the concepts used on the metamodel that are important in order to understand the link concepts. The main characteristic of our method is to deal with each dimension independently, using their own concepts and abstractions, without relying on others. In this manner, each dimension has its own metamodel that is linked with the others by a set of specific concepts, discussed in section 4. Next, in section 5, we present an example that briefly describes the method focusing on the specification of the border concepts. Finally, in the section 6, we present some considerations about our proposal and some future works to improve our proposal.

2 Methods for openness

Openness implies some issues that affect the system since the design phase, as the MAS may be designed and developed even when we do not know all its components. At runtime, these components, unknown at design time, will take part in the MAS, working together with the existent ones. Designing openness is complex since it cannot be specified in one single component of MAS. In fact, openness just arises at runtime, when new components are being included in the system, thus it is not easy to determine what is necessary for a method properly deal with openness.

Methods for analysis and design of MAS present different approaches in order to deal with openness. Some of them [4,10,20] are more concerned with agent

dimension. Usually, in these cases, some concepts typically used to specify organization or environment, are used mainly in the early phases of the method, to clarify the problem to be solved, and along the method, they are replaced by agent's concepts. For instance, Prometheus Methodology [20] uses the organizational concept of role to specify agent's capabilities, and through the method, these roles give rise to the agents. In that way, at the end of these methods, only agent concepts are coded and all information about other dimensions is included at agent's internals. Openness issues at runtime have to be handled by the agents of the system. One reason for it may be the fact that the most used development platforms, like JADE [3], JADEX [6] and Jason [5] do not provide first class entities to deal independently with organization, environment or interaction dimensions, and their concepts cannot be implemented. Nevertheless, these methods are not exactly suited for openness since the system cannot be designed without the agent dimension. Moreover, all information about environment, organization, and interaction may be included in the agent, then we have to know these dimensions at design time. It does not prevent the design of open MAS, however, it constraints the specification of openness specific issues and do not offer guidelines to help the integration of new agents.

Other approaches [13,14,15,16] focus on organizational concepts to allow the design of MAS, mainly the ones with openness at agent dimension. In these works, MAS can be designed without any information about agent dimension, and the organization dimension is used to regulate the agents that will take part in the system at runtime. However, some of them just deal with concepts of organizational dimension and they do not offer specific concepts to deal with openness in environment, organization or interaction dimensions.

Current methods are concerning, mainly, with openness at agent dimension, each one using a specific approach to deal with openness issues. Broadly speaking, if a MAS is open at agent dimension, at design time we do not know anything about agent's mental states, capabilities or plans. However, the other dimensions should be designed and developed in a way that, at runtime, these agents may enter in the system and work with them. That is, the agent may be able to take part in a protocol, play a role in the organization, use environmental artifacts and so on.

In this manner, a method to better deal with openness may allow each dimension being specified independently, with its own concepts, without relying on other dimension and provides specific concepts that help the integration of new components.

3 Our proposal

We are proposing a method for analysis, design, and development of open MAS in which dimensions are designed independently in order to help the team to deal of openness specific issues. Besides its focus on openness, the method also allows the design of ordinary systems, in which all components are known at design time. An overview of the method is presented in figure 1.

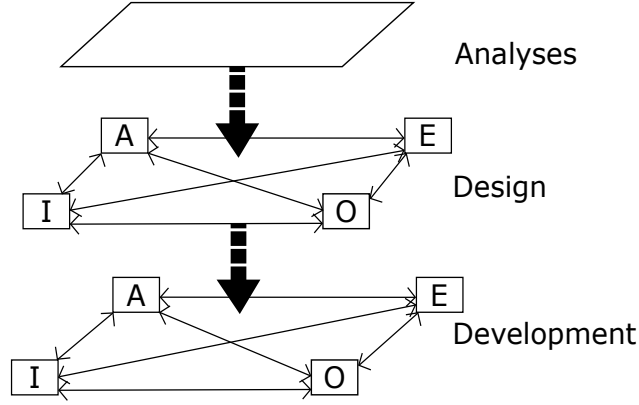


Fig. 1: Method overview

The analysis phase deals with general concepts used to clarify the system requirements. It starts with an initial system specification and aims the definition of a detailed description of the systems functionalities. This detailed description helps the design team to define which dimensions can be designed for this system since the next phase requires only those dimensions that are known at design time.

In the design phase, the goal is to specify the elements composing the system, like agents, roles, norms, protocols and artifacts. This phase starts with the detailed system description defined early and deals with dimensions specific abstractions. Only those dimensions defined at the end of analysis phase are designed, using its own specific set of abstractions, without relying on other dimension. Later in this phase, some abstractions used to link dimensions are also defined. In this manner, at the end of the design phase, we have a set of elements that are part of the system, including dimensional and linking abstractions, as well their relationships, without platform specific details.

In the last phase, those abstractions defined at the design phase are refined to include specific information regarding the target platform. All dimensions and linking abstractions are fitted to a specific platform of development, complementing them with those abstractions used by the platform. It allows the initial system code to be generated automatically. Although any development platform may be used in this phase, we adopt the JaCaMo framework [1] since it allows the development of each dimension independently.

Each of these phases uses a specific set of concepts and the metamodels changes throughout them. In this paper, we focus on the metamodel used in the design phase since it deals with computational concepts without any specific target platform. Moreover, in order to allow the design of open systems, our metamodel deals with concepts of each dimension independently. In other words, each dimension has its own metamodel and they are linked by some concepts that are used to connect them. In this way, the open system may be designed removing

the unknown dimension without affecting the others. The concepts used in each dimension are not new since the focus of our proposal is the concepts used to link them. Along this section, we briefly discuss the set of concepts used to define each dimension since they are relevant in the definition of linking concepts, presented in next section.

3.1 Agent Dimension

The concepts of the agent metamodel presented in figure 2 are defined based on the BDI architecture, largely used by MAS community [4,7,20,10,5,1] to implement deliberative agents. The main concepts used in this metamodel is the *agent* that represents an *autonomous* entity. An agent has a set of *beliefs* that represent things it knows about its world; a set of *goals* that represent things it wants to achieve; and a set of *plans* that represents its know how to achieve its goals. Plans are composed of an ordered set of *actions* that are performed by the agent. Agents also receive some *perceptions* from the environment that will update its beliefs or trigger new goals.

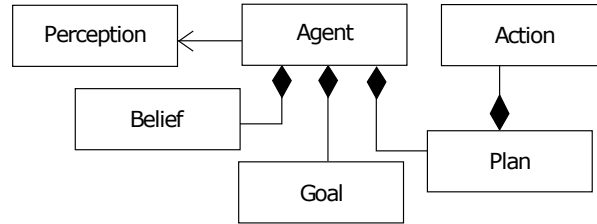


Fig. 2: Agent dimension metamodel

3.2 Environment Dimension

As proposed in Agent and Artifacts (A&A) framework [19], the environment is composed of a set of entities encapsulating functions and services for the agents, possibly connecting them to the real environment of the application. The set of concepts used to specify the environment dimension is presented in figure 3. The main concept in the environment metamodel is the *artifact* that represents resources or tools that can be used and shared by agents. A resource is a finite set of supply, like battery charge, while tools include sets of functionalities, like a calculator. Artifacts provide a set of *operations* that represent computational process executed inside the artifact, and may produce some *events*. Moreover, artifacts are grouped into one or more containers called *workspaces*.

3.3 Interaction Dimension

The metamodel for the interaction, based on [25], is presented in figure 4. In this metamodel, the main concept is the *protocol* that specifies a sequence of

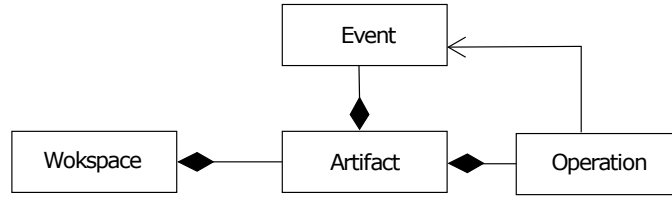


Fig. 3: Environment metamodel

interactions among participants. The protocol is composed of a set of *states*, linked by transitions that specify how the protocol evolves from one state to another. A *transition* may be triggered by a message sent by a participant or the event of some change in an observable property.

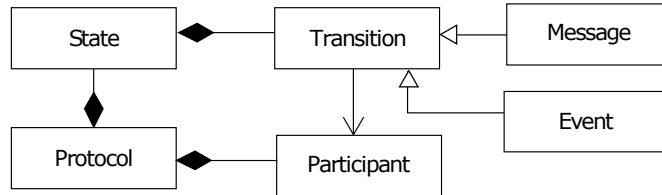


Fig. 4: Interaction metamodel

3.4 Organization Dimension

The organization metamodel, presented in figure 5, is based on the concepts defined by some organizational models, like Moise [17] and Opera [13]. In this metamodel, a social scheme is defined. The social scheme is composed of a set of *goals* that represents things the system wants to achieve. Goals can be decomposed into subgoals. Organization also has *roles* that encapsulate the expected behaviour for an agent when it takes part in the organization. Related roles can be organized in a *group* of roles, in order to define specific links among them. The agents playing roles have to satisfy the *norms* that are used to regulate the achievement of goals. Moreover, a *protocol* can be specified to define the precise actions to achieve a goal.

4 Linking Concepts

Linking concepts are those abstractions that are used as an interface between one dimension and other, allowing them to work together. For instance, when an agent plays a role in the organization, a connection between agent and organization dimensions is established at runtime, even when no special information about why and how it occurs is provided at design time.

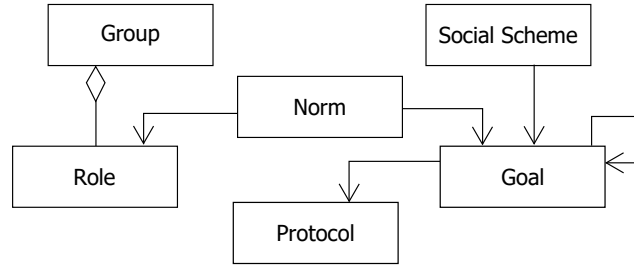


Fig. 5: Organizational metamodel

In regular systems, when all dimensions are known at design time, agents may be designed to play a specific role or the role specified to be played by an agent in particular. On the other hand, in case of openness at the agent dimension, the agents are not developed specifically for this system and they are usually unknown at design time. Providing more information about the available roles, and the necessary skills for being correctly played may help agents to choose the appropriate role to play when entering the system.

In our work, linking concepts are used to assist these elements that are unknown at design time to be better integrated at runtime. Consequently, linking concepts are used mostly in the case of openness since they provide, at design time, details that are used, at runtime, by open dimensions to become familiar with those elements already running in the system. They are grouped into two categories, discussed below: related concepts and border concepts.

4.1 Related Concepts

Related concepts are those concepts in one dimension that have a similar meaning in concepts of other dimension. In other words, a related concept presents one same abstraction in a different point of views. For example, the perception concept, defined at agent dimension, is related with the event concept, defined at environment dimension. That means that an event generated by the environment is considered as a perception by the agent. For instance, the agent perception *whether(raining)* corresponds to the event *whether(raining)* produced by a kind of forecast artifact. These concepts are commonly used by MAS community and each dimension has their own set of abstractions to link them with other dimensions. Our proposal for those links is presented in figure 6.

As mentioned before, agent perception is related with the environment event concept since events are perceived for agents. Similarly, action in the agent dimension is related with the environment operation concept since an agent may perform an action at environment by executing an artifact operation, and also is related with the interaction message concept since an agent may execute an action to send a message. The interaction concept of participant is related with the concepts of agent, from agent dimension, artifact, from environment dimension, and role, from organization dimension since any of them, agents, roles, and

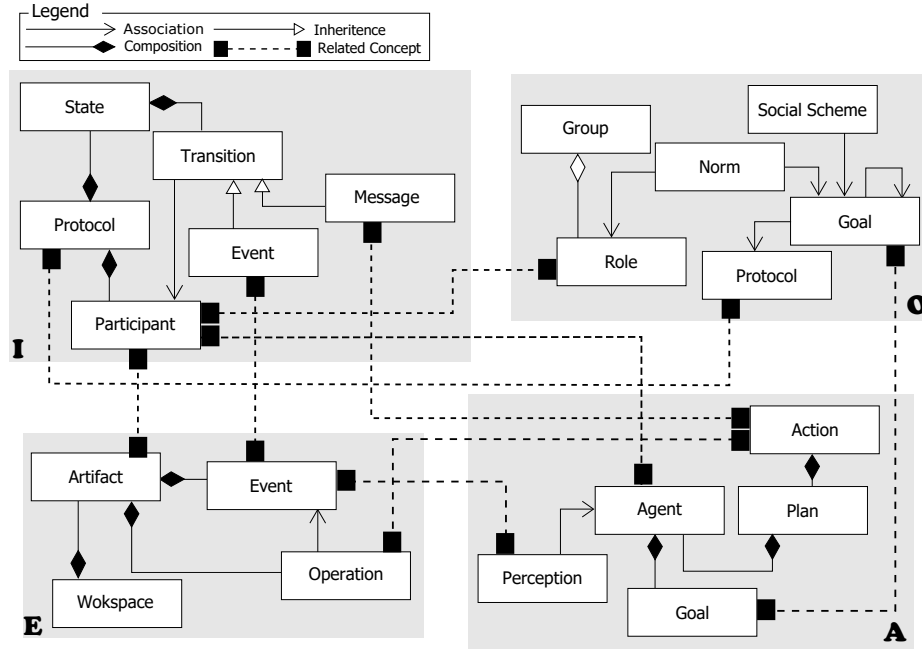


Fig. 6: Related Concepts

artifacts, may participate in a protocol of interaction. Also, the event concept at interaction dimension is related with the event concept at environment dimension since an event that occurs in the environment may be a transition in an interaction protocol. Similarly, the interaction protocol concept is related with the organization protocol concept since a protocol may describe how an organizational goal may be achieved. Finally, the organization goal concept is related with the agent goal concept since those goals defined in the organization can be assigned to the agents.

Related concepts link dimensions like when an environmental event occurs and is perceived by the agent. However, in the case of openness, some additional information about these concepts may be necessary, in order to allow the full integration of new elements. That is why we defined a set of special concepts called border concepts, discussed below.

4.2 Border Concepts

Border concepts are special concepts used to provide, at design time, specific information about how dimensions are linked at runtime. These concepts do not belong to a specific dimension and are used mainly in case of openness, to help open elements to become familiar with those existent ones and be integrated with them. For instance, an artifact manual is a border concept since it is used by the agents to know artifacts functionalities. In that way, in a system with

openness at agent dimension, an incoming agent entering the system can read this manual and properly works with the environmental artifacts. Border concepts are presented in figure 7, as well as their connections with other concepts. Basically, four border concepts are included in our metamodel: manual, skill, brute fact and status function.

