

Alma Mater Studiorum Università degli Studi di Bologna

Seconda Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

STUDYING AND ENGINEERING METHODOLOGICAL ASPECTS OF MULTI-AGENT SYSTEMS

Tesi in: Sistemi Multi-Agente LS

Elaborato di:
Antonio Cauzo

Relatore:
Prof. Andrea Omicini

Correlatore:
Prof. Tom Holvoet

Anno Accademico 2007/2008
Sessione II°

Key words

AGV Transportation System

Delegate MAS

A&A meta-model

Infrastructure

Integration

*Pensare ai loro occhi mi scalda il cuore
vedere i loro sguardi mi fa capire l'amore
scavare dentro me fa zampillare ardore*

*“Lo studio è noia, sacrificio ed alienazione”
Antonio Gramsci*

Abstract

From one hand a decentralized AGV Transportation System is a system which uses Automated Guided Vehicles (AGVs) in a warehouse to transport loads from one location to another. In such a system, AGVs are controlled by autonomous agents which make their own decisions, rather than being controlled by a central server. On the other hand, Delegate MAS represent an innovative approach to BDI agents which alleviates agent complexity, using the environment and its resources to obtain BDI functionality.

Can Delegate MAS approach fit AGV quality requirements?

To this end, Delegate MAS has to be applied to the AGV Transportation System. Several ways to do that are possible, but among them, taking into account the mapping of the Delegate MAS model into the A&A framework the Horizontal Integration way has been opted. In this thesis all the theoretical concepts related to this issues have been discussed.

Abstract

Da una parte un *AGV Transportation System*, sistema basato su *Automated Guided Vehicles* (AGVs) in grado di trasportare carichi tra le diverse posizioni nell'ambito di un' ambiente industriale. Per fronteggiare la complessità di tale sistema, l'astrazione di agente autonomo è utilizzata, dove ogni agente è in grado di controllare un AGV: gli agenti sono in grado di prendere le proprie decisioni senza che esista un server centralizzato in grado di controllare i vari agenti. Dall'altra parte un novativo approccio per agenti BDI, il Delegate MAS, si propone di alleviare la complessità degli agenti nel sistema *delegando* la loro delegazione all'ambiente, entità di prima di classe nella modellazione dei sistemi multiagente.

Ora, è possibile applicare il Delegate MAS all'AGV Transportation System in maniera da rendere più efficiente il funzionamento dell'intero sistema?

Per rispondere a questa domanda, un' Integrazione Orizzontale è stata presentata, tenendo in considerazione che il Delegate MAS è stato mappato nell' A&A meta-modello e concordemente al concetto di artefatto di coordinazione come servizio: in questo modo il Delegate MAS rappresenta un servizio per l' *AGV Transportation System*.

Contents

1	Introduction	1
1.1	The Structure of the Thesis	1
I	Background	3
2	Multiagent Systems	5
2.1	The Notion of Agent	5
2.2	Agent Societies	6
2.3	The Environment	7
3	Automated Guided Vehicles	9
3.1	AGV Transportation Systems	9
3.1.1	Main Functionalities	10
3.1.2	Quality Requirements	11
3.1.3	AGV Agents and Transport Agents	12
3.1.4	Virtual Environment	12
3.2	Architecture of an AGV Transportation System	13
3.2.1	Architecture-Centric Software Development	13
3.2.2	A Reference Architecture for Situated Multiagent Systems	15
3.3	Collaborating Components View Packets	17
3.3.1	Collaborating Components View Packet: Agent	17
3.3.2	Collaborating Components View Packet: Application Environment	20
3.3.3	The AGV Control System	24
3.3.4	The Transport Base System	31
3.3.5	Transport Assignment	33

4	Delegate MAS Overview	37
4.1	Basic agents	38
4.2	Ant agents	39
4.3	Conclusion	40
5	A&A Meta-Model Overview	41
5.1	Agents	42
5.2	Artifacts	43
5.2.1	Artifacts Taxonomy	43
5.3	MAS in the A&A meta-model	44
5.4	Summury	45
II	AGV Transportation System and Delegate MAS	47
6	AGV Transportation System using Delegate MAS model	49
6.1	Introduction	49
6.2	AGVs and transports	49
6.3	Basic Abstractions and Functionalities for AGV Transportation System	50
6.3.1	Basic Abstraction	50
6.3.2	Functionalities to Support AGV Transportation System	52
6.4	Delegate MAS for AGV Transportation System	52
III	Horizontal Integration	57
7	Integration: the theoretical issues	59
7.1	Introduction	59
7.2	Infrastructure Landscape	60
7.3	Some Important Notions on Environments	61
7.4	Infrastructures for Environment	66
7.4.1	MAS Infrastructures for Environment: TuCSon	67
7.4.2	MAS Infrastructures for Environment: Virtual Environments in the AGV Application	67
7.5	Towards Integration	70
7.5.1	Delegate MAS as Coordination Service?	70
7.5.2	Delegate MAS as an AGV Coordination Service	71
7.5.3	Adding the Delegate MAS Service	73

IV	Conclusion	79
8	Conclusion	81
8.1	Future Works	81

Chapter 1

Introduction

The notion of *coordination* in MAS is a key issue in the engineering of Complex Systems since it allow constructive interaction towards the system goal achievement.

Several ways to coordinate (society of) agents have been proposed. Among them, Delegate MAS can be found. This approach has been proposed by Holvoet et al. [28] to alleviate agent complexity in order to achieve system quality requirement and systems goal. To manage system complexity the main idea was to exploit the *environment as a coordination infrastructure for cognitive agents* [10], separating, this, the responsibilities in the system: obviously the *agents' computational burden* is less than putting the full complexity of coordination in them.

Delegate MAS have been developed for applications that perform real-time resource allocation and that supervise the execution of the tasks requiring these resources. Among them AGV Transportation System is an interesting application domain. AGV Transportation System uses Automated Guided Vehicles (AGVs) in a warehouse to transport loads from one location to another.

Besides, a mapping of the Delegate MAS model into the A&A framework and its mapping into an implementation oriented architecture built up on TuCSoN infrastructure [29] have been discussed.

The main goal of this thesis is to explore the way in which the (TuCSoN -based implemented) Delegate MAS service can be used by the AGV Transportation System and to argue all the theoretical issues stem from this matter.

1.1 The Structure of the Thesis

The thesis is structured as follows: Background, Horizontal Integration.

- Part I: *Background* - This part discusses in details the background context

of the thesis, that is based on several issues: the engineering of complex software systems through multiagent systems and agent societies, the reference architecture for the AGV Transportation System, the Delegate MAS model and the A&A meta-model.

- Part II: *AGV Transportation System and Delegate MAS* - This part presents the application of the Delegate MAS approach to the AGV Transportation System, underlying the main abstraction and issues in the model.
- Part III: *Horizontal Integration* - This part introduces the problem of adding the Delegate MAS to the existing AGV Transportation System, pointing out all the key issues needed to make Horizontal Integration conceptually consistent.

The thesis concludes by drawing some conclusions and identifying possible future works.

Part I

Background

Chapter 2

Multiagent Systems

The area of Multi-agent systems (MASs) brings together and draws on results, concepts and ideas of many disciplines including artificial intelligence (AI), distributed artificial intelligence (DAI), Parallel & Distributed Systems P&D, Mobile Computing, Programming Languages and Paradigms (PL), Software Engineering (SE), Robotics.

Generally speaking, MAS is an effective paradigm for modelling, understanding, and engineering complex systems, providing a basic set of high level abstractions that make it possible to directly capture and represent the main aspects of such complex systems, such as *interaction, multiplicity and decentralization of control, openness, dynamism* to cite a few.

A MAS can be characterized by three key abstractions: *agents, societies* and the *environment*. Agents are the basic active components of the systems, executing pro-actively and autonomously some kind of work. Agent societies are formed by sets of agents that suitably interact and communicate so as to do some kind of collective work, which requires the contribution - including knowledge, skills, activities - of multiple agents. In order to do their work agents typically need to exploit and affect the environment where they are situated.

2.1 The Notion of Agent

The first key concept of a MAS is that of “agent”. To give an agent definition is not easy because, actually now, there is no accepted notion of the term “agent”, and indeed there is a good deal of ongoing debate and controversy on this subject; we can find a lot of agent definitions, more or less convergent, coming from the different areas cited above.

Essentially there is a general consensus and agreement that autonomy is central

to the notion of agent. Part of the difficulty is that various attributes associated with agent are of different importance for different domains.

Nevertheless, some sort of definition is important. The most accepted and cited is that of [39]:

“an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.”

Various definitions and classifications have been given in literature for the agent concept, with different characterisations coming from different fields. From the different contexts, different acceptance of the agent abstraction have emerged, still sharing the basic issues of autonomy and situatedness. A synthesis is currently ongoing in the MAS community.

2.2 Agent Societies

When adopting an agent-oriented view, it soon becomes apparent that most application domain contexts require or involve multiple agents: to represent the decentralised nature of the problem, the multiple loci of control, the multiple perspectives or the competing interests. As obvious, MAS emphasise the multiplicity of the agents composing a system as a “society” of agents.

Interaction is therefore a main dimension of multi-agent systems. Agents need to interact either to fulfil their individual tasks or to take part to the collective processes which characterise the MAS as a whole. Interactions can vary from simple semantic interoperation through traditional client/server type of interactions, to rich social interactions (the ability to cooperate, coordinate, and negotiate in the same social context). More generally, we can say that different kind of interaction models can be adopted for enabling interaction and communication inside a MAS.

A first main distinction is between *direct* and *mediated* interaction. In the former case agents interact by directly exchanging informations; in the latter case, the media enabling agent interaction and communication, are explicitly modelled, and actually become fundamental in the engineering of the whole MAS. In this case the interaction and communication is not considered to take place directly between agents, but through these media as *first class entities* of the system. Such media could be the environment itself, or a different object.

2.3 The Environment

First of all, the notion of environment is what makes it possible to define agents' situatedness, defining a notion of locality, a model for the stimuli that can be perceived by the agents and also a model for the actions available to agents to affect their environment.