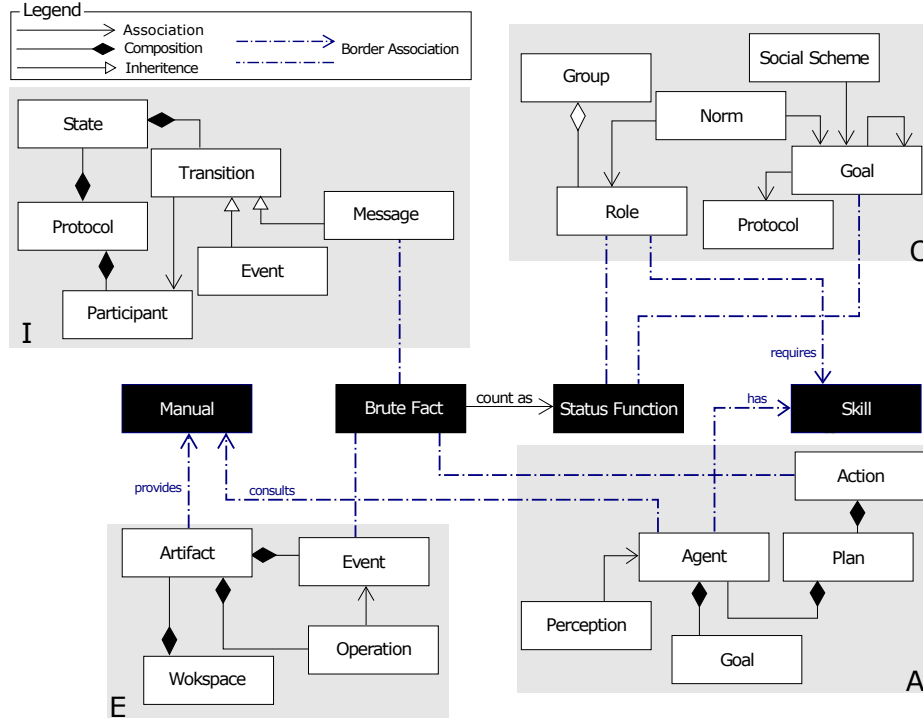


Fig. 7: Border Concepts

The manual concept was defined based on the A&A framework [19,22] and is used to make agents familiar with the environment artifacts. In this fashion, a manual is used as a way to present to agents the environment, their functionalities, and events. Agents can read the manual, in the same way that a human can read the calculator's manual to learn how to use it.

The skill concept is used to establish the required competencies and capabilities for an agent to correctly play an organizational role. In this way, each role may define its own set of required skills and agents that do not present one or more of them are not able to play the role. The skill concept was defined based on the work of [24] and allows new agents to be familiar with roles before committing to them.

While the manual concept is used to allow new elements to properly interact with the environment, and skill is used to constrain which agents may take part

in an organization, brute fact and status function concepts are used together to define how facts related with agents, environmental events or messages may affect the organization. A brute fact can be a message, an event or an agent, and status functions are their organizational counterparts, representing the organizational interpretation of brute facts [23]. A common status function is to play a role or an organizational goal to be achieved. For example, in an auction system, to send a bid message (a brute fact) count as the agent playing the role of the bidder (a status function) in the auction. As presented in the figure, brute fact and status function concepts are linked by a count as relationship that means a brute fact become a status function in organization. These concepts are defined based on the work of [9].

5 Example

We illustrate our metamodel, especially the linking concepts, with the traveler assistant example. The assistant aims to help the tourist during visitation in a specific tourism region. The assistant provides information about tourist-related spots and services, indicating places to visit, eat or relax according to the tourist profile, budget, time availability, needs, and preferences. The tourist may negotiate prices and dates with services providers (restaurants, hotels, tour guides or tour operators) and access relevant information about the visiting places (parks, historical places, monuments), like history and main characteristics, as well as information about available services next to the attraction. The main assistant functionalities are presented below:

1. *Planning*: the tourist may define which places it intends to visit based on its schedule and profile and the information about tourism region. These places are marked on the map.
2. *Traveling*: the tourist may consult the route to move until next visiting place defined on the map. The map may present some relevant places existing along the route.
3. *Visiting*: next to a place, the tourist may consult relevant information about it, like the history, main characteristics, and services in the neighborhood. Moreover, a list of related places to visit, with same history or subject, is presented as a suggestion.
4. *Services*: service providers may contact the tourist in order to offer services, as well as, a tourist may find a provider using the map.
5. *Negotiation*: a negotiation between tourist and providers to define prices and dates may be done before to schedule a service. The tourist cannot schedule a new appointment at the same date and hour of another appointment already scheduled, save in case of hotel reservation.

In the analysis phase, the development team creates a detailed description of the system that is used in the design phase as the starting point for the definition of the elements that compose the system. In this example, we present only the

diagrams used during the design phase. The diagram notation is inspired by the work of [21].

Based on the detailed system description, we notice that the system is *open at agent dimension* since each tourist and service provider may develop its own agent. In this case, organization, environment, and interaction dimensions should be designed in any order since dimensions are independent.

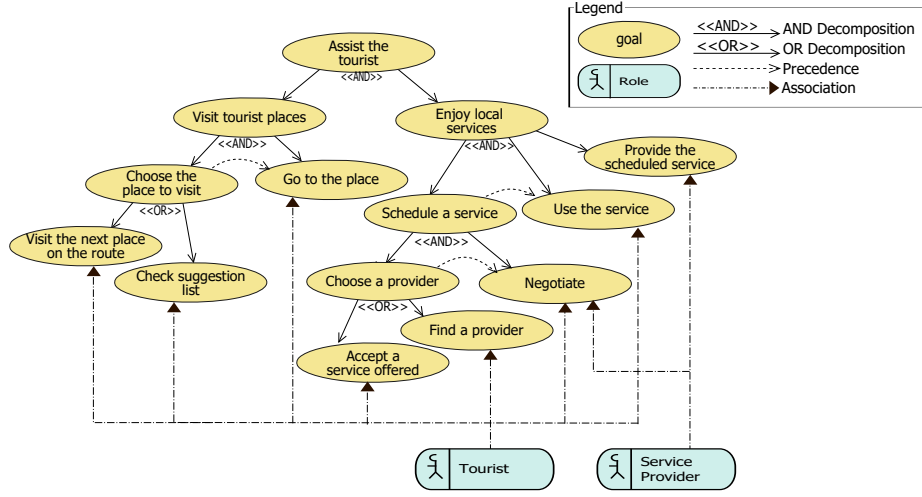


Fig. 8: Goal tree diagram with the related roles

Starting by the organization dimension, we define an initial set of system goals that are structured in an AND/OR tree. This initial set must be refined in order to complete the goal tree. In the example, the main system goal is to *assist the tourist* during the visit in a tourist region, by *visiting tourist places* and *enjoying local services*. To visit tourist places, it should first *choose the place to visit*, by *visiting the next place on the route* or *checking the suggestions list* presented in the actual place, and then *go to the place*. To enjoy local services, the tourist should first *schedule a service*, then it can *use the service* while the service provider should *provide the scheduled service*. To schedule a service, the tourist should *choose a provider*, by *accepting a service offered* or *finding a provider* in the map, and then tourist and providers may *negotiate* prices and dates to providing the service.

In defining the goal tree, we notice that two roles should be specified: *tourist* and *service provider*. Roles are commonly used in open systems, mainly in case of openness at agent dimension, since it encapsulates the expected behavior for the incoming agent. Part of the MAS can be designed based on roles, without considering the specification of the agents. To be integrated in the system at runtime, each agent must play at least one of the defined roles. For the traveler assistance, the *tourist* role represents agents that are visiting the tourist region,

while the *service provider* role represents agents that provide services to the tourist. Each role is related to a set of goals that must be achieved by the agent playing this role. In the example, the agent playing the role tourist has to achieve the goals *go to the place* and *use the service*, among others, and the agent playing the role service provider has to achieve the goal *provide the service*. Figure 8 presents the goal tree as well as the roles associated with each goal.

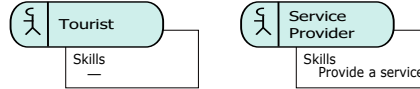


Fig. 9: Required skills for Roles

In many cases, a role requires some specific skills to be properly played, like the *service provider* role that must be played by an agent able to provide a service. If some agent does not have the specified skills, it may not achieve one or more goals and the system may not work properly. Since the agents probably do not know in advance which activities are related to the role, it is important to specify the set of required skills to play each role, allowing the agent to decide whether to play a role or not. Figure 9 presents skills related to the roles defined for the traveler agent example. In this example, the traveler role does not require any skill to be played.

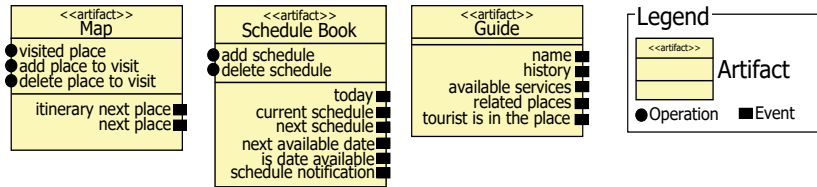


Fig. 10: Artifacts

In the example, each tourism region provides a *guide* artifact that provides useful information for new agents interest in the region. An agent playing the role of tourist has its own *map* artifact in which the places to visit are marked, and a *schedule book* artifact used to schedule the services. Likewise, agents playing the services providers roles have their own *schedule book*. Each artifact presents a set of operations and produces some events as shown in the diagram of figure 10. The map's operations include *add place to visit*, *delete place to visit*, and *mark a place as visited* and its events are used to inform agents about its *current place*, *next place* and the *itinerary to the next place*. The schedule book's operations are: *add schedule* and *delete schedule*; and it produces events like *today*, *current schedule*, *next schedule*, *is date available*, *next available date*, and *schedule notification*.

Finally, the guide artifact does not provide any operation since it is used just to provide information about the tourist place, like *name*, *history*, *available services*, and *related places*. Guides also inform when the *tourist is in the place*.

Since these artifacts may be necessary to achieve some organizational goals, a manual is provided to help new agents to be familiar with them. The manual describes all artifact's operators and events, as briefly presented in figure 11. To simplify the example, we used natural language to describe operations and events, however formal languages instead.

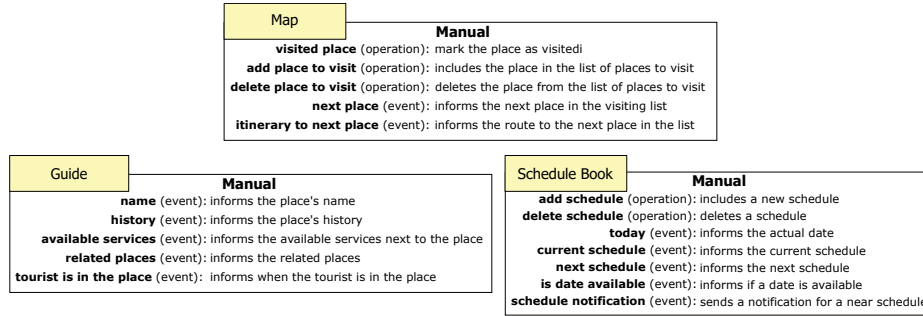


Fig. 11: Manuals for environmental artifacts

Regarding the interaction dimension, tourists and services providers interact with the environmental artifacts and with each other in order to reach goals. Since in our example the system is open at agent dimension, we use the set of roles to specify a protocol, as well as the artifacts in the environment, like in the case of the protocol of negotiation. In this protocol, a tourist and a service provider interact to determine the available dates in which a service may be provided and the service price. To define available dates, the tourist and the service provider may consult their schedule books. In order to participate in a negotiation, the agent must play the role of tourist or the role of service provider. In our method, we choose to use the AUML sequence diagram to represent the protocols of interaction, as in negotiation protocol presented in the figure 12.

The protocol is finalized when the tourist agent sends a message to the service provider to schedule the service in the agreed date. In this moment, the goal *negotiate* is achieved. In other words, the (brute) fact that an agent sends a message *count as* the organizational goal being achieved. In the specification, at least one count as relationship should be defined for each role and goal in the system defining how they are concretely realized in the environment. In the traveler assistance example, some of the count as relations are presented in figure 13.

In this case, when an agent is next to a visiting place it plays the organizational role of tourist. Thus, the brute fact *agent is next to a visiting place* **count as** *agent playing the role tourist*. Likewise, *an agent offering a service* **count as**

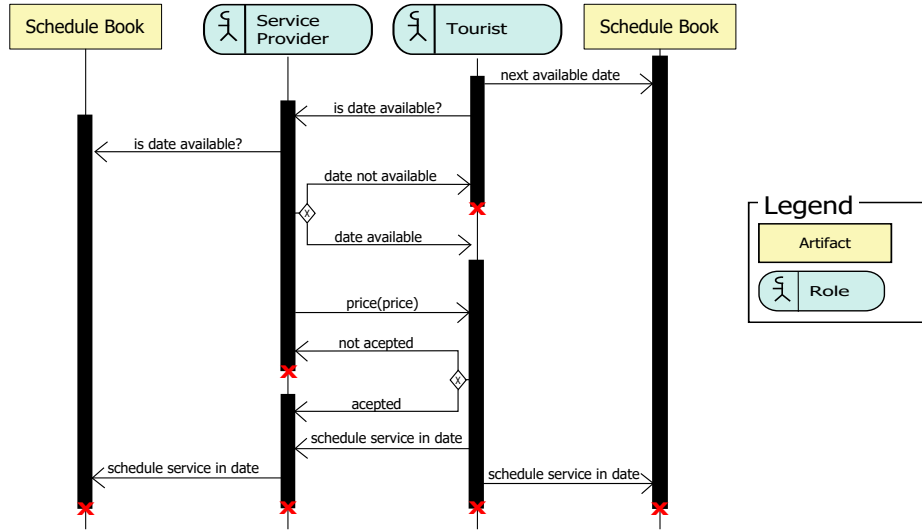


Fig. 12: Negotiation protocol

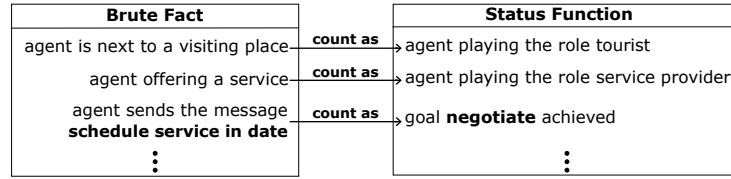


Fig. 13: Brute facts and status functions

*the agent playing the role service provider and by sending the message **schedule service in date** count as goal **negotiate** achieved.*

When a system specified as shown in this example is implemented and running, incoming agents (that were not specified since the agent dimension is open) have several facilities for them to successfully participate into the system. They can consult artifact manuals, reason about adopting a role based on required skill, reason about the consequence of their actions on artifacts based on count as relations, etc. If we consider the possibility of malicious agents entering the system, both the environment available for the agents constraints their actions and the organization regulates their behavior by means of norms. Linking concepts are thus useful to deal with openesses from agent and system perspectives.