Accordingly, the environment plays a key role of *enabler* and *mediator* of agents' interaction: such a mediation role is crucial for realising forms of communication among agents than are complimentary to direct communication (which takes place directly between agents through some kind of agent communication language).

Chapter 3

Automated Guided Vehicles

Automatic Guided Vehicles (AGVs) are fully automated vehicles that are able to transport goods in an industrial environment. An AGV control system receives transport requests from a client system such as a warehouse managing system or machine operating software, and instructs AGVs to execute the transports.

The stream of transports that enter the transportation system is typically irregular and unpredictable. AGVs are provided with low-level control software connected to sensors and actuators to move safely through the warehouse environment [5]. It is fully automated, which means that it can perform all its actions without manual intervention.

An AGV is equipped with a battery as energy source, which must be recharged when the AGV runs out of energy. It also contains software, which enables it to communicate with its environment and perform actions. AGVs can communicate by means of a wireless LAN. To move around the warehouse or factory, AGVs must follow a number of fixed paths [6].

3.1 AGV Transportation Systems

Fig. gives a general overview of the AGV Transportation System and shows how the application software is allocated to computer hardware. The software consists of two types of subsystems: **Transport Base System** and **AGV Control System**.

In sections 3.3.3 and 3.3.4 will be deeply presented respectively *AGV Control System* and *Transport Base System*, which are the main part of the AGV Transportation Systems.

To introduce the two subsystems important issues such as functionalities, quality requirement and reference architecture, necessary for the development of the

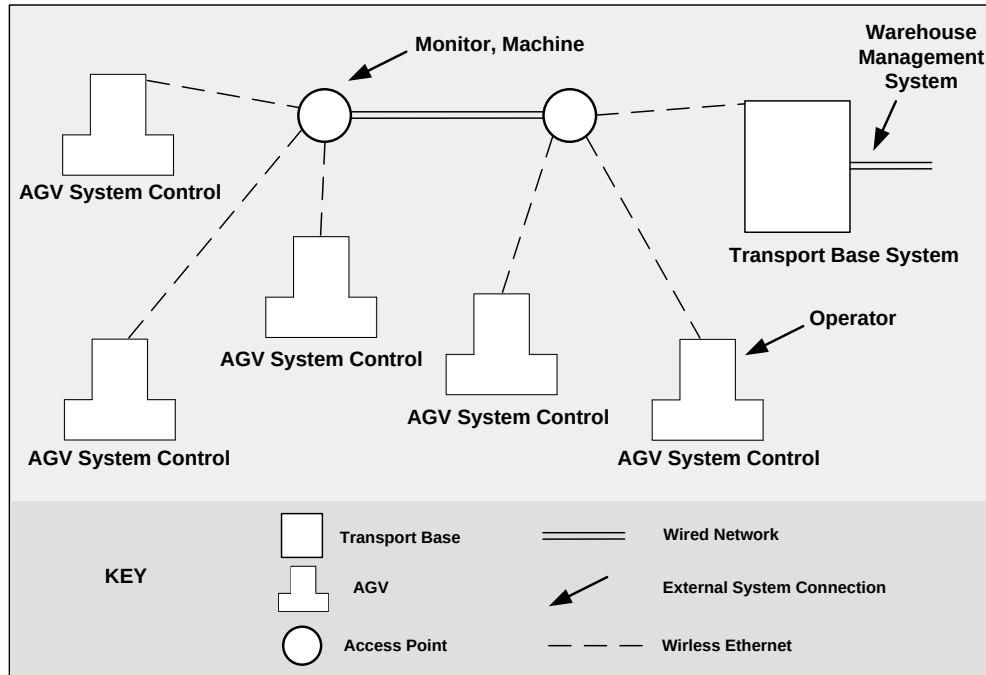


Figure 3.1: Deployment view of the AGV transportation System [5]

software system's architecture, need to be considered.

3.1.1 Main Functionalities

The main functionality AGV Transportation Systems has to perform is handling transports, i.e. moving loads from one place to another. There should be enough AGVs available to execute the transports that enter the system, i.e. the AGVs should be able to handle the load of the system. In order to execute transports, the main functionalities the system has to perform are:

Transport assignment transports are generated by clients and have to be assigned to AGVs that can execute them.

Routing AGVs must route efficiently through the layout of the warehouse when executing their transports.

Collision avoidance safety measures are necessary when AGVs cross the same intersection at the same moment and when AGVs pass each other on closely located paths.

Deadlock avoidance since AGVs are relatively constrained in their movements (they cannot divert from their path), the system must ensure that AGVs do not find themselves in a deadlock situation.

Battery Recharging to perform transport tasks, AGVs have to maintain their battery. AGVs can charge their battery at the available charging stations.

Parking when an AGV is idle, that means it does not have work to do, it can park at a free park location.

Avoiding Traffic since there are other AGVs in the environment, it might be required for an AGV to be able to adapt its route in order to avoid traffic.

3.1.2 Quality Requirements

The general quality requirements posed by the stakeholders of an AGV Transportation System are:

Performance is a major quality requirement of an AGV Transportation System. Customers expect that transports are handled efficiently by the system.

Configurability allows installations to be easily tailored to client-specific demands.

Robustness any manual interventions have to be restricted to a minimum, because they are time consuming and thus costly.

Other quality requirements are put forward by the evolution of the market and by the way to solve the problem. In fact, considering that agents are autonomous, in that they encapsulate control along with a criterion to govern [7], it is not so feasible to conceive the control system as centralized.

So, on the one hand Customers request for *self-managing systems*, which are able to autonomously adapt their behavior with changing circumstances, and on the other hand the autonomy of agents have led to a *decentralized AGV Transportation System*.

Flexibility refers to the system's ability to deal with dynamic operating conditions. A flexible control system allows an AGV that is assigned a transport and moves toward the load, to switch tasks along the way if a more interesting transport pops up, allows the AGV to anticipate possible difficulties and, anyway, it can be something that allow the system to handle particular situations autonomously, for example adapting its behaviour if some conditions in the environment are changed.

Openness of an AGV Transportation System refers to the ability of a system to deal with AGVs leaving and (re-)entering the system. AGVs can leave for maintenance or for charging, and return when done, can leave because of hardware or software failure, and later re-enter the system when repaired.

Summing up, the decentralization of the system had opened perspectives to improve *flexibility* and *openness*, which are high-ranking quality requirements for today AGV Transportation Systems but also to “confirm” the important general requirements as Performance, Configurability, Robustness and Scalability and to face in a suitable manner the design, develop and managing of complex software systems, as AGV Transportation Systems, typically distribute. One way to cope with these requirement could be to design a self-tuning system that adapts its parameters to improve performance.

3.1.3 AGV Agents and Transport Agents

In the system there are two types of agents: AGV Agents and Transport Agents.

AGV Agent is responsible for obtaining and handling transports, and ensuring that the AGV gets maintenance on time. As autonomous entity, AGV agent can take advantage of opportunities that occur in its vicinity, and that can enter and leave the system without interrupting the rest of the system. *Each AGV in the system is controlled by an AGV agent.*

Transport Agents is responsible for assigning the transport to an AGV and reporting the status and completion of the transport to the client that has requested the transport. Transport agents have to interact with AGV agents to find suitable AGVs to execute the transports and reside at a transport base, i.e. a dedicated computer located in the warehouse. *Each transport in the system is represented by a transport agent.*

3.1.4 Virtual Environment

Most application domain contexts, as AGV Transportation Systems, require or involve multiple and different agents: to represent the decentralised nature of the problem/domain, the multiple loci of control [8].

Ricci has argued in [9]

“The engineering of interaction is a fundamental issue for designing and building systems providing effectively their overall functionalities,

achieving the objectives designed by engineers. Coordination, in particular plays, a fundamental role in making all the parts work fruitfully together, toward the achievement of global objectives, besides the scope of the individual parts.”

So, it is straightforward that to achieve the system functionality, AGV agents and transport agents have to be coordinated. The choice was to exploit the prospective of the *environment as a coordination infrastructure for cognitive agents* [10]. In this mode it is possible to separate responsibilities in the system and to manage the complexity, starting from the agents that are, obviously, more simple than putting the full complexity of coordination in them.

Besides, the environment provides a medium for sharing information and mediating coordination among agents and, as a mediator, not only enables interaction but it also constrains it [10].

In Fig 3.3 an high-level model of an AGV Transportation System is illustrated.

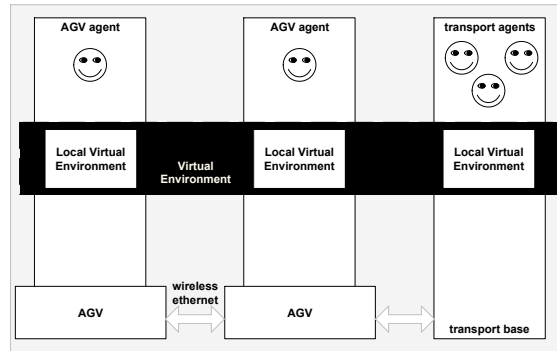


Figure 3.2: High-level model of an AGV Transportation System [6]

3.2 Architecture of an AGV Transportation System

3.2.1 Architecture-Centric Software Development

Since the early 1990s, *software architecture* has been subject of increasing interest in software engineering research and practice [11].

In [12], Bass et al. describe the software architecture of a program or computing system as

“the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships between them. The term also refers to documentation of a system’s software architecture.”

Another interesting definition is provided by Weyns [11]:

“A software architecture defines constraints on detailed design and implementation, it describes how the implementation must be divided into elements and how these elements must interact with one another to fulfil the system goals. On the other hand, a software architecture does not define an implementation, many fine-grained design decisions are left open by the architecture and must be completed by designers and developers.”

Why architecture is important to software systems? There are many reasons [11] [13]:

- software architecture provides a basis for creating mutual understanding and consensus about the software system
- software architecture is the manifestation of the earliest design decisions that have the most significant influence on system qualities
- software architecture is a reusable, transferable abstraction of a system
- software architecture constitutes a surveyable model for how a system is structured and works
- software architecture model is transferable to other systems with similar requirements and can promote large-scale reuse of design

How is possible to develop new software architectures? Weyns in [11] has proposed to apply a *reference architecture* for the design of a concrete software architecture.

A definition of reference architecture, by Weyns, can also be found in [11]

“Reference architectures [14, 12] go one step further in reuse of best practices in architectural design..Rather than being a rigid frame that restricts the architect’s creativity, the reference architecture is meant to serve as a valuable guidance for the architect that should be complemented with additional architectural approaches.”

3.2.2 A Reference Architecture for Situated Multiagent Systems

The design of a software architecture for multiagent system is a complex matter: on the one hand the role of the environment and the functionalities that can be assigned to it [10, 11] and on the other hand the agent functionalities, i.e. action-selection and perception mechanisms, communication etc.

For these reasons Weyns et al. have decided to adopt a *reference architecture*, whose goal is to support the architectural design of self-managing applications. This perspective is generally referred to as *situated agent systems*.

In a situated multi agent system (1) agents are considered as “simple” entities and (2) the environment is considered as *infrastructure for coordination*, mediating agent behaviour and interaction and providing services.

So, the environment is a “*first-class abstraction in multiagent systems*” [10] and, more precisely, it can be conceived as a *coordination infrastructure*.

Concrete motivations for the adoption of the reference architecture are argued by Weyns in [11]. The scope of this chapter is to give an overview of the main concepts. So, subsequently, a brief overview of the reference architecture defined by Weyns and colleagues is discussed.

This reference architecture integrates the *agent* and *environment* functionalities, mapping them onto an abstract system decomposition, i.e. software elements and relationships between them. It abstracts from the concrete deployment of the multiagent system application, which highly depends on the particular system requirements.

The software elements make up a system decomposition that cooperatively implement the agent and environment functionalities. This system decomposition, the reference architecture, provides a blueprint for instantiating target systems that share the common base of the reference architecture.

In a situated multiagent system two basic modules can be identified: **Agent** and **application environment**. Fig. 3.3 shows the top-level module decomposition of the reference architecture of situated MAS in which the main software units in the system, as discussed above, are displayed.

Agent is an autonomous computational entity: they encapsulate control along with a criterion to govern it [7]. Agents are situated in an environment which they can perceive and in which they can act and interact: in fact the overall application goals result from interaction among agents, rather than from sophisticated capabilities of individual agents.

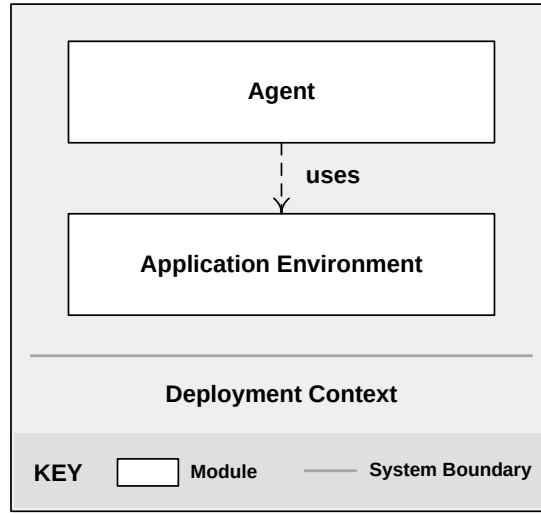


Figure 3.3: Top-level module decomposition of a situated MAS [15]

A concrete MAS application typically consists of heterogeneous agents, which have different capabilities and are assigned different application goals [15].

The **Application Environment** is the part of the environment that has to be designed for a concrete MAS application. The application environment enables agents to share information and to coordinate their behavior. As a mediator, the application environment not only *enables* perception, action and interaction, it also *constrains* them. Depending on the specific application requirements, different application environment types may be provided.

The **deployment context** consists of the given hardware and software and external resources such as sensors and actuators, a printer, a network, a database, a web service, etc. For a distributed application, the deployment context consists of multiple processors deployed on different nodes that are connected through a network.

Thus, the environment is conceived as consisting of two parts: the *application environment* and the *deployment context*.

3.3 Collaborating Components View Packets

In this section, the collaborating components view of the reference architecture is presented, in which the environment and agent functionalities are synthesized.

For Weyns et al., the collaborating components view represent an excellent vehicle to learn the runtime behavior of a situated MAS: in fact the view shows the *data flows* between runtime components and the interaction with data stores, and it specifies the functionalities of the various components in terms of *incoming* and *outgoing* data flows.

For clarity, the model, and so the presentation, is divided in two parts: situated agent and application environment.

3.3.1 Collaborating Components View Packet: Agent

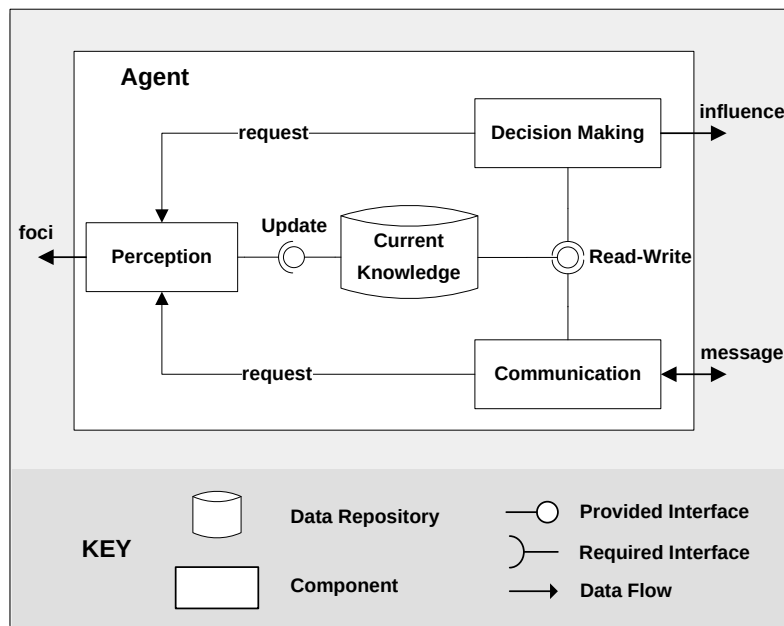


Figure 3.4: Collaborating Components of Agent [15]

As be seen in Fig. 3.4 the Agent component consists of three *subcomponents*: Perception, Decision Making and Communication. These components share the Current Knowledge repository.

The subcomponents in detail

Perception is responsible for collecting runtime information from the environment (application environment and deployment context). The perception component supports *selective perception* [16]. Selective perception enables an agent to direct its perception according to its current tasks. To direct its perception an agent selects a set of *foci* and *filters*:

Foci allow the agent to sense the environment only for specific types of information.

Sensing results in a representation of the sensed environment.

The perception module maps a representation of elements or resources in the environment to a percept, i.e. a description of the sensed environment in a form of data elements that can be used to update the agent's current knowledge. The selected set of *filters* further reduces the percept according to the criteria specified by the filters [15].

Decision Making is responsible for action selection. This action model, based on the influence-reaction model introduced in [17], distinguishes between *influences* that are produced by agents and are attempts to *modify the course of events in the environment*, and *reactions*, which result *in state changes in the environment*. As discussed in [18], the main responsibility of the decision making module has a twofold aspect: the *selection* of influences to realize the agent's tasks, and the *invocation* of the influences in the environment [18].

Weyns et al. have extended behavior-based action selection mechanisms with **roles** and **situated commitments** to enable situated agents to *set up collaborations* [19, 20, 21].

A *role* represents a coherent part of an agent's functionality in the context of an organization.

A *situated commitment* is an engagement of an agent to give preference to the actions of a particular role in the commitment.

Agents typically commit relative to one another in a collaboration. Roles and commitments have a *well-known name* that is part of the domain ontology and that is shared among the agents in the system.

Communication is responsible for communicative interactions with other agents. The agents, sharing the well-known name, are enabled to share information and to set up collaborations via message exchange.

The communication module processes incoming messages, and produces outgoing messages according to well-defined communication protocols [21]. A communication protocol specifies a set of possible sequences of messages.

The information exchanged via a message is encoded according to a shared communication language, which defines the format of the messages, i.e. the subsequent fields the message is composed of. A message includes many fields, i.e. fields with the identity of the sender and the identities of the addressees of the message, a field with the performative of the message, etc.

Communicative interactions among agents are based on an *ontology* that defines a shared vocabulary of words that agents use in messages. The ontology enables agents to refer *unambiguously* to concepts and relationships between concepts in the domain when exchanging messages.

Current Knowledge repository contains data that is shared among the data accessors. Data stored in the current knowledge repository refers to state perceived in the environment, to state related to the agent's roles and situated commitments, and possibly other internal state that is shared among the data accessors.

Fig. 3.4 shows the interconnections between the current knowledge repository and the internal components of the agent.

The current knowledge repository exposes two interfaces:

- The provided interface **Update** enables the perception component to update the agents knowledge according to the information derived from sensing the environment.
- The **Read-Write** interface enables the communication and decision making component to access and modify the agent's current knowledge.

Collaborations

The result of the coordination of Decision making and Communication is the key for the fulfillment of the overall behavior of the agent.

Decision making is responsible for selecting suitable influences to act in the environment.

Communication is responsible for the communicative interactions with other agents.

As can be seen in Fig. 3.4, decision making and communication typically request perceptions to update the agent's knowledge about the environment. This issues from two important topics:

- A dominant assumption is that the point of activity is to change the environment in a way that (presumably) leads to goal satisfaction [22].
- Communication requires that the information transmitted by one agent result in a state change of another [23].

Besides, decision making and communication also coordinate during the progress of a collaboration, typically established via message exchange.