6 Discussion and conclusion

In this paper, we present the metamodels of a method to help the specification of open MAS. We consider a MAS being composed of four dimensions, each one

concerned with a system aspect: agents, environment, interaction, and organization. For each system, a different arrangement of these dimensions may be defined, depending on the system requirements. A system is open when one or more dimensions are unknown at design time and may be integrated into the system at runtime. The design of open systems is a complex task since the system may be designed to allow the integration of unknown elements at runtime. Current methods commonly do not deal with openness specific issues since some of these methods generally deal only with concepts of agent dimension.

We propose a method in which each dimension is designed using its own set of concepts that allows the system being designed even when one dimension is unknown at design time. In other words, dimensions are independent and do not rely on others. Since those unknown elements should be integrated at runtime, we also propose a set of concepts that are used to link dimensions at runtime. They are grouped into two categories: related concepts and border concepts. Most of the related concepts are commonly used by MAS community and represent the same abstraction in different points of views. These concepts allow dimensions to work together. On the other hand, border concepts are defined to provide information that helps new elements to be familiar with those already in the system, allowing them to be properly integrated at runtime.

It is important to notice that this is a part of a work in progress. These meta-models are in their preliminary version and some improvements may be made along next steps. First, we aim to ensure that the metamodels are complete enough to allow the specification of a wide range of MAS, by comparing them with the existent ones. Next, we define a set of work products and the methodological guidelines that help the specification of the system, like the minimal set of elements that should be specified in the system. Later, we must a set of tests with the proposed method and its metamodels.

References

1. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747 – 761, 2013.
2. A. Artikis and M. J. Sergot. Executable specification of open multi-agent systems. *Logic Journal of the IGPL*, 18(1):31–65, 2010.
3. F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
4. C. Bernon, P. Glize, G. Picard, and P. Glize. The ADELFE methodology for an intranet system design. In *AOIS 2002*, pages 27–28. 2002.
5. R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.
6. L. Braubach, W. Lamersdorf, and A. Pokahr. Jadex: Implementing a bdi-infrastructure for jade agents, 2003.
7. M. Cossentino and C. Potts. PASSI: a process for specifying and implementing multi-agent systems using UML, 2002.
8. F. Dalpiaz, A. Chopra, P. Giorgini, and J. Mylopoulos. Adaptation in open systems: Giving interaction its rightful place. In J. Parsons, M. Saeki, P. Shoval,

- C. Woo, and Y. Wand, editors, *Conceptual Modeling - ER 2010*, volume 6412 of *LNCS*, pages 31–45. Springer, 2010.
9. M. de Brito, J. F. Hübner, and O. Boissier. A conceptual model for situated artificial institutions. In *Proceedings of Computational Logic in Multi-Agent Systems - 15th International Workshop CLIMA XV Proceedings*, pages 35–51, 2014.
 10. S. A. DeLoach and J. C. García-Ojeda. O-mase: a customisable approach to designing and building complex, adaptive multi-agent systems. *Int. J. Agent-Oriented Softw. Eng.*, 4(3):244–280, 2010.
 11. Y. Demazeau. From interactions to collective behaviour in agent-based systems. In *1st. ECCS*, pages 117–132, 1995.
 12. Y. Demazeau and A. C. R. Costa. Populations and organizations in open multi-agent systems. In *Proceedings of the National Symposium on Parallel and Distributed Artificial Intelligence (PDAI'96)*, 1996.
 13. V. Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht, 2004.
 14. M. Esteva, J. A. Padget, and C. Sierra. Formalizing a language for institutions and norms. In J.-J. C. Meyer and M. Tambe, editors, *ATAL*, volume 2333 of *Lecture Notes in Computer Science*, pages 348–366. Springer, 2002.
 15. J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In *LNCS n. 2935: Procs. of AOSE 03*, pages 214–230. Springer, 2004.
 16. J. Gonzalez-Palacios and M. Luck. Towards compliance of agents in open multi-agent systems. In R. Choren, A. F. Garcia, H. Giese, H. fung Leung, C. J. P. de Lucena, and A. B. Romanovsky, editors, *SELMAS*, volume 4408 of *LNCS*, pages 132–147. Springer, 2006.
 17. J. F. Hübner, J. S. Sichman, and O. Boissier. Developing organised multiagent systems using the MOISE+ model: Programming issues at the system and agent levels. *IJAOSE*, 1(3/4):370–395, 2007.
 18. W. Jamroga, A. Meski, and M. Szreter. Modularity and openness in modeling multi-agent systems. In G. Puppis and T. Villa, editors, *GandALF*, volume 119 of *EPTCS*, pages 224–239, 2013.
 19. A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008. Special Issue on Foundations, Advanced Topics and Industrial Perspectives of Multi-Agent Systems.
 20. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. Halsted, 2004.
 21. L. Padgham, M. Winikoff, S. A. DeLoach, and M. Cossentino. A unified graphical notation for aose. In *AOSE'08*, pages 116–130, 2008.
 22. A. Ricci, M. Pianti, and M. Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *AAMAS*, 23:158–192, 2011.
 23. J. Searle. *Making the Social World: The Structure of Human Civilization*. Oxford University Press, 2010.
 24. R. K. Teussop. *Management of openness within multi-agents organizations : an approach based on organizational artifacts*. Thesis, Oct. 2011.
 25. M. R. Zatelli and J. Hübner. The interaction as an integration component for the jacamo platform. In F. Dalpiaz, J. Dix, and M. van Riemsdijk, editors, *Engineering Multi-Agent Systems*, volume 8758 of *LNCS*, pages 431–450. Springer, 2014.

Two Concepts of Module, for Agent Societies and Inter-Societal Agent Systems

Antônio Carlos da Rocha Costa

PPGComp/FURG, Rio Grande, RS, Brasil.
PGIE/UFRGS - Porto Alegre, RS, Brasil
`ac.rocha.costa@gmail.com`

Abstract. This paper introduces two concepts of module, for agent societies and inter-societal agent systems. The first concept builds modules on the basis of agent organization, the second, on the basis of whole agent societies. The former serves the modularization of agent societies. The latter, the modularization of inter-societal agent systems. Both are also shown to serve as agent-based modules for conventional software systems. The two concepts are first discussed informally, and then presented formally with the help of the sketch of a module language for agent societies, in which are presented simple examples of systems using each type of module.

1 Introduction

It's a long time that agent technology endeavors to enter the main stream of Software Engineering, without much success. The area of agent-oriented software engineering developed models, methodologies and techniques that remained, for the most part, without communication with the corresponding models, methodologies and techniques for conventional software systems. Effective integration of agent systems into conventional software systems, when successful, seems to have been achieved on a case-by-case basis.

We claim that this lack of integrability and interoperability, between agent systems and conventional software systems is due to the lack of a proper notion of *modularity* for agent systems.

Thus, in this paper, we introduce an approach to the modularization of agent systems, based on the notions of *agent organizations* and *agent societies* as modules.

The paper shows both that *agent organizations* (more generally, *organization units*) may serve as modules for agent societies and for conventional software systems, and that *agent societies*, on their turn, may serve as modules for both inter-societal agent systems (a notion we define in the paper), as well as for conventional software systems.

The paper is organized as follows. Section 2.1 reviews the usual notion of software module, and its importance in conventional Software Engineering. It also reviews the usual ways to define software modules in programming languages and systems.

Section 3 reviews the concept of agent society adopted in the paper, and presents the concept of inter-societal agent system.

Sections 4 and 5 respectively introduce the way organization units may be construed as modules for agent societies, and the way agent societies may be construed as modules for inter-societal agent systems.

Section 6 shows how the notion of *artifact* may be used to allow for a way for organization units and agent societies to be integrated, as modules, in conventional software systems.

Section 7 makes use of a sketchy module language for agent societies to give three examples of use of organization units and agent societies as modules.

Section 8 considers the issue of the leveled structure of input-output ports and import-export channels.

Section 9 briefly discusses related work. Section 10 is the Conclusion.

2 Modules and the Modularization of Software Systems

2.1 The Importance of Modules

Since at least [1], the notion of *structured programming* became a dominant requirement in the *programming-in-the-small* range of activities of Software Engineering. Since at least [2], the notion of *module*, embodying combined principles of *encapsulation* (or, *information hiding*) and *hierarchical structuring*, became a serious requirement in the *programming-in-the-large* activities.

AI programming, with its tendency to depend on declaratively (functionally or logically) structured programming languages, beginning with languages LISP, Prolog, etc., was born very close to the framework of structured programming, in its programming-in-the-small tasks.

But it seems that AI never realized the requirements determined by the modularization requirements of Software Engineering, in its programming-in-the-large tasks.

A first attempt in that direction seems to have been given with the notion of *agentification* [3], where *agents* were proposed as the basic constituent modules of agent systems (see also [4]).

It happens, however, that the complexity of the *functional interfaces* of agents (the variety of their possible ways of interacting with the external context) has always prevented the adoption of agents as the modular units of those systems. And, to the best of our knowledge, no alternative notion of modularization of agent systems has been proposed, since then.

Thus, indispensable as the notion of *module* has become for the *good practices* of Software Engineering, it is no surprise that agent technology has not yet been integrated into the mainstream technology of large-scale software systems.

In this paper, we propose notions of modules for agent systems that may help to open the way for such integration.

2.2 The Concept of Software Module

We picture the general notion of *software module* as in Fig. 1. The overall module (M) is shown to be composed of sub-modules (M_1 to M_6), each sub-module endowed with an interface (I), shown split in various parts, for graphical convenience). As usual, modules represented at the same graphical level are taken to be located at the same hierarchical (logical) level¹.

The interface of the overall module M is taken to be composed of the interfaces of all the sub-modules that have interfaces connecting to the external environment (Env), as shown by the relation between Figs. 1(a) and 1(b).

The external environment Env may be composed of other software modules or other types of elements (e.g., hardware equipments).

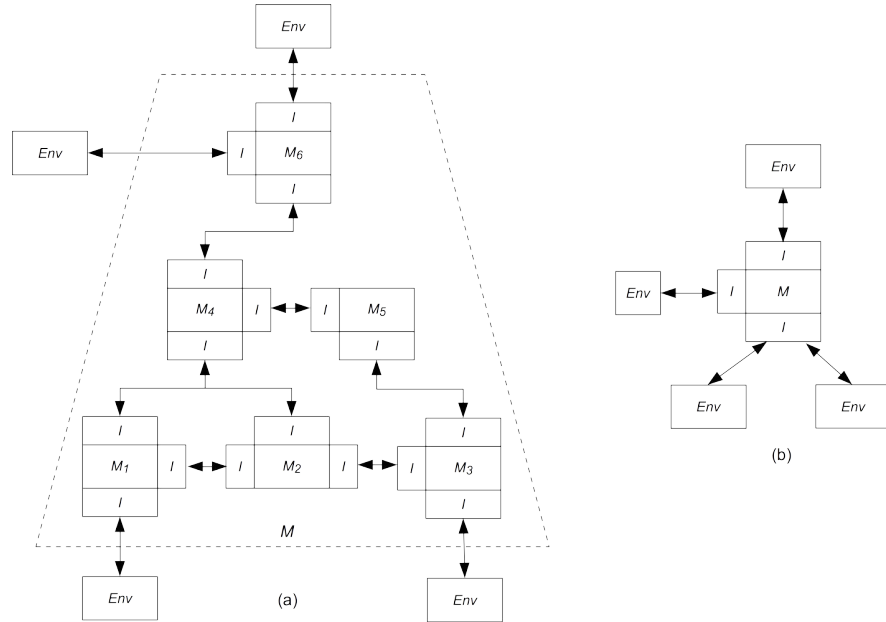


Fig. 1. A picture of the general notion of software module: (a) internal structure, (b) external structure.

2.3 Alternative Ways to Computationally Construe the Concept of Software Module

There are two main alternative ways to computationally construe the concept of software module. One seamlessly integrates the concept of module into the core

¹ Alternatively, we may graphically represent the different hierarchical levels by the recursive nesting of sub-modules, as in the example in Fig. 4.

of a single programming language, disguising it as a programming concept for the programming-in-the-small techniques.

The other, explicitly directed to the programming-in-the-large techniques, construes the concept of module in terms of directives for compilers and program linkers, making it almost independent of the different programming languages with which the modules may be programmed.

See, e.g. [5] for an example of the first way, and [6] for an example of the second way, the latter specifically regarding the modularization of systems implemented in Java.

In this paper, for simplicity, we follow the second way, informally presenting an intuitively understandable ad-hoc notation, which we use to define either type of agent-based module.

3 Agent Societies and Inter-Societal Agent Systems

In our work, we have been defining agent societies and inter-societal agent systems as follows [7–10]:

Definition 1. *An agent society is a time-indexed structure $AgSoc^t = (Pop^t, Org^t, MEnv^t, SEnv^t)$ where, at each time t :*

- *$AgSoc^t$ is the state of the agent society at that time;*
- *Pop^t is the state of the populational structure of the agent society, at that time, that is, the state of the structure composed by the agents that inhabit the society, and their behaviors and interactions;*
- *Org^t is the state of the organizational structure of the agent society, at that time, which we detail below;*
- *$MEnv^t$ is the state of the material environment of the agent society, at that time, that is, the state of the structure composed by the material objects of the society, and the causal links between them;*
- *$SEnv^t$ is the state of the symbolic environment of the agent society, at that time, that is, the state of the structure composed by the symbolic objects of the society, and the symbolic links between them.*

In what follows, we concentrate on the organizational structure of the agent societies, which we define as:

Definition 2. *The organizational structure of an agent society $AgSoc^t$ is a time-indexed structure $Org^t = (Org_\omega^t, Org_\mu^t, Org_\Omega^t)$ where, at each time t :*

- *Org_ω^t is the state of the micro-organizational structure of the agent society, that is, the state of the structure composed by the organizational roles that the agents of the populational structure perform in the society, and the behaviors and interactions performed by those organizational roles;*
- *Org_μ^t is the state of the meso-organizational structure of the agent society, that is, the state of the structure composed by the organization units that the organizational roles of the micro-organizational structure constitute in the society, and the behaviors and interactions performed by those organization units;*

- Org_{Ω}^t is the state of the macro-organizational structure of the agent society, that is, the state of the structure composed by the organizational sub-systems that the organizations of the meso-organizational structure constitute in the society, and the behaviors and interactions performed by those organizational sub-systems.