From Fig. 3.4 another important issue stems: it is clearly visible the separation of functionality for coordination (via communication) from the functionality to perform actions to complete tasks. This approach has many advantages:

- clear design
- improved modifiability
- reusability
- parallelism, and so, difference pace of acting (consequently, adaptability and efficiency are improved)

3.3.2 Collaborating Components View Packet: Application Environment

As be seen in Fig. 3.5 the **Application Environment** component consists of seven *subcomponents* that share the **Sate** repository.

The subcomponents in detail

The **Representation Generator** provides the functionality to agents for *perceiving the environment*. When an agent senses the environment, the representation generator uses the current state of the application environment and possibly state collected from the deployment context to produce a representation for the agent. Agents' perception is subject to perception laws that provide a means to constrain perception [?].

Observation & Data Processing provides the functionality to *observe* the deployment context and *collect* data from other nodes in a distributed setting. The observation & data processing module translates observation requests into observation primitives that can be used to collect the requested data from the deployment context.

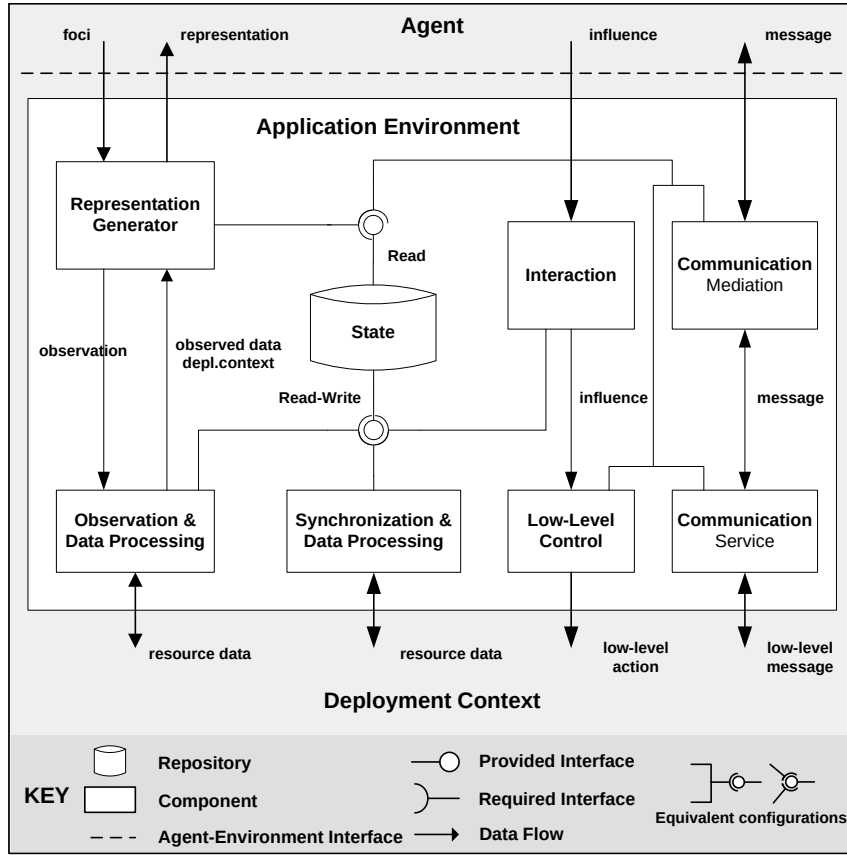


Figure 3.5: Collaborating Components of Application Environment [15]

Interaction is responsible to deal with agents' *influences* in the environment. Agents' influences can be divided in two classes: influences that attempt to *modify state of the application environment* and influences that attempt to *modify the state of resources* of the deployment context.

Action laws put restrictions on the influences invoked by the agents, representing domain specific constraints on agents' actions. Influences related to the deployment context are passed to the Low-Level Control module.

Low-Level Control bridges the gap between influences used by agents and the corresponding action primitives of the deployment context. Low-level control converts the influences invoked by the agents into low-level action primitives in the deployment context. This decouples the interaction module from the details of the deployment context.

The **Communication Mediation** mediates the *communicative interactions* among agents. It is responsible for collecting messages; it provides the necessary infrastructure to buffer messages, and it delivers messages to the appropriate agents. Communication mediation regulates the exchange of messages between agents according to a set of applicable communication laws [21].

To actually transmit the messages, communication mediation makes use of the Communication Service module.

Communication Service provides that *actual infrastructure* to transmit messages. In this mode the designer can express the communicative interactions between agents, abstracting from the underlying communication technology (as, for example, FIPA ACL message [25]).

However, to actually transmit such messages, they have to be translated into low-level primitives of a communication infrastructure provided by the deployment context.

Depending on the specific application requirements, the communication service may provide specific communication services to enable the exchange of messages in a distributed setting, such as white and yellow page services (as, for example, Jade [26]).

Synchronization & Data Processing synchronizes state of the application environment with state of resources in the deployment context as well as state of the application environment on different nodes.

State updates may relate to dynamics in the deployment context (i.e. a network abstraction maintained in the state of the application environment) and dynamics of state in the application environment (i.e. evaporation of digital pheromones) that happens independently of agents or the deployment context.

Synchronization & data processing converts the resource data observed from the deployment context into a format that can be used to update the state of the application environment. Such conversion typically includes a processing or integration of collected resource data.

The **State** repository contains data that is shared between the components of the application environment. Data stored in the state repository typically includes an *abstraction of the deployment context* together with additional *state related to the application environment*.

The state repository may also include agent-specific data, such as the agents' identities, the positions of the agents, and tags used for coordination purposes.

Collaborations

The representation generator *collects perception requests* from the agents and generates representations according to the given *foci*. Representation generator collects the required state from the state repository, and optionally it requests observation & data processing to collect additional data from the deployment context and possibly state of other nodes. State collection is subject to the perception laws. Observation & data processing returns the observed data to representation generator that generates a representation that is returned to the requesting agent.

Interaction *collects* the concurrently invoked *influences* of agents and converts them into *operations*. The execution of operations is subject to the action laws of the system. Operations that attempt to modify state of the application environment are *immediately executed* by the interaction component. Operations that attempt to modify state of the deployment context are *forwarded to low-level control* that converts the operations into low-level interactions in the deployment context.

Communication mediation handles the *communicative interactions* among agents. The component collects the messages sent by agents, applies the communication laws, and subsequently passes the messages to the communication service. This latter component converts the messages directed to agents on other nodes into low-level interactions that are transmitted via the deployment context.

Furthermore, communication service collects low-level messages from the deployment context, converts the messages into a format understandable for the agents, and forward the messages to communication mediation that delivers the messages to the appropriate agents. Messages directed to agents that are located at the same node are *directly transferred* to the appropriate agents.

Synchronization & data processing performs its tasks *independently* of other components of the application environment. To synchronize the state of the application environment in a distributed setting, synchronization & data processing components on different nodes have to coordinate according to the requirements of the application at hand.

In this section an overview of the reference architecture introduced by Weyns et al. has been given. Starting from the reference architecture and the system requirements, presented in section 3.1.2, in the next section will be argued:

- the architectural design of the decentralized AGV control system

- how the software architecture of the AGV control system relates to the reference architecture

zooming in on the evaluation of the software architecture.

3.3.3 The AGV Control System

This section gives an overview of a software architecture for an AGV Transportation System. Most of the architecture described in this chapter is inspired by the architectural design discussed in section 3.2.2.

The architecture discussed in this chapter is considered as background architecture: in fact it is used as a foundation to further build upon, in an attempt to enable Delegate MAS service integration.

In order to separate the responsibilities, a layered approach is used to decompose the AGV control system. In this mode every layer has its own responsibilities and allows tasks to be handled at different levels of abstraction: this allows *better management of complexity*.

The AGV control system is composed of three layers: the AGV agent layer, the local virtual environment layer and the ObjectPlaces middleware & E'nsor layer (Fig. 3.6).

The **AGV agent** is responsible for *controlling* an AGV. It makes the main decisions of the AGV control system.

The **virtual environment** serves as a *medium* for AGV agents and transport agents to exchange information and to coordinate their behavior. Moreover, the local virtual environment:

- offers *high-level primitives* to agents to act and perceive the environment, and communicate with other agents: this enables them to share information and coordinate their behavior.
- the virtual environment serves as a *suitable abstraction* that shields the agents from low-level issues.

The **middleware** enables communication with software systems on other nodes (other AGV control systems and the transport base system at the lowest level), by handling the actual transmission and receiving of low-level messages, providing a means to *synchronize* the state of the local virtual environment with the state of local virtual environments on neighboring nodes. Moreover it allows the steering of

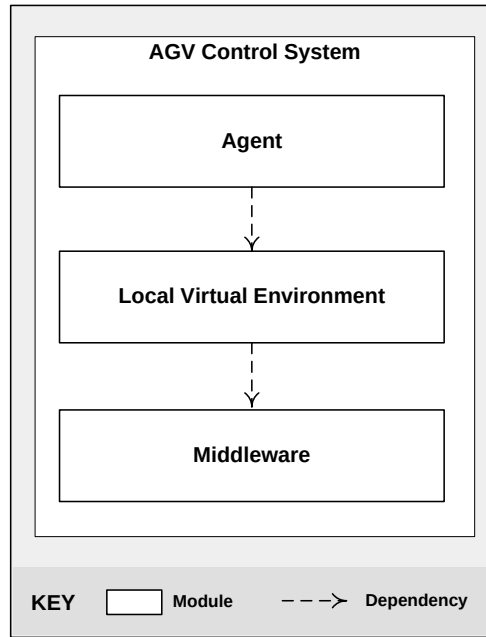


Figure 3.6: Module Decomposition View of the AGV Control System [6]

the AGV vehicle, with use of actuator commands and regularly receives data from sensors (such as the AGV's current position).

In the next sections, the collaborating components view for the three different layers are presented, in which AGV functionalities are synthesized. The AGV Control System issues from the reference architecture presented in section 3.2.2: only the concepts related to the AGV Transportation System will be presented.

AGV Agent Layer

Fig. 3.7 shows a collaborating components view of the AGV agent: it shows the software system as a set of interacting runtime components that use a set of shared data repositories to realize the required system functionalities [11].

As can be see observing Fig. 3.7, a new software component has been added to the already existing design to handle AGV system requirements.

The **current knowledge** is a shared data repository which contains all kinds of *states* the agent uses for decision making and communication. This knowledge repository contains *static* and *dynamic* states. An example of a static state is the layout. The dynamic states are typically states which are collected through

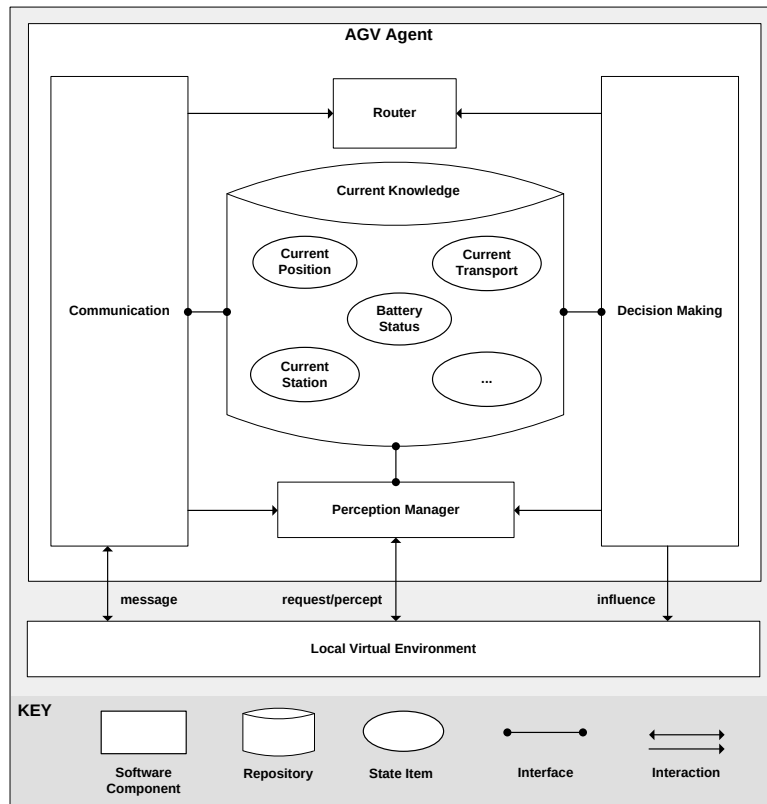


Figure 3.7: Collaborating Components View of the AGV Agent [6]

the observation of the environment. Examples of dynamic states are the current position of the AGV, the battery status, the current station the AGV resides at, etc. These types of dynamic states can be updated with information *from the local virtual environment*.

The **perception manager** enables the AGV agent to *sense the local virtual environment* according to the perception requests of communication and decision making, and to *update the agent's current knowledge* accordingly. AGV agents use different foci (3.3.1) to sense the state of the local virtual environment that represents state in the physical environment (e.g., the positions of neighboring AGVs) and state that relates to virtual representations (e.g., fields that are used for transport assignment).

The **communication component** handles the AGV agent's communicative interactions with other AGV agents or transport agents. In the developed system, this is only used for transport assignment, but it is straight-forward to add other components which use high-level communication as well. A collaborating components view for the agent's communication component can be found in [6].

The **decision making** component handles the AGV's actions. Depending on the situation the AGV agent is currently in, the component decides which influence (3.3.1) needs to be executed. This decision is then forwarded to the local virtual environment.

Due to the complexity of decision making of the AGV agent, it has been modeled as a hybrid architecture that combines a blackboard pattern with sequential processing.

A collaborating components view for the agent's decision making component can be seen in Fig. 3.8. The current knowledge repository serves as blackboard, while the *action controller* coordinates the selection of a suitable action. After *job selection* (execute transport, charge battery, etc.), the *action selection* component selects an action at a fairly high-level (move, pick, etc.).

Action generation transforms the selected high-level action into a concrete action (e.g., move(Segment x)).

Collision avoidance and *deadlock avoidance* are responsible to lock the trajectory associated with the selected action. As soon as the trajectory is locked, the selected action is passed to the execution component that invokes the action in the local virtual environment. If during the subsequent phases of decision making the selected action can not be executed, feedback is sent to the action controller that will inform the appropriate component to revise its decision.

The **router** provides an interface to find routes. This may be used by both the decision making and the communication component. The decision making component will need this to find the best (possibly the shortest) route to a certain destination. The communication component may need the router to be able to inform a particular transport agent how far the AGV is located from a specific load. The transport agent can hereby decide on which AGV is best suitable to handle a particular transport task.

The Local Virtual Environment Layer

Fig. 3.9 shows the software architecture of the local virtual environment, its main components and how they interact with each other.

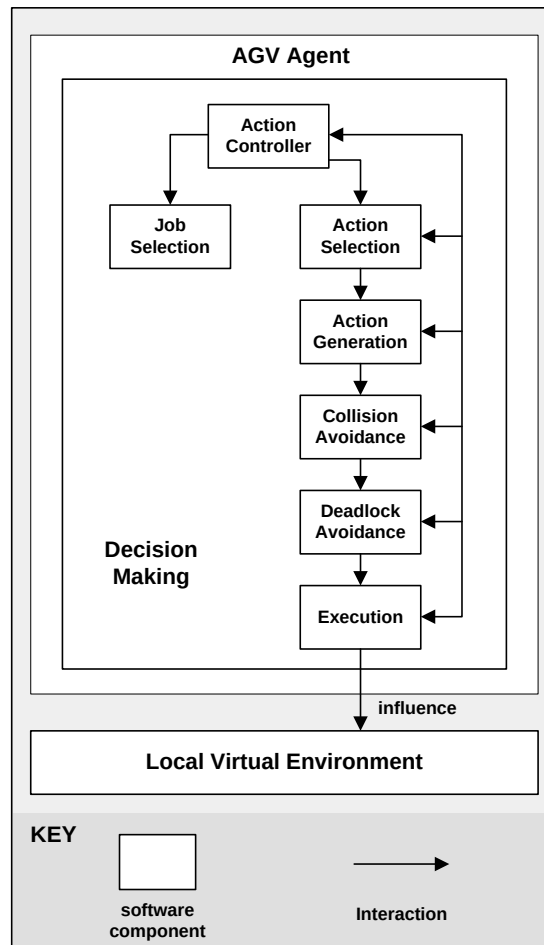


Figure 3.8: Collaborating Components View of the Decision Making Component of the AGV Agent [6]

As can be seen, the general structure of the local virtual environment is very similar to the structure of the agent. Just as in the AGV agent layer, there are two main components: the *communication* and *action handling* component, which are both linked to a shared data repository, called the *current state*.

The **current state** repository keeps track of the states of the AGV. Some of the states in this repository are states which are only needed by components at the AGV agent layer, such as the battery status. Other states are only used in the local virtual environment, such as hulls and flags.

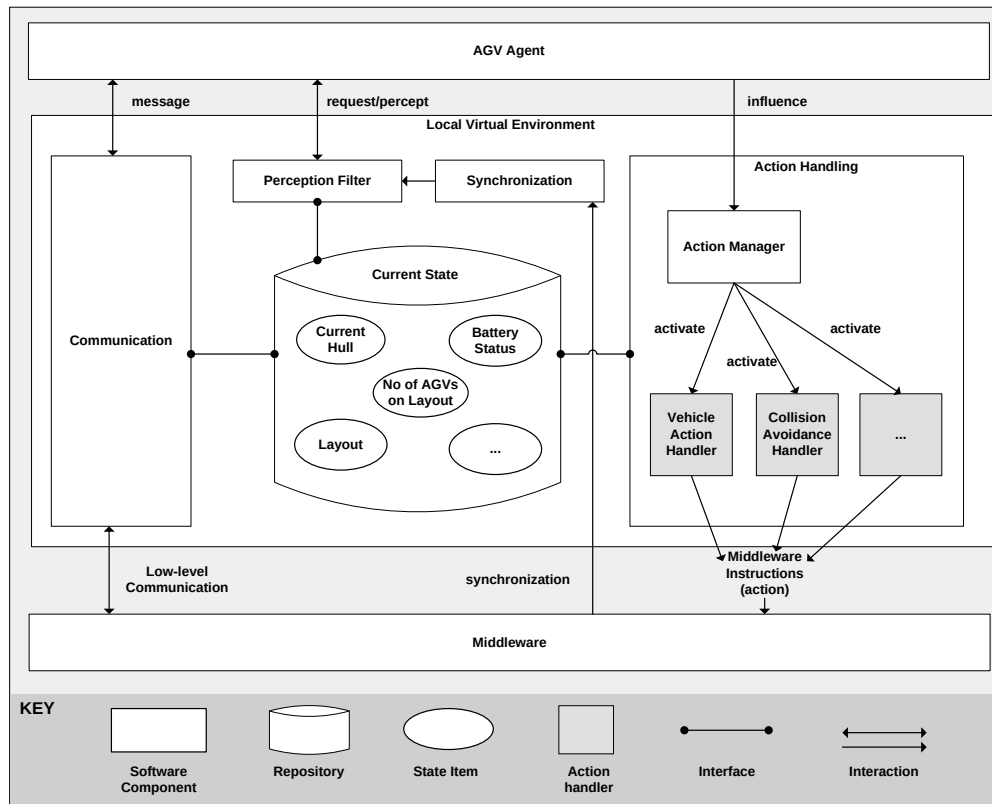


Figure 3.9: Collaborating Components View of the Local Virtual Environment of the AGV [6]

The **perception filler** handles perceptions at the local virtual environment. When an action handler at the AGV agent layer needs to know the current position of an AGV, it forwards a “current position perception”, and the perception filler fills this perception with the current position value before returning the perception.

The **synchronization** component has been added to the already existing design. Its mainly behaviour is to systematically executes a synchronization process with the AGV agent layer. The component uses the perception filler to synchronize specific data of the current state (of the local virtual environment) with the current knowledge of the AGV agent.