Figure 2 gives a general picture of the architecture of agent societies that conform to the above two definitions. The dashed vertical arrows denote the *implementation relations* between the various components of the agent society. The double-headed horizontal arrows denote interactions. The dotted trapezoids indicate the structural scope of their corresponding *organizational sub-systems*.

The figure does not show the *access relations* to the material environment by the organizational components of the agent society (organizational roles, organization units, etc.).

Organizations can be defined as *maximal organization units*, that is, organization units that do not belong as sub-units in other organization units.

Organizational sub-systems are implemented by *inter-organizational networks*, constituted by the organizations of the society. And the society as a whole is implemented by its *networks of organizational sub-systems*.

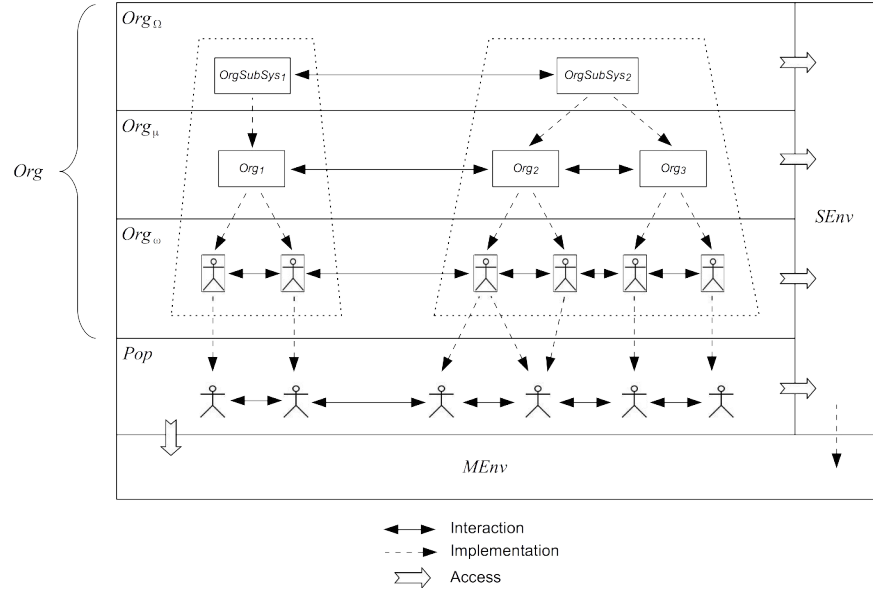


Fig. 2. Sketch of the architecture of agent societies.

On the structure given in Def. 1, we impose the following constraints:

- that agent societies be *open*, in the sense that agents may freely enter and leave them;

- that the organizational structure of an agent society be *persistent*, in the sense that it should persist in time, independently of which agents enter or leave the society (up to a minimum size of the society's population).

In addition, we define a particular type of agent society, namely, *interoperable* agent societies, that is, agent societies endowed with means that allow them to interact with each other.

In this paper, we consider a particular type of interaction means for agent societies, namely, *import-export channels*. And we call *import-export agent societies* the agent societies endowed with import-export channels.

Figure 3(a) illustrates the idea of import-export agent societies. Figure 3(b) shows a simple system constituted by import-export agent societies. We call *inter-societal agent system*, such type of system.

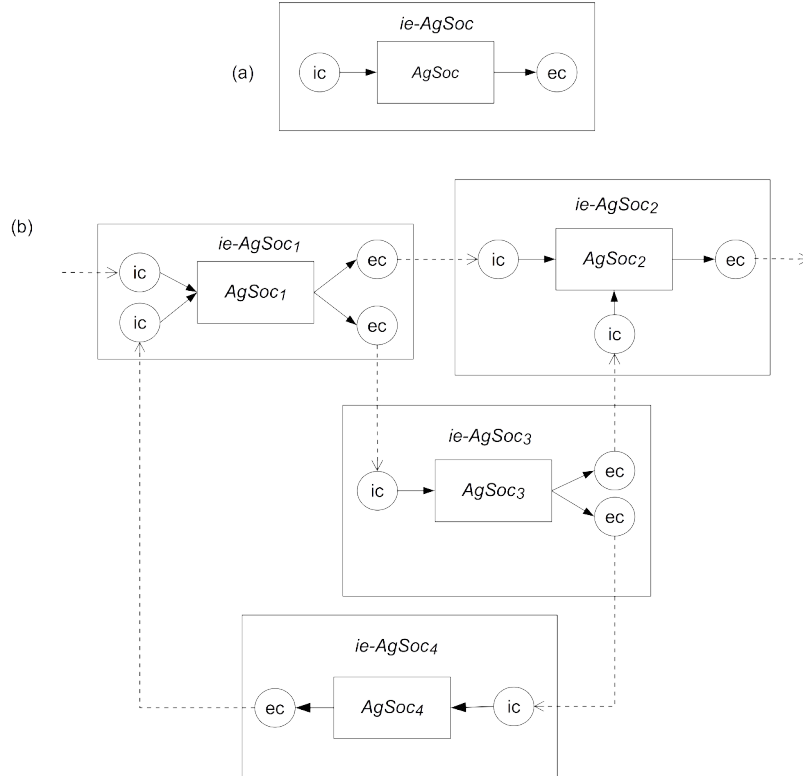


Fig. 3. An import-export agent society (a), and an inter-societal agent system (b).

Formally, we have the following:

Definition 3. An import-export agent society (or, ie-agent society) is a time-indexed structure $\text{ie-AgSoc}^t = (\text{AgSoc}^t, \text{ImpChnls}^t, \text{ExpChnls}^t)$ where, for each time t :

- AgSoc^t is the state of the agent society that constitutes ie-AgSoc^t ;
- ImpChnls^t is the set of import channels of ie-AgSoc^t ;
- ExpChnls^t is the set of export channels of ie-AgSoc^t .

In the present paper, we leave undefined the way the import and export channels of ie-AgSoc^t connect to the internal structure of AgSoc^t .

Also, we have:

Definition 4. An inter-societal agent system is a time-indexed structure $\text{IntSocAgSys}^t = (\text{IE-AgSoc}^t, \text{IE-Conn}^t)$, where, for each time t :

- IE-AgSoc^t is a non-empty set of ie-agent societies;
- IE-Conn^t is a relation between import and export channels, so that for each pair $(ec, ic) \in \text{IE-Conn}^t$ it happens that ec is an export channel of an ie-agent society and ic is an import channel of an ie-agent society.

In Fig. 3(b), the relation IE-Conn^t is denoted by the set of dashed arrows.

A Remark

Notice that, as explained below, construing organizations as modules for agent societies extends the concept of an agent society's *openness* to the *organizational level*: the concept of *organizations as modules* adequately supports the free entering and leaving of organizations in agent societies.

In the same way, the concept of *ie-agent societies as modules* adequately supports the free entering and leaving of agent societies in inter-societal agent systems.

4 Organization Units as Modules for Agent Societies

Organization units can be construed as modules for agent societies. For that, it is necessary that certain symbolic objects, of the symbolic environments of agent societies, be capable to operating as *input* or *output ports* for organization units.

Also, it is necessary that certain roles within each organization unit (module) serve as *interface roles*, regulating the exchange of elements (data, objects, etc.) that go between organization units through the input-output ports.

Formally, an organization unit that serves as a module for agent societies can be defined as:

Definition 5. An organization unit is a time-indexed structure $\text{OrgUn}^t = (\text{OR}^t, \text{Interf}^t, \text{Inp}^t, \text{Out}^t)$, where, at each time t :

- OrgUn^t is the overall state of the organization unit;

- OR^t is the set of organizational roles that compose $OrgUn^t$;
- $Interf^t \subseteq OR^t$ is the interface of $OrgUn^t$, that is, the set of organizational roles responsible for the interactions that the organization unit perform with external elements (agents, other organization units, etc.);
- $Inp^t \subseteq Port$ is the set of input ports of $OrgUn^t$, that is, the set of ports that $OrgUn^t$ uses to receive (material and/or symbolic) objects from external elements (agents, other organization units, etc.);
- $Out^t \subseteq Port$ is the set of output ports of $OrgUn^t$, that is, the set of ports that $OrgUn^t$ uses to send (material and/or symbolic) objects to external elements (agents, other organization units, etc.).

Figure 4 pictures an agent society constituted by an interconnected set of organization units. The interface roles are separated from the others by the dotted vertical lines.

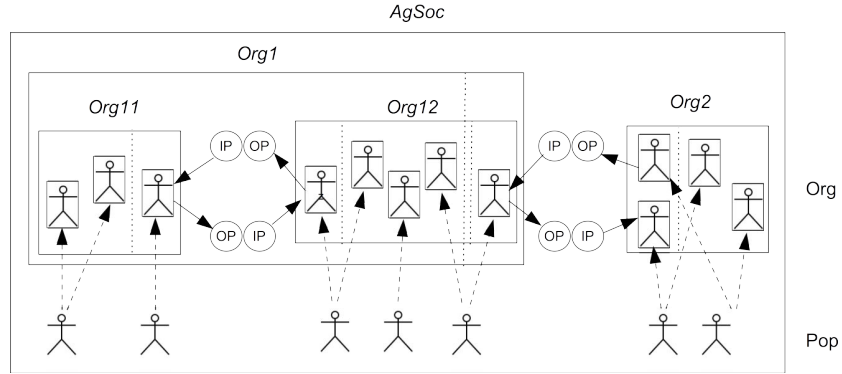


Fig. 4. A network of io-organizations that serve as modules for an agent society (the dashed lines separate the interface organizational roles of the organization units).

5 Import-Export Agent Societies as Modules for Inter-Societal Agent Systems

We limit ourselves here to remark, regarding this subject, that import-export agent societies may serve as *modules* for inter-societal agent systems, with the elements of *IE-Conn* serving as *module connectors* (see Fig. 3), and with the way of connection between the import-export channels and the internal structure of import-export agent societies being left undetermined, in the present paper, as mentioned above.

6 Organization Units and IE-Agent Societies as Modules for Conventional Software Systems

A natural way exists of construing *organization units* (in particular, maximal organization units, that is, full-fledged *organizations*) and *ie-agent societies* as modules for conventional software systems, if one realizes input-output ports and import-export channels as *artifacts* [11].

6.1 Objects and Artifacts

In conventional software systems, *objects* have been used to realize the two essential computational concepts of *processes* and the *data storage*.

In agent systems, *agents* are the concept used to realize the computational concept of process. *Artifacts* may be thought of as the concept proposed to realize data storages operated by agents.

Figure 5(a) illustrates the main features of artifacts, as artifacts are implemented in the CArtAgO platform [12]. The internal state of an artifact is modified by means of the *operations* available on the artifact's interface. Modifications in the internal state of the artifact may be observed externally by means of *observable events* generated by the artifact².

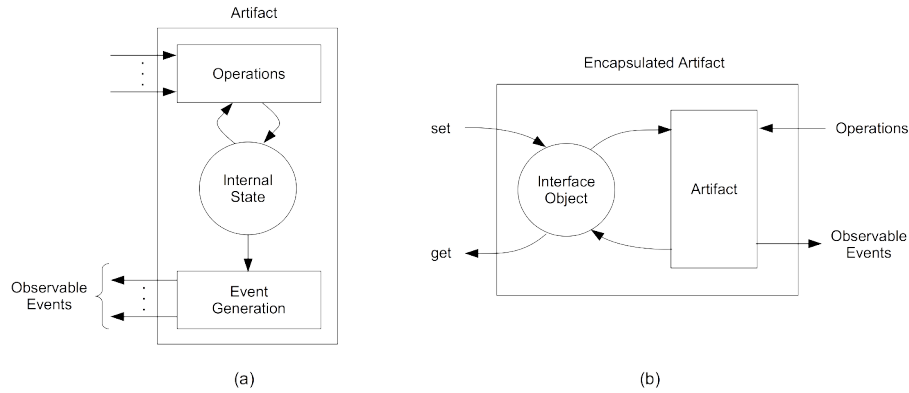


Fig. 5. General view of an artifact, and its encapsulation to support its appearance as a conventional object.

Figure 5(b) illustrates how an *artifact* may be encapsulated so as to appear as a conventional *object* to the users that access it through the left interface, while still operating in the standard artifact way to the users that access it through the right interface.

² Which, in the JaCaMo platform [13], appear as *immediate perceptions* for the agents that performed the special operation *focus*, on that artifact.

A special attention should be given to the *get* operation. It should enforce the artifact to generate an observable event capable of reporting current item of internal state that is being gotten. The artifact should be arranged so as to respond to that enforcement operation by properly generating the corresponding observable event.

6.2 Integrating Organization Units and IE-Agent Societies as Modules, in Conventional Software Systems

Organization units (in particular, full-fledged organizations) and ie-agent societies may be easily integrated into conventional software systems by realizing their input-output ports and import-export channels through artifacts, and by encapsulating them in the form suggested in Fig. 5(b).

More specifically, the organization units and ie-agent societies should integrate to the conventional software systems by presenting to them the *conventional object side* of those encapsulated artifacts.

Then, the encapsulated input-output ports and import-export channels continue to appear to the internal components (agents, roles, etc.) of such organization units and ie-agent societies with their usual artifact interfaces, as if they were not encapsulated, while appearing to the conventional software system as conventional objects that can be operated in the usual way.

Notice that the integration in the reverse direction, allowing for conventional software systems to be integrated as modules in agent systems, is a procedure that is already largely employed (e.g., it is a standard technique in the multiagent-oriented programming of the JaCaMo platform, on the basis of the so-called *artifacts* [13]).

7 Examples

To illustrate the notion of organization units and ie-agent societies as modules, we introduce here, in an informal way, the sketch of a *module language* [5] for agent societies. Figure 6 shows schematically the declaration of an agent society, with organization units as modules.

Roles, role behaviors, role interactions and input and output ports are local to the modules, they can be externally accessed. Input and output ports connected to each other by means of *connectors*. Roles can be implemented by agents of the population of the agent society.

Figure 7 illustrates the declaration of an inter-societal agent system, with ie-agent societies as modules.