The **action handling** component translates the influences forwarded by the AGV agent layer to concrete actions to be performed by the local virtual environment

and/or the AGV vehicle and it contains two main component types: the *action manager* and a number of *action handlers*. This software component has extended the reference architecture to fit with the AGV system requirements.

The *action manager* receives influences from the agent layer, where the decision making component decides which influence needs to be executed, based on the situation the AGV is currently in, and forwards these influences to the *appropriate action handlers*.

An *action handler* handles influences. It may appropriately change states in the current state, and create actuator commands, which can be forwarded to the middleware. Examples of such action handlers are the vehicle actions handler and the collision avoidance handler, which are used for steering the AGV vehicle and avoiding collisions respectively.

The Middleware Layer

Fig. 3.10 shows the software architecture of the middleware and its main components. This software component has extended the reference architecture to fit with the AGV system requirements.

The **current state** data repository contains data concerning the state of the AGV. This is data which is received from sensors. Further, the current state also contains data which is used by low-level protocols performed at the middleware layer, such as a messaging protocol for node-locking.

The **synchronization** component handles systematic synchronization with the local virtual environment. Whenever a state, which requires synchronization, in the middleware changes, its corresponding state in the local virtual environment is updated immediately. This ensures that the data repository of the local virtual environment always contains *consistent* information about the whole system.

The **action handling** component receives instructions from the local virtual environment to either execute a particular protocol, or perform a physical action:

- in the former case, the action handling component updates the current state with appropriate data, so that the communication component can further perform the actual execution of the protocol
- in the latter case, the action handling component sends low-level actuator commands to the AGV vehicle

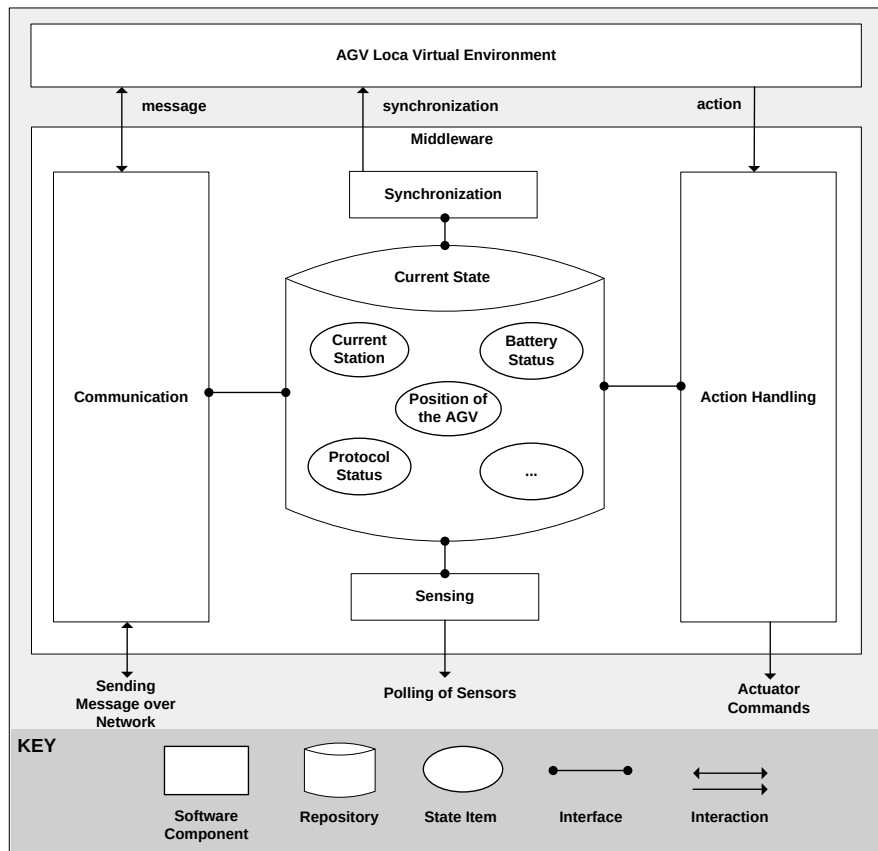


Figure 3.10: Collaborating Components View of the Middleware of the AGV [6]

The **sensing** component continuously polls the sensors of the AGV vehicle and updates the current state of the middleware with sensory data.

3.3.4 The Transport Base System

The **Transport Base System** provides the software to manage transports in the AGV system. Moreover:

- provides a *public interface* that enable an external client to request transports, allowing also external monitor systems to observe the status of the AGV Transportation System (see Fig. 3.1)
- handles the communication with the warehouse management system and external client systems (see Fig. 3.1)

- *receives* transport requests and *assigns* the transports to suitable AGVs, reporting the status and completion of the transports to the *warehouse management system*
- executes on the transport base, i.e. a stationary computer

Fig. 4.5 shows the decomposition of the transport base system.

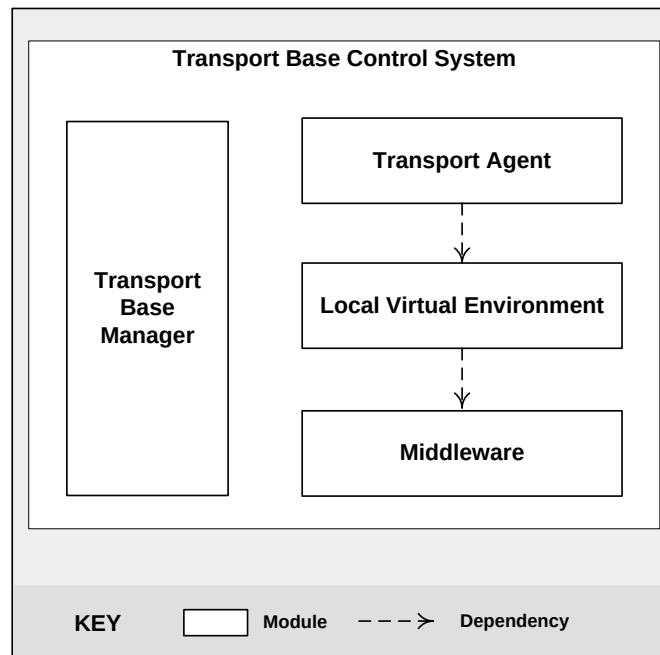


Figure 3.11: Collaborating Components View of the Transport Base System [6]

The **transport base manager**, as well as handles the communication, manage the new transport requests from the client, creating a new transport agent for each of them (every transport agent represents a transport task).

The **transport agent** responsibility is to assign the transport task to an appropriate AGV. It is important to underline that:

Physically a transport agent is located at the transport base manager where it is created

Logically the transport agent is situated at the location where the load of its concerning transport is to be picked up by an AGV

The **local virtual environment**, as in the AGV Control System, shields the transport agents from low-level communication issues. Instead, contrary to the AGV control system, there is only one local virtual environment for the transport base system: the same local virtual environment is thus used by the transport base manager and the different transport agents.

The local virtual environment has, also, to maintain a list of all transport agents, to forward message dedicated to a particular transport agent and, finally, to enable communication with AGV agents by translating incoming and outgoing messages (essential when a transport agent has find a suitable AGV agent to carry out a particular transport task).

The **middleware** basically handles the actual transmission and receiving of low-level messages. This enables communication between transport agents and AGV agents at the lowest level.

3.3.5 Transport Assignment

As presented in the previous section, one of the main responsibility of the Transport Base System is to assign a new transport task to an *appropriate* AGV. This responsibility is handled from the transport agent. The problem is: in which mode an AGV can be considered as being appropriate for the task assignment? And how it can be handled in an adaptive way? In fact, if from one side clients expect that transports are handled efficiently by the system, on the other side openness and flexibility require *adaptive transport assignment*.

Holvoet et al. have so developed two different approaches for adaptive transport assignment: **FiTA** (Field-based Transport Assignment) and **DynCNET** (Dynamic Contract Net Protocol):

- the former employs computational fields in the virtual environment to guide AGVs to loads
- the latter is an extension of the well-known contract net protocol (**CNET** [27]), with “Dyn” referring to support for dynamic task assignment.

FiTA

The basic idea of field-based transport assignment is to let each idle AGV follow the gradient of a field that guides it toward a load that has to be transported. The AGV agents continuously reconsider the situation of the environment and transport

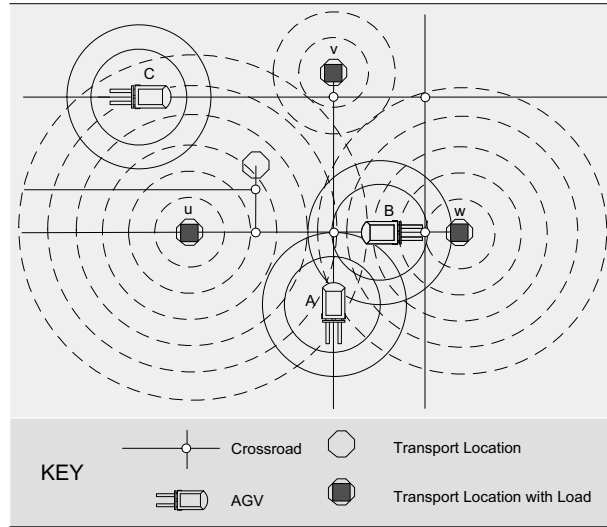


Figure 3.12: Transport Assignment with FiTA

assignment is delayed until the load is picked, which benefits the *flexibility of the system* (in Fig. 3.12 a FiTA scenario can be seen).

Coordination between the AGV agents working in the system is achieved mainly through the field emission and the interaction between AGV agents and transport agents: both emit fields in the local virtual environment. However, to avoid multiple AGVs driving toward the same transport, AGVs emit also *repulsive* fields. AGV agents, to reach pick locations of transports, have to combine received fields and to follow the gradient of the combined fields. Fields have some properties:

- have a certain range and contain information about the source agent
- AGV fields have a fixed range, while the range of transport fields is variable
- fields are refreshed at regular times, according to a predefined refresh rate

When an AGV agent perceives fields, it stores the data contained in these fields in a field-cache that consists of a number of cache entries (each cache entry contains the identity of the received field, the most recent data contained in that field and the up-to-date frequency).