Figure 8 illustrates the integration of two ie-agent societies into a conventional software system. The conventional software system is a search server, that searches the web on the basis of a combination of two societies of search agents.

```

org-unit modules:
  org-unit OU1:
    output ports:
      OP1 -> ...
    roles R1,...,Rn
    role behaviors:
      R1 -> ...
      ...
      Rn -> ...
    role interactions:
      R1 x R2 -> ...
      ...
      Rm x Rn -> ...
  org-unit OU2:
    input ports:
      IP1 -> ...
    roles R1,...,Rn
    role behaviors:
      R1 -> ...
      ...
      Rn -> ...
    role interactions:
      R1 x R2 -> ...
      ...
      Rm x Rn -> ...

agent society AgSoc1:
  population:
    ag1, ag2, ..., agm
  organization units:
    OU1
    OU2
  connections:
    OU1.OP1 -> OU2.IP1
  implementation:
    OU1.R1 -> ag1
    ...
    OU1.Rn -> agk
    ...
    OU2.R1 -> agn
    ...
    OU2.Rn -> agm

```

Fig. 6. Sketch of the declaration of an agent society, with organization units as modules.

```

org-unit modules:
  ...
ie-ag-soc modules:
  ie-ag-soc ie-AgSoc1:
    export-channels:
      EC1 -> ...
    population:
      ...
    organization units:
      ...
    connections:
      ...
    implementations:
      ...
  ie-ag-soc ie-AgSoc2:
    import-channels:
      IC1 -> ...
    population:
      ...
    organization units:
      ...
    connections:
      ...
    implementations:
      ...
inter-societal agent system ISAS1:
  ie-agent societies:
    ie-AgSoc1
    ie-AgSoc2
  connections:
    ie-AgSoc1.EC1 -> ie-AgSoc2.IC1

```

Fig. 7. Sketch of the declaration of an inter-societal agent system, with ie-agent societies as modules.

```

system WebSearch:
  org-unit modules:
    ...
  ie-ag-soc modules:
    ie-ag-soc ie-AgSoc1:
      import-channels:
        IC1 -> ...
      export-channels:
        EC1 -> ...
      ...
    ie-ag-soc ie-AgSoc2:
      import-channels:
        IC1 -> ...
      export-channels:
        EC1 -> ...
      ...
  ie-AgentSocieties:
    ie-AgSoc1
    ie-AgSoc2
  connections:
    ie-AgSoc1.EC1 -> ie-AgSoc2.IC1
main:
  procedure LookForUserCommand:
    ...
  procedure InformResultToUser(Res):
    ...
  cycle forever:
    cmd <- LookForUserCommand()
    export cmd to ie-AgSoc1.IC1
    import res from ie-AgSoc2.EC2
    InformResultToUser(res)

```

Fig. 8. Sketch of the integration of two ie-agent societies, as modules, in a conventional software system.

8 The Leveled Structure of Ports and Channels

Input-output ports connect organizational units, import-export channels connect ie-agent societies. Since both organizational units and ie-agent societies are hierarchically structured, each with several architectural levels, one should expect that ports and channels can also be seen as internally structured. For that allows one to see exchanges between organizational units and between ie-agent societies in a hierarchically structured way, with specific communication levels for each of their architectural levels.

Figure 9 illustrates the internal hierarchical structure of input-output ports and import-export channels. At each architectural level, an element which is meaningful at that level can be transmitted, on the basis of implementation relations (not shown in the figure) established between each two levels, with each implementation relation also construable as a *counts-as* relation.

In consequence, one can, as usual in hierarchical communication channels, consider in themselves the exchanges that occur at each specific level, abstracting away the exchanges that occur in higher or lower architectural levels.

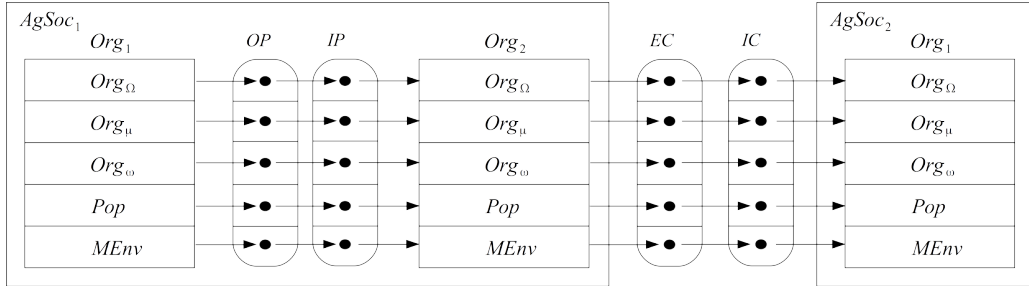


Fig. 9. The leveled structure of input-output ports and import-export channels in the internal and external modular connection of two agent societies.

9 Related Work

1) *Modularity* is an issue that has been explored in the multiagent systems area at various architectural levels.

At the *intra-agent architectural level*: Dastani et al. [14, 15] and Hindriks [16]. At the *inter-agent behavioral and interactional level*: Jamroga et al. [17], Ricci and Santi[18]. At the *intra-organizational level*: Oyen et al. [19].

To the best of our knowledge, Costa [20] was the first to treat modularity at the *inter-organizational level*, with the idea of *organizations as system modules*.

2) Regarding the particular issue of *interoperability*, a direction has been explored in the literature, concerning the so-called *interoperability of organizational models*. Coutinho and Sichman [21] provided the first extensive analysis of the

possibilities concerning that approach. Aldewereld [22] (see in particular [23]) furthered the issue.

From our point of view, however, that effort should better be seen as an concerning the *compatibility of organizational models*, rather than the *interoperability of agent systems* (and surely not the interoperability of agent-based and conventional software systems).

3) Regarding the adaptation of software engineering concepts, techniques and methodology to agent technology, the literature is vast, under the acronym AOSE (Agent Oriented Software Engineering), and requires no review here.

However, the usual AOSE approach is mostly centered around the idea of taking *agents as system modules*. And, as the present paper attempts to show, that is a too low architectural level to support modular interoperability between agent-based and conventional software system: the higher architectural levels of *organization units* and *agent societies* seem to be right ones.

10 Conclusion

In this paper, we have proposed a means to construe organization units as modules for agent societies, and organization units and import-export agent societies as modules for conventional software systems.

At the conceptual level, the proposed construal of organization units as modules imply the introduction of *input-output ports* as means for the organization units to communicate with their external contexts, and the introduction of *import-export channels* as corresponding means for agent societies.

At the implementation level, in order to have a definite way to sketch a module language for agent societies, we have assumed that input-output ports and import-export channels are realized by artifacts. This allows such communication means to be suitably encapsulated, so that they can appear as conventional objects, allowing organization units and ie-agent societies to be integrated, as modules, into conventional software systems.

On the basis of modular techniques similar to the one introduced here, allowing the seamless integration of agent systems into conventional software systems, it may happen that agent technology finds its way into (a place for its own in) mainstream Software Engineering.

In addition, seen modules as structures that perform services for the systems where they are integrated, the proposal allows for agent societies to serve as an architectural foundation for service-oriented computing systems [24].

Acknowledgments

The author thanks the reviewers for their very useful comments. This work had partial financial support from CNPq (Proc. 310423/2014-7).

References

1. Dijkstra, E.: Go to statement considered harmful. *Communications of the ACM* **11** (1968) 147–148
2. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. *Communications of the ACM* **15** (1972) 1053–1058
3. Shoham, Y.: Agent oriented programming. *Artificial Intelligence* **60** (1993) 51–92
4. Sycara, K.: Multiagent systems. *AI Magazine* **19** (1998) 79–92
5. Friedman, D.P., Wand, M.: *Essentials of Programming Languages*. MIT Press, Cambridge (2008)
6. Reinhold, M.: The state of the module system. Available online at: <http://openjdk.java.net/projects/jigsaw/spec/sotms/> (2016)
7. Costa, A.C.R., Demazeau, Y.: Toward a formal model of multi-agent systems with dynamic organizations. In: *Proc. of ICMAS 96 – 2nd. Int’l Conf. on Multiagent Systems*, Kyoto, IEEE (1996) 431
8. Costa, A.C.R., Dimuro, G.P.: A minimal dynamical organization model. In Dignum, V., ed.: *Handbook of Multi-Agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global, Hershey (2009) 419–445
9. Costa, A.C.R.: On the bases of an architectural style for agent societies: Concept and core operational structure. *Open publication on www.ResearchGate.net* - DOI: 10.13140/2.1.4583.8720 (2014)
10. Costa, A.C.R.: Ecosystems as agent societies, landscapes as multi-societal agent systems. In Adamatti, D.F., ed.: *Multiagent Based Simulations Applied to Biological and Environmental Systems*. IGI Global, Hershey (2016) 1–16
11. Ricci, A., Viroli, M., Omicini, A.: Programming MAS with Artifacts. In Bordini, R.P., Dastani, M., Dix, J., El Fallah Seghrouchni, A., eds.: *Programming Multi-Agent Systems*. Volume 3862 of *LNAI*. Springer (2006) 206–221 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26/jul/2005. Revised and Invited Papers.
12. Ricci, A., Santi, A., Piunti, M.: Cartago – common artifact infrastructure for agents open environments (2013) Website at <http://cartago.sourceforge.net>.
13. Boissier, O., Bordini, R., Hübner, J.F., Ricci, A., , Santi, A.: Multi-agent oriented programming with JaCaMo. *Science of Computer Programming* **78** (2013) 747–761
14. Dastani, M., Mol, C.P., Steunebrink, B.R.: Modularity in agent programming languages. In Bui, T.D., Ho, T.V., Ha, Q.T., eds.: *Intelligent Agents and Multi-Agent Systems (LNCS vol. 5357)*, Springer (2008) 139–152
15. van Riemsdijk, M.B., Dastani, M., Meyer, J.J.C., de Frank S. Boer: Goal-oriented modularity in agent programming. In Nakashima, Y., Wellman, M., Weiss, G., Stone, P., eds.: *Proceedings of AAMAS 2006*, ACM (2006) 1271–1278
16. Hindriks, K.: Modules as policy-based intentions: Modular agent programming in goal. In Dastani, M., Seghrouchni, A.E.F., Ricci, A., Winikoff, M., eds.: *Intl. Workshop on Programming Multiagent Systems - ProMAS 2007 (LNCS vol. 4908)*, Springer (2008) 156–171
17. Jamroga, W., Męski, A., Szreter, M.: Modularity and openness in modeling multi-agent systems. In Puppis, G., Villa, T., eds.: *Fourth Int. Symposium on Games, Automata, Logics and Formal Verification*. Volume 119., *EPTCS* (2013) 224–239
18. Ricci, A., Santi, A.: Concurrent object-oriented programming with agent-oriented abstractions - the ALOO approach. In Jamali, N., Ricci, A., Weiss, G., eds.: *AGERE!’13 Workshops*, ACM (2013) 127–138

19. Oyenán, W.H., DeLoach, S.A., Singh, G.: Exploiting reusable organizations to reduce complexity in multiagent system design. In Gleizes, M.P., Gomez-Sanz, J., eds.: AOSE 2009 (LNCS vol. 6038). (2011) 3–7
20. Costa, A.C.R.: Proposal for a notion of modularity in multiagent systems. In van Riemsdijk, M.B., Dalpiaz, F., Dix, J., eds.: Informal Proceedings of EMAS 2014, AAMAS @ Paris (2014)
21. Coutinho, L., Sichman, J.S., Boissier, O.: Modelling dimensions for agent organizations. In Dignum, V., ed.: Multi-Agent Systems: Semantics and Dynamics of Organizational Models. IGI Global (2009) 18–50
22. Aldewereld, H., Boissier, O., Dignum, V., Noriega, P., Padget, J., eds.: Social Coordination Frameworks for Social Technical Systems. Springer, Berlin (2016)
23. Aldewereld, H., Álvarez Napagao, S., Garcia, M.E., Sanz, J.G., Jiang, J., Cardoso, H.L.: Conceptual map for social coordination. In Aldewereld, H., Boissier, O., Dignum, V., Noriega, P., Padget, J., eds.: Social Coordination Frameworks for Social Technical Systems. Springer, Berlin (2016) 11–23
24. Singh, M.P., Huhns, M.N.: Service-Oriented Computing - Semantics, Processes, Agents. Wiley (2005)

An Automated Approach to Manage MAS-Product Line Methods

Sara Casare¹, Tewfik Ziadi²

¹ LTI, University of São Paulo, Brazil

² LIP6, University Paris-Sorbonne, France

`sjcasare@uol.com.br, tewfik.ziadi@lip6.fr`

Anarosa A. F. Brandão^{2,3}, Zahia Guessoum^{2,4}

³ Computing Engineering and Digital Systems Department
University of São Paulo, Brazil

⁴ CReSTIC, University Reims, France

`anarosa.brandao@poli.usp.br, zahia.guessoun@lip6.fr`

Abstract. Multiagent systems (MAS) can vary in several ways: by involving different agents, distinct interaction patterns, various forms of agent organizations and environments. One promising approach to consider this variability in MAS is the use of the concept of Multiagent System-Product Line (MAS-PL). The idea is to implement a family of MAS that belong to the same domain, instead of a single MAS. However, there is still a lack of methodological support to develop MAS-PL. This paper tackles this problem with a rigorous respect of software product line principals. We propose an automated approach, the Meduse for MAS-PL, which provides families of MAS-PL methods that offer software product line best practices integrated with existing MAS development approaches. To illustrate, we present a case study involving a family of MAS-PL methods that extends Gaia and Tropos.

Keywords: multiagent systems, software product line, variability, method

1 Introduction

Managing variability in Multiagent systems (MAS) has been identified during the last years as one of the main issues within Agent-Oriented Software Engineering (AOSE) [2][10][18][19]. Indeed, the intrinsic properties of MAS, such as modularity, make them variability-rich systems. The notion of software variability is defined as the ability of a software system to be changed, customized or configured for use in a particular context [25]. Therefore, AOSE approaches should consider this variability to design and develop not only a single MAS, but families of systems at the same time.

More than twenty AOSE methods have been proposed by the MAS community for multiagent systems development. Each of them proposes a set of development activi-

ties for analyzing, designing, and implementing the typical MAS components: agents, environment, interaction, and organization [12]. Gaia [26] and Tropos [3] are among the most popular AOSE methods. However, they only proposed to design and develop a single MAS variant at time. Especially, they do not propose in their initial definition any explicit activity to manage variability and develop families of systems rather than a single system. One promising approach for tackling this limitation has been already identified by the MAS community since the early 2000's through the concept of Multiagent System-Product Line (MAS-PL) [10]. Instead of considering a single MAS, MAS-PL aims to organize a family of multiagent systems according to their similarities (i.e., commonalities) and differences (i.e., variabilities) in order to build MAS customized according to specific needs. MAS-PL reuses concepts proposed by Software Product Line Engineering (SPLE), which is a systematic approach for variability management proposed by the Software Engineering community [1] [8].

Although introduced into the AOSE community, there is still a lack of methodological support to implement MAS-PL. As we will see in Section 2, among the more than twenty AOSE methods, only few of them support the specificities of MAS-PL, e.g. Gaia-PL[11], Peña et al [19], and Nunes et al.[18]. In these methods the proposed extensions are very light and do not cover all Software Product Line activities (i.e. Domain Engineering and Application Engineering). Moreover, they are tailored for MAS families in specific domains and may not be suited for MAS families in diverse domains.

This paper tackles the mentioned issues by proposing **Meduse for MAS-PL**, an automated approach that provides steady methodological support for MAS-PL development by integrating SPLE best practices with several existing AOSE methods. Indeed, this approach results from cooperation between AOSE and SPLE teams: we capitalize all knowledge on SPLE activities in a way that such activities can be automatically incorporated into AOSE existing methods as a set of SPLE best practices in order to develop MAS families in diverse domains. The proposed approach follows Method Engineering techniques [4][14] and is built upon the Meduse framework [7]. To illustrate our approach we present a case study that shows how we can provide MAS-PL methods that take into account SPLE best practices to extend two popular AOSE methods, Gaia and Tropos.

The paper is organized as follows: Section 2 presents background and motivations. Section 3 presents our approach for dealing with method extensions to develop MAS-PL. Section 4 illustrates such an approach using a case study. Finally, Section 5 concludes this work and discusses future work.

2 Background and Motivations

In this section we present some basic notions that are essential for understanding the main aspects of our approach related to SPLE and MAS-PL methods.

2.1 Multiagent System Product Line Methods

MAS-PL methods reuse concepts of SPLE for MAS families' development. Before discussing existing MAS-PL methods, we will briefly introduce the general SPLE framework. Fig. 1 shows its four main activities - domain analysis, domain implementation, requirement analysis, product derivation - organized in the Domain and Application Engineering levels, as well as in the Problem and Solution spaces [1].

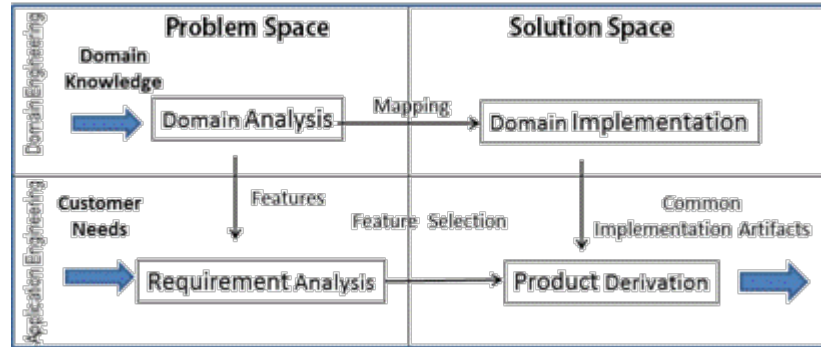


Fig. 1. Software Product Line Engineering general framework [1]

Domain analysis: this activity aims to define the scope of the domain and explicitly specify commonality and variability between products that are included in the product line. The results of domain analysis are usually documented in a variability model. Several formalisms have been proposed to specify the variability model. Among them, we distinguish *Feature Model* [16] and *Decision Model* [23]. The former derives from the work on Feature Oriented Domain Analysis (FODA) [16], while the latter has its roots on the Synthesis method [5] [23].

A feature consists of a distinctive user-visible characteristic of a software product line, used to represent identifiable functional abstractions that shall be present in the final product [16]. A feature model describes relationships between features, and formally specifies which feature selections are valid. This is made by hierarchically organizing features in a tree, where edges are used to represent parent-child relationships (see next section, Fig. 3). These parent-child relationships could be of the following types: *mandatory* (the child feature must be present in a product whenever its parent appears); *optional* (the child feature may be present when its parent appears); *alternative* (exactly one child feature must be present when the parent feature appears); *or* (at least one child feature must be present whenever the parent feature appears). Moreover, a set of cross-branch constraints is used to indicate dependencies between features pertaining to different tree branches. Finally, in order to establish their consistency feature models can be described through a set of propositional formulas. Thus, a Satisfiability (SAT) solver tool can be used to determine the feature model consistency, i.e. if it will generate at least one valid product, or whether a given product is valid.

Decision Model is another way of modelling variabilities. In this kind of models, decisions describe the variabilities available in a product line, and specify the set of choices during product derivation. Therefore, taking a decision involves analyzing multiple options and then selecting those that better reflect customers' needs.

Domain Implementation: In this activity reusable artifacts (also called assets) are implemented and organized according to the specified variability model.

Among the domain implementation approaches we can cite Delta-oriented Programming [20][21], a compositional and modular approach to manage variability based on modifications applied to a core product, which is transformed into another variant of the product line by incrementally applying a set of *delta modules* that propose additions, removals, or alterations of elements. A *condition* is a propositional constraint attached to every delta module through a *when* clause and it determines for which features the specified modifications are to be carried out. Therefore, conditions create the connection between the modifications prescribed in delta modules and the features. A *list of delta modules* and *attached conditions* determines the modifications required to implement different products of the product line, as well as the order in which such modifications shall be applied. Indeed, such a list groups delta modules in ordered partitions and partitions can be partially ordered, i.e., while the order of partitions is fixed, deltas in the same partition can be applied in any order. Finally, the core product can be an empty product.

Requirement Analysis: This activity is defined at the application engineering level. It aims to consider needs of a specific customer in order to select the required variabilities (e.g. feature selection), also called a configuration [1]. The idea is to instantiate the variability model according to the customer needs.

Product Derivation: Once we have a selection of the required variabilities representing customer needs, the product derivation activity aims to generate (or derive) the product itself. As underlined by Apel et al. [1], depending on the implementation approach this activity can be automated. For compositional approaches as Delta-oriented Programming, the idea is to use what is referred as *composer* to derive product variants.

Extending AOSE methods to support MAS-PL should consider the integration of the different SPLE activities discussed above. As mentioned early, there are just a few MAS-PL methods. Each of them extends particular AOSE method(s) to integrate SPLE activities. However, this integration is often partial and it only concerns some of the SPLE activities previously presented. For instance, Nunes et al. propose a Domain Engineering method to develop MAS-PL that uses PLUS [13] as the SPLE approach, and PASSI [9] combined with MAS-ML [22] as AOSE method.

Peña et. al. propose the use of MaCMAS and UML to define the core architecture of a MAS-PL. Their method only considers the Domain Engineering level and it adopts MaCMAS models in several levels of abstraction to guide the building of the MAS-PL variability model. In the same way as Nunes and colleagues, Pena and colleagues' approach results into a single MAS-PL, instead of a family one. Dehlinger and Lutz [10] [11] propose an approach that combines SPLE techniques and Gaia. The method is called Gaia-PL and it is devoted to the Domain Engineering level of a software product line. Their goal was reusing requirements specifications and a heu-

ristic is provided to guide the building of the feature model. Nevertheless, the proposed MAS-PL approaches may not support different application domains than the ones presented as running example.

2.2 Motivating Example

We consider the Tropos method as a running example to discuss the current methodological limitations in the MAS-PL development: MAS-PL methods propose a partial integration of SPLE activities with particular AOSE methods.

Tropos offers a set of development phases to deal with requirements gathering as well as to analyze, design, and code two MAS components: agents and interaction. Fig. 2 depicts the five subsequent phases of Tropos, from requirement to implementation phases.

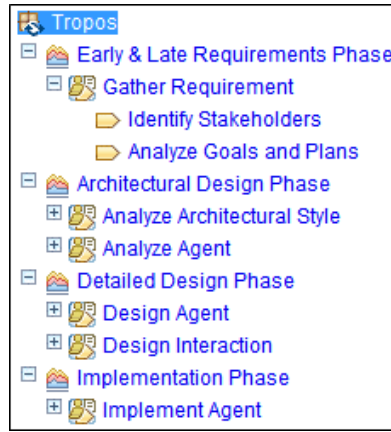


Fig. 2. The Tropos phases using the SPEM notations.

We follow the standard notations of the *Software and System Process Engineering Meta-model* (SPEM) [17], which is the *de facto* standard for representing development methods [15]. As initially proposed, Tropos only considers the phases that are related to the development of a single MAS without any methodological support for MAS-PL development. To be suitable for MAS-PL development, the original phases of Tropos (see Fig. 2) should be extended by integrating the SPLE activities. For instance, we need to add the domain analysis phase to specify variability. In addition, we also need to add activities that are related to domain implementation and product derivation. Many interesting issues can be considered in this context:

- How can we integrate the different SPLE activities with the Tropos method of Fig. 2, concerning both MAS-PL domain and application engineering levels?
- How can we provide MAS-PL methods based on Tropos for developing MAS families in diverse domains?
- As early discussed in Section 2.1, there are two kinds of formalisms to specify the variability model during SPLE domain analysis: Feature Mod-

el and Decision Model. Then, how can we manage these different integration possibilities into Tropos?

- How can we automatically generate a new MAS-PL method based on Tropos and SPLE best practices according to project needs?

The next section presents how our approach deals with these issues. As we will show, Meduse for MAS-PL can automatically generate variants of MAS-PL method based on Tropos, as well as based on other AOSE methods like Gaia or PASSI. Moreover, it allows the fully integration of SPLE activities with these AOSE methods. Finally, our approach provides MAS-PL methods for developing MAS families in diverse domains.

3 From MAS Methods to MAS-PL Methods with Meduse

In this section we present the Meduse for MAS-PL approach. In few words, it proposes the extension of existing AOSE methods to explicitly support MAS-PL development by integrating SPLE activities with them according to project needs. Through this approach we aim to provide steady methodological support for MAS-PL development following SPLE best practices.

To achieve such a goal our approach takes advantage of Method Engineering techniques. Moreover, it is based on the Meduse framework, which itself adopts SPLE principles. Therefore, Meduse for MAS-PL uses SPLE principles in two levels: proposing the use of families of methods to generate variants of MAS-PL methods and, in this case, SPLE principles are used in a meta-level fashion to generate method variants, instead of generating product variants. Next, such method variants will be used to develop MAS-PL and then to generate MAS variants, i.e. product variants.

Since our approach is based on the Meduse framework, we start presenting it.

3.1 Meduse in a nutshell

Meduse Framework is a general approach to generate software development methods. Meduse adopts SPLE principles to manage method variability, as well as to automatically derive valid method variants. Moreover, it adopts Method Engineering principles to manage reusable method artifacts, so-called method fragments, which consist of standardized building blocks based on a coherent part of method [14]. Method variants are built upon these fragments.

Fig. 3 presents the big picture of the Meduse framework, showing from the method domain analysis to the final method variant that is automatically generated according to project needs. First, Meduse proposes to represent method domain knowledge in terms of similarities and differences among variants of a same method family by means of a feature model.

Second, during method domain implementation Meduse uses reusable method fragments together with a compositional approach based on Delta-oriented programming. Thus, method derivation relies on the application of delta modules over an empty method: the modification proposed by a delta module consists of the additions

and/or removals of reusable method fragments from the method variant. Therefore, one can define the partially-ordered sequence in which method fragments will appear in the method variant, since delta modules are grouped in fixed-ordered partitions, and modules in the same partition can be applied in any order (see Section 2).

Third, method variants are specified through a method configuration, which consists of a set of features selected according to project needs. Finally, Meduse proposes a tool – the Meduse Composer - that automatically derives the specified method variant: it takes the empty method as starting point of the work breakdown structure and incrementally adds or removes method fragments according to the modifications specified in the delta modules connected with the selected features through attached conditions.

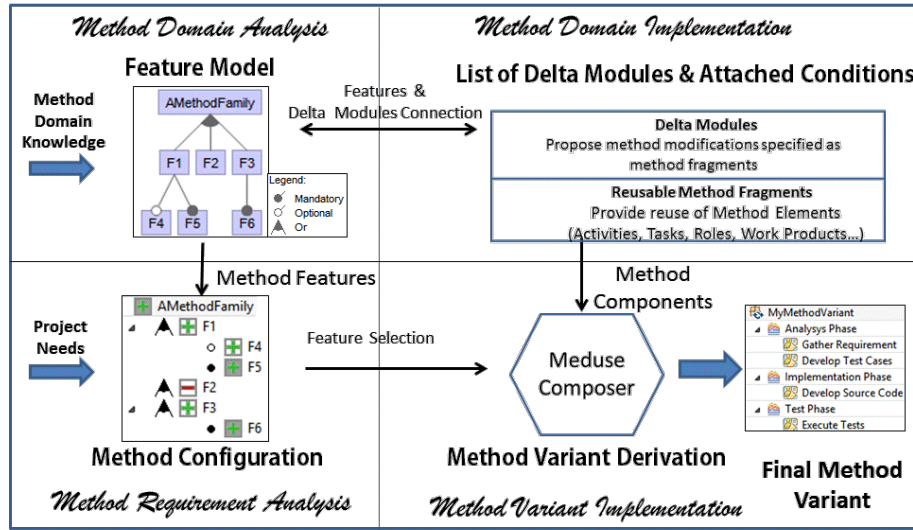


Fig. 3. A big picture of Meduse framework

3.2 Managing MAS-PL Method Family with Meduse

After introducing the Meduse framework, in the following we present how our approach provides MAS-PL method variants that integrate SPLE activities with AOSE methods in order to cover both Domain and Application Engineering levels: from MAS-PL Domain Analysis to Implementation, from MAS-PL Requirement Analysis to Product Derivation.

Such method variants are generated through method families where the MAS-PL methodological knowledge is represented as method similarities and differences by means of a feature model. Thus, on one hand features can represent the various AOSE methods that may be extended, like Tropos, Gaia, or PASSI. On the other hand, features can represent a large set of SPLE activities and related techniques that may be chosen as best practices to develop MAS-PL. Examples of such activities are those described by the general SPLE framework proposed in [1] (see Section 2). One

of the main advantages of using feature models to specify a MAS-PL method family is that we can use SAT solver tools to determine whether such a feature model is consistent, i.e. if at least one valid method configuration exists, and whether or not a given configuration is valid. Therefore, a valid MAS-PL method variant is guaranteed.