For example, in Fig. 3.12 the field-cache of AGV A will consist of three entries, one for transport *u*, one for transport *w*, and one for AGV *B*.

Finally, an AGV agent constructs a calculated-field, which is a combination of the received fields, to decide in which direction to drive from a node and follows

the calculated-field in the direction of the smallest value. This can be considered as following the gradient of the calculated-field downhill.

Why this method can guarantee an adaptive task assignment?

Final transport assignment is delayed until an AGV actually reaches a pick location and picks up the load. This allows agents to adapt the assignment of transports while the AGVs drive toward loads. By delaying transport assignment, FiTA can cope with *changing circumstances* that arise.

In [5] the software architecture of AVG agents that are involved in the generation of move actions in FiTA is argued.

DynCNET

The DynCNET protocol describes communicative interactions among AGV agents and transport agents to fulfill the transport assignment: one important point of this protocol is that agents dynamically can switch the assignment of tasks.

DynCNET is an $m \times n$ protocol: an initiator that offers a task can interact with m participants, i.e. the candidate agents that can execute the task. On the other hand, each participant can interact with n initiators that offer tasks.

In the AGV Transportation System, an initiator corresponds with a *transport agent* that represents a task (i.e., a transport) in the system and the participant corresponds with an *AGV agent* that can execute tasks.

The dotted circles in Fig. 3.13 show the current areas of interest of AGV *A* (top) and transport *x* (bottom). For transport *x*, there are currently two candidate AGVs to execute the transports: *F* and *G* (AGV *E* is delivering a load). For AGV *A* on the other hand, there are three possible transports to execute: *u*, *v*, and *w*.

Due to the dynamics in the system, the set of candidate tasks (initiators) and agents that can execute a task (participants) can change over time.

The *default protocol* consists of four steps (these four steps are basically the same as in the standard CNET protocol [27]) :

- the initiator sends a call for proposals
- the participants respond with proposals
- initiator notifies the provisional winner
- and finally, the selected participant informs the initiator that the task is started

Why this method can guarantee an adaptive task assignment?

The flexibility of DynCNET is based on the possible revision of the provisional task

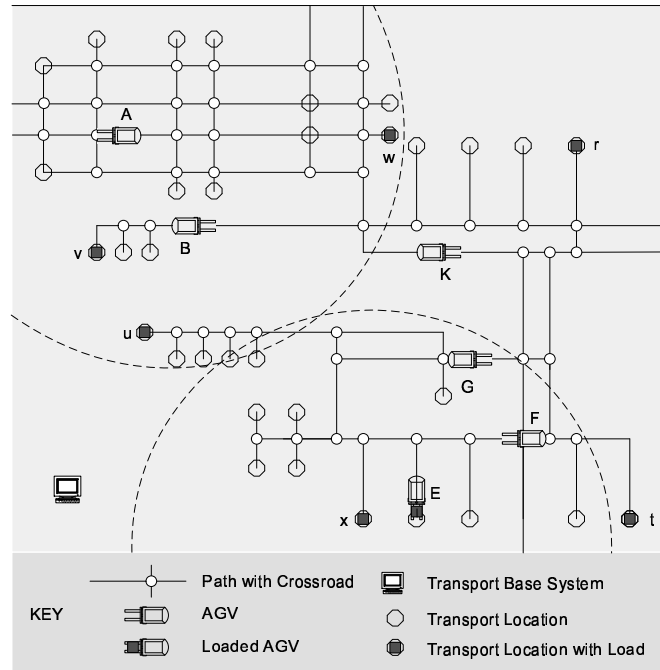


Figure 3.13: Transport Assignment with DynCNET

assignment between the third and fourth step of the protocol (in [5] is explained in detail how agents can switch tasks when the conditions in the environment change).

Chapter 4

Delegate MAS Overview

In the previous chapter FiTA and DynCNET have been present to resolve the task assignment problem for the AGV Transportation System. These approaches have one crucial property: in both agents are directly involved in the task assignment.

FiTA AGV agents have to combine received fields and to follow the gradient of the combined fields: the agent action generation need to be extended

DynCNET being basically based on contract net protocol, AGV agents have to communicate with transport agent(through different message types): his communication module have to be extended to integrate the protocol properties

In this section an innovative approach to BDI agents which alleviates agent complexity will be discussed, presented by Holvoet et al. [28]: this particular BDI based approach is called “Delegate MASs”.

To manage system complexity the main idea was to exploit the *environment as a coordination infrastructure for cognitive agents* [10], separating, this way, the responsibilities in the system: obviously the *agents’ computational burden* is less than putting the full complexity of coordination in them.

So, rather than creating and maintaining complex world and agent models themselves, the agents **delegate** this to the environment through light-weight agents, that are more simple than A&A agent. They can be issued either by:

- *resources* for building and maintaining information on the environment
- *task agents* in order to explore the options on behalf of the agents and to coordinate their intentions

Light-weight agents can alleviate agents responsibilities *exploring* the environment and *bringing* relevant information back to their responsible task agent, evaluating optional paths and putting the intentions of their task agent as information in the environment.

This allows delegate MASs of different agents to coordinate by aligning or adapting the information in the environment according to their own tasks.

4.1 Basic agents

Delegate MASs is thought for the family of MAS applications characterized by their large scale in terms of number of agents and physical distribution, such as manufacturing control, traffic control and web service coordination. For this reason two BDI agents are defined: Task agents and Resource agents.

Resource agents In the real world, such as a factory, a resource is a production means, such as machines, conveyors, tool holders, material storage or even personnel and is able to perform some kind of operations (capabilities). A *resource agent* is an abstraction of the resources and lives in a virtual world that represents the underlying system. The resource itself offers processing capacity and functionality to the resource agent.

Task agents In a factory there can be one or more tasks to be performed correctly and in time. A *task agent* represents and controls a task in the coordination and control application, and resides on a (physical or virtual) mobile entity in the environment.

A task agent is responsible for performing its task by guiding its mobile unit through the environment, and communicating with resource agents in order to perform operations on the unit. Every task agent is aware of the goal of its task, and has available the schemes or plans that can be followed in order to reach this goal.

Besides, task agents need to explore the *feasible paths* that correspond to their task schemes. A feasible path describes a sequences of resources that can be reached by following this path and corresponds to a task scheme if following this path routes the task agent along the resources that are necessary to reach its final goal. A task agent needs to consider all possible schemes (i.e. sequences of operations) which can bring the current task toward its goal, and match these plans with the feasibility information.

4.2 Ant agents

As presented above, Delegate MASs consist of light-weight agents which are issued either by resource agents or by task agents in order to reach the goal of its tasks. To present and to understand well light-weight agents' nature is important to underline some relevant Delegate MASs' aspects.

In fact, this pattern, to some degree is inspired by collective intelligence systems and some of its properties are the theoretical basis for Delegate MASs:

- the computation is *distributed*
- agents equipped with *simple computational capabilities*
- *multiple interactions* among agents through *direct* and *indirect* interactions (one agent modifies the environment and the other agent reacts to the changed environment)

In particular, foraging behavior of ants is interesting because:

- ants can find the shortest path between the nest and a food source
- while walking ants deposit a substance called *pheromone* on the ground
- they choose with higher probability paths that are marked by stronger pheromone concentrations
- *cooperative interaction* leads to the emergence of short(est) paths

In this sense light-weight agents are “ant agents”: these ant agents are issued by basic and reside in a virtual software environment which reflects the application environment, and in which ant agents can navigate.

There are three types of ant agents: Feasibility ants, Exploration ants, Intention ants. Each of them has a particular activity to perform autonomously, yet which information they distribute or how the information they gather is used, is the responsibility of the issuing basic agents. Particularly:

Feasibility Ants form a delegate MAS that is issued by resource agents. Their purpose is to roam the environment and, at each node they pass, drop information on feasible paths that start from this node. This way Task Agents are alleviated from direct topological information handling as they can retrieve it from the environment [29].

Exploration Ants form a delegate MAS that explore a feasible route through the underlying system, evaluating it. They alleviate task Agents by managing path exploration [29].

Intension Ants form a delegate MAS that is issued by task agents. Their purpose is to propagate the intention of task agents through the environment. They alleviate task Agents by managing intension propagation [29].

4.3 Conclusion

After this considerations and taking into account the application domain of Delegate MASs, some interesting principles can be recognized:

- direct communications between BDI agents is substituted with *environment mediated ones*
- place relevant information as signs in the environment ensuring that locally available data informs about remote system properties (direct and indirect interactions)
- limit the lifetime of this information (evaporation) and refresh the information as long as it remains valid - this allows the system to cope with changes and disturbances.

Besides, resource agents and task agents have other responsibilities:

- resource agents need to be able to make *schedules* based on requests from task agents
- resource agents must also be able to answer **what-if questions**: a task agent may ask a resource agent when and according to what quality standards a particular operation could be performed if the task would arrive at a future time. This allows task agents to evaluate the total time to completion and the expected quality of the finished task for a particular plan
- delegate MASs can be used for obtaining feasibility information, exploration and propagation of intentions toward the resource agents

Chapter 5

A&A Meta-Model Overview

Agents are not alone in multi-agent systems (MAS). They interact with other agents, and with the surrounding environment as well: a shared view exists in the literature that agent, society, and environment can be taken as the three basic categories to interpret and model the structure and dynamics of non-trivial MAS. However, whereas the study of the direct intercourse between agents appears as quite a well-explored subject in MAS literature, grounded on solid philosophical foundations like Searle's philosophy of human language, the study of interaction within agent societies and of agent interaction with and through the environment apparently still lacks a well-grounded conceptual foundation.

In the work of Omicini et al., MAS are modelled and engineered based on two fundamental computational abstractions, *agents* and *artifacts*. Recent work on the A&A framework has introduced workspaces as a new abstraction in the meta-model.