The implementation of such MAS-PL method families involves developing reusable method fragments sourced from SPLE approaches and AOSE methods. We have capitalized - as a set of reusable method fragments - all knowledge on SPLE activities proposed in [1]. Concerning SPLE related techniques, our approach offers reusable fragments that provide detailed guidance to deal with feature and decision models, sourced from FODA and Synthesis, respectively. All of these method fragments encompass fine-grained elements, like tasks, work products, and roles. On the other hand, fragments related to AOSE methods may be developed from scratch or provided by existing libraries of AOSE method fragments, like the Medee framework [6], among others.

Next, a MAS-PL method variant is specified through a method configuration that consists of a selection of features - concerning both AOSE and SPLE development aspects - that meet the needs of the MAS-PL project at hand. For instance, one may select Tropos as the AOSE method to be extended by integrating SPLE activities related to the domain engineering.

Finally, the derivation of a MAS-PL method variant is achieved according to the following steps: (i) find all delta modules that shall be applied to the MAS-PL method variant, i.e., those delta modules attached to a valid condition. Valid conditions are those that consist of propositional formulas - defined over the set of features - evaluated to true for a given method configuration. For instance, the propositional formula $A \wedge B$ is evaluated to true whenever a method configuration includes these two features, and therefore all delta modules attached to this condition are taken into account during method derivation; (ii) generate the MAS-PL method variant by applying the modification proposed by the selected delta modules respecting the order defined by the list of delta modules, involving method fragments sourced from both AOSE methods and SPLE approaches.

We have developed a prototype implementation of Meduse for MAS-PL, in which we adopted FeatureIDE [24] as tool to specify the feature model and to create method configurations, as well as SPEM and Eclipse Process Framework (EPF)¹ to manage method fragments, delta modules and their attached conditions. Interested readers may access the Meduse for MAS-PL website² to see available reusable method fragments for MAS-PL development and MAS-PL method variants automatically derived during the case study presented in the next section.

4 Creating MAS-PL Method Family: a Case Study

In this section we present a case study that shows how we can use our approach to extend two AOSE methods - Gaia and Tropos - to support MAS-PL development

¹ <https://eclipse.org/epf/>

² <https://pages.lip6.fr/Tewfik.Ziadi/EMAS17/>

following SPLE best practices. It consists of creating a MAS-PL method family and then derivate several method variants.

The scope of this method family is defined as follows. It aims at providing method variants to develop MAS product lines based on Gaia or Tropos. Moreover, such method family would offer two techniques for modeling MAS variabilities - feature model and decision model - as well as it would deal with the development of four MAS components: Agent, Environment, Interaction, and Organization. Finally, this method family would involve both Domain and Application Engineering levels in order to provide a full SPLE development cycle: from MAS-PL Domain Analysis to the MAS-PL Product Derivation. It should be noted that another scope definition, for instance involving different MAS components or other AOSE methods, like PASSI, is also possible and would give rise to a different MAS-PL method family.

The resulting feature model contains eighteen features, where eleven of them represent the similarities among the method variants pertaining to such a family, while the remainder seven features represent how such method variants may vary. We (re)used a set of method fragments sourced from Gaia, Tropos, FODA, Synthesis, and Apel et al. to implement the method family, and the Meduse Composer to derive the twenty possible variants. The remainder of this section describes this case study in detail.

4.1 Feature Model for a MAS-PL Method Family

As aforementioned, similarities and differences among the method variants of our MAS-PL method family were specified through the feature model presented in Fig. 4. This feature model is composed of eighteen features and one constraint (environment v organization => Gaia). The root of that diagram is labelled with *MAS_PL_Method* to represent a MAS-PL development method. It has three mandatory child features: *MASComponent*, *MASMethod*, and *SPLEngineering*.

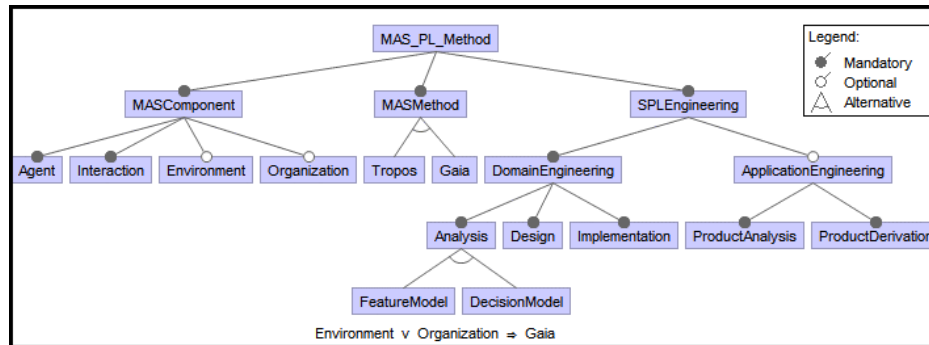


Fig. 4. A feature model diagram for a MAS-PL method family

MASComponent has two mandatory and two optional child features – *Agent* and *Interaction*, *Environment* and *Organization* - respectively, because in this case study we consider that a MAS-PL method must cover at least the development of agents and their interaction, and may cover also environment and organization development.

Thanks to the unique constraint defined in this model, whenever a method variant includes the features *Environment* or *Organization* it must include *Gaia*, because *Gaia* deals with the development of the four MAS components, while *Tropos*, the optional MAS-Method child feature, takes into account only the development of Agent and Interaction.

Moreover, *SPLEngineering* has the two SPLE levels as child features: *DomainEngineering* and *ApplicationEngineering*. The former is a mandatory feature while the latter is an optional one because in this case study we consider that a MAS-PL method must propose at least the Domain Engineering level. The *DomainEngineering* feature has three mandatory child features – *Analysis*, *Design*, *Implementation* – which correspond to the three domain development phases proposed by Apel et al. Additionally, *Analysis* offers two alternative child features - *FeatureModel* and *DecisionModel* - that represent the choice of SPLE techniques to model MAS variability. Therefore, exactly one of these techniques must be present in a MAS-PL method variant.

Finally, whenever a MAS-PL method includes the *ApplicationEngineering* feature it must include its two child features: *ProductAnalysis* and *ProductDerivation*. It should be observed that such a feature model corresponds to the previously described scope. Another scope would give rise to a different feature model.

4.2 Domain Implementation for a MAS-PL method family

While similarities and differences among the method variants of the MAS-PL method family were specified through a feature model, we used Delta-oriented programming to implement such features. Therefore, the implementation of that MAS-PL method family comprised a list of delta modules and attached conditions. Moreover, these delta modules encompass method fragments sourced from the two AOSE methods considered in our scope, *Gaia* and *Tropos*, as well as from FODA, Synthesis, and Apel et al. approach.

The rest of this section describes how we implemented those delta modules and method fragments, starting with method fragments.

Method Fragments for MAS-PL Family

In order to deal with SPLE activities and techniques we reused ten fragments provided by Meduse for MAS-PL. These fragments were sourced from FODA (for feature model), Synthesis (for decision model), and Apel et al. (for SPLE domain and application activities). Besides, method fragments sourced from *Gaia* and *Tropos* were provided by the Medee Framework: (i) six fragments sourced from *Gaia* were used to guide the analysis and design of agents, environment, interaction, and organizations; (ii) five fragments sourced from *Tropos* were used to analyze and design agents and interaction. Summing up, this case study involved twenty-one method fragments.

Fig. 5 shows two examples of these method fragments, one sourced from *Tropos* (*Gather MAS-PL Requirement with Tropos*) and other sourced from Synthesis (*Define MAS-PL Decision Model*). The first fragment encompasses two tasks proposed by *Tropos* for dealing with MAS requirements gathering: *Identify Stakeholders* and *Ana-*

alyze *Goals and Plans*. Such tasks are performed by the System Analyst role and produce two diagrams as work products: *Tropos Actor Diagram* and *Tropos Goal Diagram*, respectively. Since Synthesis proposes that we should define the SPLE domain scope before defining the decision model, the second fragment presented in Fig. 5 encompasses two tasks, *Define Domain Scope* and *Define Decision Model*, which produce *Domain Definition* and *Decision Model* as work products, respectively.

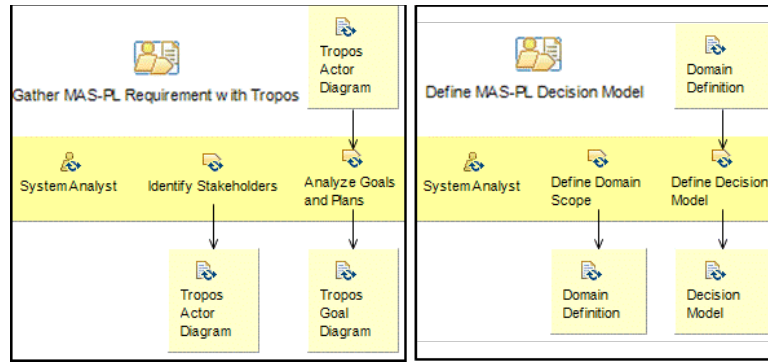


Fig. 5. Examples of two method fragments for a MAS-PL method family, sourced from Tropos (left) and Synthesis (right).

Delta Modules and Attached Conditions for MAS-PL Method Family

A list of eleven delta modules and attached conditions were implemented to specify the modifications to be applied during the derivation of MAS-PL method variants according to a given method configuration.

Fig. 6 (left) illustrates four of these delta modules, *D-TroposFMAAnalysis*, *D-GaiaDMAAnalysis*, *D-TroposDMAAnalysis*, and *D-GaiaFMAAnalysis*, as well as the comprised method fragments of the first two delta modules. Note that two of these fragments correspond to those depicted in Fig. 5, namely *Gather MAS-PL Requirement with Tropos* and *Define MAS-PL Decision Model* fragments.

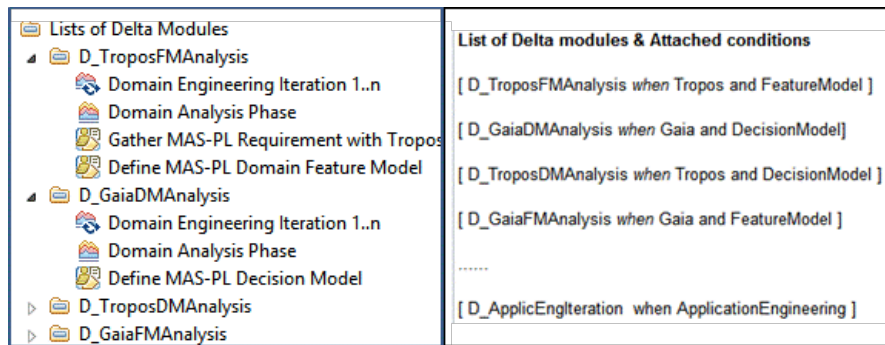


Fig. 6. A partial representation of the List of Delta modules containing method fragments (left), and the list of Delta modules and attached conditions (right)

Moreover, the list of delta modules and attached conditions determines the order in which such delta modules must be applied during method derivation (see Fig. 6 right). As explained before, these attached conditions allow the connection of delta modules with combinations of features. For instance, a MAS-PL method variant would include the four method fragments defined in the *D-TroposFMAalysis* delta module whenever the *Tropos* and *FeatureModel* features were selected to configure the MAS-PL method variant.

Method Configuration and Derivation of MAS-PL Method Variants

After implementation, the MAS-PL method family is ready to give rise to method variants according to a given method configuration. As shown in Table 1 (left), the eighteen features proposed in this case study offered twenty possible MAS-PL method configurations, four of them based on Tropos and sixteen based on Gaia.

Table 1. MAS-PL Method Configurations and Derived MAS-PL method variants.

MAS-PL Method Configuration																			MAS-PL Method Variants		
Configuration #	MAS-PL Features																	num. of Selected Features			
	F1 MAS Component	F2 Agent	F3 Environment	F4 Interaction	F5 Organization	F6 MAS Method	F7 Gaia	F8 Tropos	F9 SPL	F10 Domain Engineer.	F11 Analysis	F12 Feature Model	F13 Decision Model	F14 Design	F15 Implementation	F16 Applic.	F17 Product Analysis				F18 Product Derivation
1	*	*		*		*		x	*	*	*	*	x		*	*			11	Tropos-PL Variants	V1: Domain Eng. w/ FM & A.I.
2	*	*		*		*		x	*	*	*	*		x	*	*			11		V2: Domain Eng. w/ DM & A.I.
3	*	*		*		*		x	*	*	*	*	x		*	*	x	x	14		V3: Domain&Application Eng. w/ FM & A.I.
4	*	*		*		*		x	*	*	*	*		x	*	*	x	x	14		V4: Domain&Application Eng. w/ DM & A.I.
5	*	*		*		*	x		*	*	*	*	x		*	*			11	Gaia-PL Variants	V1: Domain Eng. w/ FM & A.I.
6	*	*	*		*	x		*	*	*	*		x	*	*				11		V2: Domain Eng. w/ DM & A.I.
7	*	*	x	*		*	x		*	*	*	*	x		*	*			12		V3: Domain Eng. w/ FM & A.E.I.
8	*	*	x	*		*	x		*	*	*	*		x	*	*			12		V4: Domain Eng. w/ DM & A.E.I.
9	*	*		*	x	*	x		*	*	*	*	x		*	*			12		V5: Domain Eng. w/ FM & A.I.O.
10	*	*		*	x	*	x		*	*	*	*		x	*	*			12		V6: Domain Eng. w/ DM & A.I.O.
11	*	*	x	*	x	*	x		*	*	*	*	x		*	*			13		V7: Domain Eng. w/ FM & A.E.I.O.
12	*	*	x	*	x	*	x		*	*	*	*		x	*	*			13		V8: Domain Eng. w/ DM & A.E.I.O.
13	*	*		*		*	x		*	*	*	*	x		*	*	x	x	14		V9: Domain&Application Eng. w/ FM & A.I.
14	*	*		*		*	x		*	*	*	*		x	*	*	x	x	14		V10: Domain&Application Eng. w/ DM & A.I.
15	*	*	x	*		*	x		*	*	*	*	x		*	*	x	x	15		V11: Domain&Application Eng. w/ FM & A.E.I.
16	*	*	x	*		*	x		*	*	*	*		x	*	*	x	x	15		V12: Domain&Application Eng. w/ DM & A.E.I.
17	*	*		*	x	x	x		*	*	*	*	x		*	*	x	x	15		V13: Domain&Application Eng. w/ FM & A.I.O.
18	*	*		*	x	x	x		*	*	*	*		x	*	*	x	x	15		V14: Domain&Application Eng. w/ DM & A.I.O.
19	*	*	x	*	x	*	x		*	*	*	*	x		*	*	x	x	16		V15: Domain&Application Eng. w/ FM & A.I.E.O.
20	*	*	x	*	x	*	x		*	*	*	*		x	*	*	x	x	16		V16: Domain&Application Eng. w/ DM & A.I.E.O.
FM = Feature Model DM = Decision Model A = Agent I = Interaction E = Environment O = Organization																					
(*) mandatory feature (x) optional selected featured (-) optional disable feature () optional unselected feature																					

For instance, *Configuration 1* is among the smallest MAS-PL method configurations and encompasses eleven features: *MAS component*, *Agent*, *Iteration*, *MAS Method*, *Tropos*, *SPL Engineering*, *Domain Engineering*, *Analysis*, *Feature Model*, *Design*, *Implementation*. On the other hand, *Configurations 19* and *20* encompass sixteen features and therefore are among the largest ones. The remaining possible configurations encompass sets ranging from twelve to fifteen features.

The derivation of a MAS-PL method variant consists of incrementally applying to an empty method the modifications specified by the delta modules attached to valid conditions in a given configuration. Such a derivation was achieved according to the following steps:

(i) Find all delta modules that shall be applied to the MAS-PL method variant, i.e. those modules attached to a condition evaluated to true for the given configuration. For example, for *Configuration 1* the delta module *D-TroposFMAAnalysis* (see Fig. 6 right) shall be selected, since its attached condition (*Tropos and FeatureModel*) is evaluated to true.

(ii) Generate the method variant by applying the modification proposed by the selected delta modules respecting the order defined by the list of delta modules.

All the twenty MAS-PL method variants presented in Table 1 (right) have been derived and are available in the Meduse for MAS-PL website. Two of these method variants are depicted in Fig. 7: *Tropos-PL Variant 1* and *Gaia-PL Variant 16*. The similarities among these variants are highlighted in red: the two method fragments belonging to Domain Design phase and the entire Domain Implementation phase.

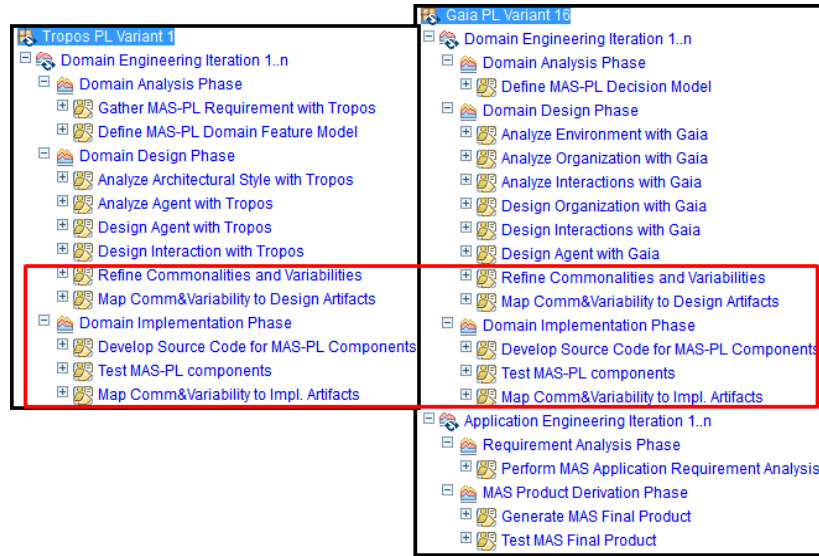


Fig. 7. Two MAS-PL method variants based on Tropos (left) and Gaia (right), highlighting in red the common fragments among them.

Tropos-PL Variant 1 is among the smallest variants: it encompasses only the Domain Engineering Iteration, which is composed of three phases and eleven activities. It proposes modeling MAS-PL variabilities using a feature model during Domain Analysis Phase. Moreover, it deals with agent and interaction development during Domain Design Phase. On the other hand, *Gaia-PL Variant 16* is among the largest variants: it covers both the Domain and Application Engineering Iterations and includes the four MAS components, i.e. agent, environment, interaction, and organization. It proposes

modeling MAS-PL variabilities using a Decision Model, and encompasses five phases that, in their turn, are composed of sixteen activities.

It should be observed that our approach can generate several extensions for the same AOSE method. For instance, the SPLE domain analysis may vary according to the formalism adopted to specify the variability model: feature model or decision model. Therefore, Fig. 8 illustrates two Tropos based MAS-PL method variants generated by our approach. In addition to the *Tropos-PL Variant 1*, we also generated the *Tropos-PL Variant 4* that uses the decision model instead of feature model and also covers all activities of the SPLE application engineering level.

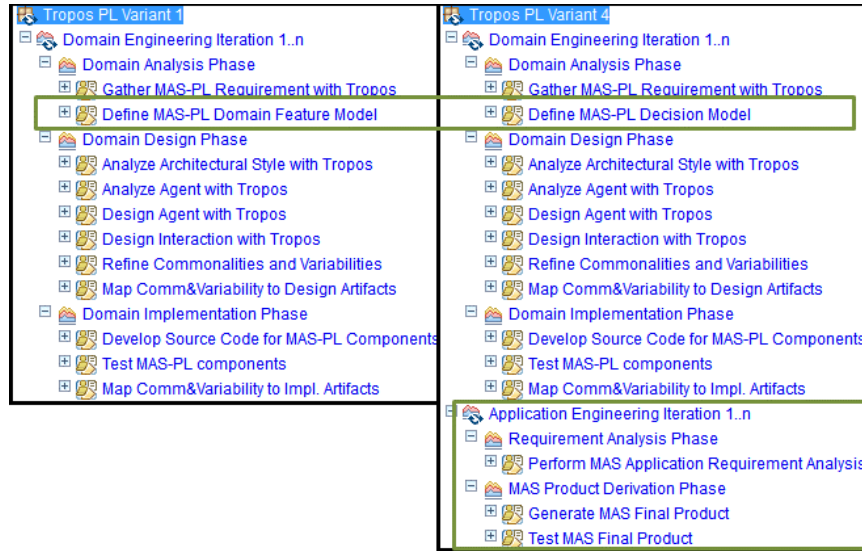


Fig. 8. Two MAS-PL method variants based on Tropos – the smallest (left) and the largest (right) - highlighting in green the variability among them.

5 Conclusion

This paper introduced Meduse for MAS-PL, an automated approach that aims at providing steady methodological support for MAS-PL development. Such a goal is achieved through the extension of existing AOSE methods to explicitly support MAS-PL development by integrating SPLE activities with them. This approach is based on the Meduse framework and results from a close collaboration between MAS and SPLE researchers. Indeed, and as illustrated by the presented case study, Meduse for MAS-PL approach involves a set of reusable method fragments that may be sourced from both AOSE methods and state-of-art SPLE development approaches.

We validate our approach by the generation of twenty MAS-PL methods that extend Gaia and Tropos. However, its principals can be applied to all AOSE methods. Some of the method variants generated during the case study were used by Master students in the University Pierre et Marie Curie (Paris 6). Those students apply these

MAS-PL methods to support the development of teams to the multiagent contest³. They consider the use of the proposed MAS-PL methods to generate several multiagent teams. This experience shows that the generated methods are promising. We plan to propose to several groups of students another project and some criteria to measure the quality of those methods.

6 Acknowledgement

Sara Casare was supported by CNPq (grant #233828/2014-1), Brazil.

7 Reference

1. Apel, S., Batory, D., Kästner, C., and Saake, G. 2013. Feature-Oriented Software Product Lines. Berlin: Springer.
2. Brandao, A., Boufedji, D., Ziadi, T., Guessoum, Z. 2015. Vers une approche d'ingénierie multiagent à base de ligne de produits logiciels. In : 23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15), Cépaduès, p. 49-58.
3. Bresciani, P.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. and Perini, A. 2004. Tropos: An Agent-Oriented Software Development Methodology. In: Journal of Autonomous Agents and Multi-Agent Systems, vol. 8, no 3, p. 203-236.
4. Brinkkemper, S. 1996. Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology*, vol. 38, no 4, p. 275-280.
5. Campbell, Grady H. Jr., Faulk, Stuart R., and Weiss, D. M. 1990. An introduction to Synthesis. Herndon, Virginia: Software Productivity Consortium.
6. Casare, S., Brandão, A. A., Guessoum, Z., and Sichman, J. S. 2014. Medee Method Framework: a situational approach for organization-centered MAS. *Autonomous agents and multi-agent systems*, vol. 28, no 3, p.430-473.
7. Casare, S., Ziadi, T., Brandão, A. A., Guessoum, Z. 2016. Meduse: an Approach for Tailoring Software Development Process. In: Proceedings of the 21th International Conference on Engineering of Complex Computer Systems, ICECCS 2016, p.197-200.
8. Clements, P., Northrop, L. 2001. Software Product Lines: Practices and Patterns, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
9. Cossentino, M. 2005. From requirements to code with the PASSI methodology. In: Henderson-Sellers, B. and Giorgini, P. (Eds.): *Agent-oriented Methodologies*, p.79–106, Idea Group Inc., Hershey, PA, USA.
10. Dehlinger, J., Lutz, R.R. 2005. A product-line requirements approach to safe reuse in multiagent systems. In: International Workshop on Software Engineering for Large-scale Multi-agent Systems, p.1–7.
11. Dehlinger, J., Lutz, R. R. 2011. Gaia-PL: a product line engineering approach for efficiently designing multiagent systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no 4, p. 17.

³ <https://multiagentcontest.org/>

12. Demazeau, Y. 1995. From interactions to collective behavior in agent-based systems. In: Proceedings of the First European Conference on Cognitive Science. Saint-Malo, p. 117-132.
13. Gomaa, H. 2004. Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley, USA.
14. Harmsen, A.F. 1997. *Situational Method Engineering*. Moret Ernst & Young.
15. Henderson-Sellers, B., Ralyté, J. 2010. Situational Method Engineering: State-of-the-Art Review. *J. UCS*, vol. 16, no 3, p. 424-478.
16. Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, A. S. 1990. *Feature-oriented domain analysis (FODA) feasibility study* (No. CMU/SEI-90-TR-21). Carnegie-Mellon University Pittsburgh Pa Software Engineering Inst.
17. OMG. 2008. Object Management Group. *Software & Systems Process Engineering Meta-Model Specification*, version 2.0. OMG document number: formal/2008-04-01. <http://www.omg.org/spec/SPEM/2.0/PDF>.
18. Nunes, I., Lucena, C. J. P., Cowan, D., Kulesza, U., Alencar, P., Nunes, C. 2011. Developing multi-agent system product lines: from requirements to code. *International Journal Agent-Oriented Software Engineering*, vol. 4, no 4, p. 353-3.
19. Peña, J., Hinchey, M.G., Resinas, M., Sterritt, R., Rash, J.L. 2007. Designing and managing evolving systems using a MAS product line approach. *Science of Computer Programming*, vol. 66, no 1, p. 71-86.
20. Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N. 2010. Delta-oriented programming of software product lines. In: International Conference on Software Product Lines. Springer Berlin Heidelberg, p. 77-91.
21. Schaefer, I., Damiani, F. 2011. Pure delta-oriented programming. In: *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*, ACM, p. 49-56.
22. Silva, V.; Choren R.; Lucena, C. 2008. MAS-ML: A Multi-Agent System Modelling Language. *International Journal of Agent-Oriented Software Engineering*, vol.2, no.4, p. 382-421.
23. SPC. 1993. *Software Productivity Consortium Services Corporation*. Technical Report SPC-92019-CMC. Reuse-Driven Software Processes Guidebook, Version 02.00.03.
24. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Program*, vol. 79, p.70-85.
25. Van Gurp, J., Bosch, J., Svahnberg, M. 2001. On the Notion of Variability in Software Product Lines. In: *Proceedings of the Working IEEE/IFIP Conference on Software Architecture* (WICSA '01). IEEE Computer Society, Washington, DC, USA.
26. Zambonelli, F., Jennings, N. R., Wooldridge, M. 2003. Developing multiagent systems: *The Gaia methodology*. *ACM Transaction on Software Engineering and Method*, vol. 12, no 3, p. 317-370.

Keyword Index

Affordances	17
Agent environments	158
Agent Oriented Programming	49
Agent Oriented Software Engineering	113
Agent societies	190
Agent-based Modeling and Simulation	97
Agent-Based Simulation	17
AmI	33
Argumentation Schemes	65
Assurance	1
Augmented reality	129
Augmented Worlds	129
Automated Planning	49
BDI	33
blockchain	81
Border Concepts	174
context-aware systems	33
contextual planning	33
Development Effort	97
Distributed environments	142
Endogenous environments	142
Environment Oriented Programming	49
Environment Programming	142
Explanation	1
Feature Model	113
Import-export agent societies as modules	190
Inter-societal agent systems	190
Internet of Things	158
JaCaMo framework	142
learning	33
Linking Concepts	174
MAS Dimensions	174
Method	174
method	206
Methodology	17

Model-driven Engineering	97
Modularization of agent societies	190
Multi-agent System Family	113
Multi-agent System Product Line	113
multi-agent systems	81
Multi-Agent Systems	65
multi-robot systems	81
multiagent systems	33, 206
Multiagent Systems	174
Openness	174
Organisational Model	65
Organizations as modules	190
Personal Assistant Agents	129
Related Concepts	174
REST	158
service-oriented computing	158
Software Engineering Reuse	113
software product line	206
Software Product Line Engineering	113
task allocation	81
Traffic Signal Control	97
Trust	1
Ubiquitous Computing	129
Validation	1
Variability	113
variability	206
Verification	1
Web architecture	158
Web of Things	158

Author Index

Akram, Muhammad Usman	49
Alves Franco Brandão, Anarosa	33, 206
Basegio, Tulio L.	81
Bazzan, Ana L. C.	97
Boissier, Olivier	158
Bordini, Rafael H.	65, 81
Boufedji, Dounia	113
Brandao, Anarosa	113
Casals, Arthur	33
Casare, Sara	206
Ciortea, Andrei	158
Croatti, Angelo	129
El Fallah Seghrouchni, Amal	33, 49
Florea, Adina Magda	158
Guerra-Hernández, Alejandro	142
Guessoum, Zahia	113, 206
Hashmi, Muhammad Adnan	49
Hubner, Jomi Fred	174
Klügl, Franziska	17
Limón Riaño, Héctor Xavier	142
Michelin, Regio A.	81
Mokhtari, Aicha	113
Nunes, Ingrid	97
Panisson, Alison R.	65
Ricci, Alessandro	129, 142
Rocha Costa, Antonio Carlos	190
Santos, Fernando	97
Timpf, Sabine	17
Uez, Daniela Maria	174

Winikoff, Michael	1
Ziadi, Tewfik	113, 206
Zimmermann, Antoine	158
Zorzo, Avelino F.	81