Agents are the *autonomous*, (pro-)active entities that encapsulate control, and are in charge of the goals/tasks that altogether define and determine the whole MAS behaviour

Artifacts are instead the *passive, reactive* entities in charge of the services and functions that make individual agents work together in a MAS, and shape agent environment according to the MAS needs. According to social / psychological theories like Activity Theory (AT), artifacts plays a fundamental role in the context of human organizations for supporting cooperative work and, more generally, complex collaboration activities. Artifacts are either physical or cognitive tools that are shared and exploited by the collectivity of individuals for achieving individual as well as global objectives

Workspaces as conceptual containers of agents and artifacts, useful for defining

the topology for the environment and providing a way to define a notion of locality.

The resulting agents & artifacts (A&A) meta-model promotes the modelling and engineering of agent societies and MAS environment as *first-class entities*.

5.1 Agents

The A&A meta-model recognises autonomy as the fundamental defining feature for agents. From a computational viewpoint, autonomy means that agents encapsulate (the thread of) control. So, control does *not flow through agent boundaries*: agents never give up control, nor are they controlled by anything-unless they deliberate to do so, of course. Correspondingly, agents provide no interfaces for being used, and cannot be invoked or controlled-agents can say “no”. Only data (information, knowledge) crosses agent boundaries.

As a result, MAS are to be viewed as a multiplicity of distinct *loci* of control, interacting with each other by exchanging information.

Literally, the etymology of the word “agent”-from Latin “agens”-means “the one who acts”. So, autonomous agents are intrinsically *active entities*: more precisely, they are proactive, since they encapsulate control, and self-govern their own course of action-where pro-activity simply means “making something happen”, rather than waiting for something to happen.

As a further consequence, the agent concept should by definition come equipped with a notion of agent action-that is, a conceptual model for agent actions is needed to provide a coherent notion of agent.

In the context of a MAS, an agent action has two potential targets: either *another agent*, or the world around the agents of the MAS, the *environment*. According to the A&A model agent actions can be roughly classified as: (i) internal actions, (ii) communicative actions, involving direct communications with one or more agents as the only means to directly affects their state, and (iii) pragmatical actions, as interactions within the environment.

Finally, any “ground” model of action is strictly coupled with the context where the action takes place: so, the model of action depends on the context where agents act: in this sense, autonomous agents are essentially *situated* entities, since any agent is strictly coupled with the environment where it lives and (inter)acts.

Other “fundamental” features have often been attributed to agents, like intelligence, mobility, or the ability to learn.

5.2 Artifacts

An A&A artifact is a computational entity aimed at the use by A&A agents.

So, as their first characterisation, artifacts are to be used by agents. From use, many other features stem, which are either essential or desirable, but need not be used as definitory ones.

First of all, artifacts are *not autonomous*, unlike agents. Since they are designed to serve some agent's purpose, artifacts are not to follow their own course of action. An artifact is a tool in the "hands" of agents, and does not need to be self-governed. Instead, artifacts are governed by agent's use: as such, artifacts are (computationally) *reactive*, that is, they are reactive in terms of control. Artifacts behave in response to agent use, and their behaviour just needs to emerge when they are used by an agent.

Then, every artifact *has a function*. Artifacts are *designed for use*: being aimed at the agent's use, artifacts are designed to serve some purpose, and built as such. At the time of their design, they are then associated to their function by the artifact designer.

Since they are aimed at being used, artifacts are the primary target/means of agent's action. Also, their function is expressed in terms of change to the environment, that is, what the artifact actually does when used by an agent. So, the artifact's model, structure and behaviour are expressed in terms of agent's actions, and *their effects on their environment*, which makes artifacts intrinsically situated. Finally, since they are situated, and also structurally reactive in computational terms, artifacts are easy to be thought as reactive to changes in the environment.

In order to be used, artifacts should *make operations available* to agents. Operations change an artifact's state, make it behave and produce the desired effects on the environment. So, either explicitly or implicitly, an artifact exhibits its interface, as the collection of the operations they made available to agents for use.

Finally, in order to promote their use by intelligent agents, artifact function should be available to agents, and understandable by them. Also, artifact behaviour should be *predictable*, so that agents could effectively exploit artifacts in their practical reasoning, and actually improve their ability to act successfully. So, *transparency* and *predictability* of behaviour are indeed desirable features of A&A artifacts.

5.2.1 Artifacts Taxonomy

Many sorts of different artifacts can populate an agent environment, providing a number of different services, embodying a variety of diverse models, technologies and tools, and addressing a wide range of application issues. In order to help

engineers while working in shaping the agent environment, the A&A meta-model introduces an artifact taxonomy, focusing on the mediation role of the artifact as classification criterion. More precisely, artifacts are classified based on the sort of the (non-artifact) MAS entities they are meant to tie together. According to this criterion individual artifacts, social artifacts, and resource artifacts are identified [38].

Individual artifacts are artifacts exploited by one agent, and mediate between an individual agent and the environment. In general, individual artifacts are not directly affected by the activity of other agents, but can, through linkability, interact with other artifacts in the MAS.

Social artifacts are instead artifacts exploited by more than one agent, and mediate between two or more agents in a MAS. In general, social artifacts typically provide MASs with a service which is in the first place meant to achieve a social goal of the MAS, rather than an individual agent goal.

Environmental artifacts are artifacts that conceptually wrap external resources, and mediate between agents of a MAS and the external resources. In principle, resource artifacts can be conceived as a means to raise external MAS resources up to the agent cognitive level. In fact, they can equip external resources with a usage interface, operating instructions, and a service description, and realise their task by dynamically mapping high-level agent interactions upon lower-level interactions.

In the end, individual, social and resource artifacts can be used as the basis for building the glue keeping agents together in a MAS, and for structuring the environment where agents live and interact: altogether, they can be taken as the conceptual, *layered* foundation for artifact design in MAS engineering.

5.3 MAS in the A&A meta-model

An A&A MAS is a *computational system* made of agents and artifacts (this definition is not fully complete, since it lacks the essential concept of workspace).

In the overall, the behaviour of A&A MAS results from the interaction of autonomous, self-governing entities (agents) and reactive, functional entities (artifacts). Since both agents & artifacts are *situated computational* entities, A&A MAS are *situated computational systems*: a MAS is always immersed within an environment, and cannot be conceived/modelled/designed in a separate way with respect to its environment.

The space of admissible interactions within a MAS in the A&A meta-model is generated by the four sorts of interaction potentially occurring between the two fundamental entities composing MAS:

communication agents *speak* with agents

operation agents *use* artifacts

composition artifacts *link* with artifacts

presentation artifacts *manifest* to agents

As a result, interaction in the A&A meta-model amounts to the four sorts above, plus the interaction with MAS environment-which, depending on the desired/required level of abstraction, we may attribute to either individual agents/artifacts, or to the MAS as a whole.

5.4 Summury

In this chapters, a background architecture of a decentralized AGV Transportation System and an overview of Delegate MAS and A&A meta-model was described. This issues serve as a basis of our architectural design for Horizontal Integration, which is thoroughly discussed in the next chapter.

Part II

AGV Transportation System and Delegate MAS

Chapter 6

AGV Transportation System using Delegate MAS model

6.1 Introduction

In chapter 4, Delegate MAS approach has been presented as an innovative approach to BDI agents which alleviates agent complexity, exploiting the *environment as a coordination infrastructure for cognitive agents* [10], separating, this way, the responsibilities in the system: obviously the *agents' computational burden* is less than putting the full complexity of coordination in them.

In [40] a Delegate MAS approach for Anticipatory Vehicle Routing has been proposed in which the prototype application has explained, initial test results have been showed. Finally, the Delegate MAS approach has been compared with related work, this way identifying benefits and novelty of this recent approach.

The objective of this part is to define a Delegate MAS model for the AGV Transportation System, similarly to the Anticipatory Vehicle Routing case.

6.2 AGVs and transports

An **AGV** is a unmanned, battery powered vehicle, capable of *picking up* a load, *driving* around and *dropping* it. AGVs can communicate by means of a wireless LAN. The AGVs are restricted to follow a predefined layout in the warehouse environment. The **layout** is defined in a map that consists of a network of stations and segments that are accessible by the AGVs. AGVs can leave and re-enter the system to charge their battery or for maintenance [37].

AGVs are equipped with low-level control software that uses sensors and ac-

tuators to stay on track, turn, pick and drop a load, and determine the current position.

A **transport** represents a task to move a load from a pick location to a drop location. Transports are generated by an external system, typically a warehouse management system. The stream of transports that enter the system is typically *irregular* and *unpredictable*.

Each transport has a priority that depends on the importance of the task and that increases over time to avoid starvation. Transports in an AGV Transportation System are characterized by delayed commencement, i.e., an AGV first has to drive to a load before it can pick the load and transport it to the destination.

While driving toward the load all kinds of changes in the system may occur. New tasks may enter the system that are *more suitable* for the AGV to execute, new AGVs may become available that are more suitable to perform the task, etc.

6.3 Basic Abstractions and Functionalities for AGV Transportation System

We now introduce the basis abstractions for a software architecture for AGV Transportation System and the required functionalities of the AGVs to have the task assigned will be discussed.

6.3.1 Basic Abstraction

The software architecture of Delegate MAS approach is based on three basic abstractions: task agent, infrastructure agent and virtual environment. This model need to be adapted to the AGV Transportation System. In fact:

- task agent is an AGV agent
- infrastructure agent is a resource agent (RA)
- transport agent (TA) is associated to a task in the system
- virtual environment reflects the real AGV environment

AGV Agent

Each AGV is equipped with a computer containing an AGV control system. The AGV control system contains a single *AGV agent* that is responsible for obtaining and executing transports, and ensuring that the AGV gets maintenance on time

(such as charging the AGV's battery). As such, an AGV becomes an autonomous entity that can take advantage of opportunities that occur in its vicinity, and that can enter/leave the system without interrupting the rest of the system.

Two important observations:

- the AGV agents are created at *application start up*
- the AGV agent is not disabled after the task execution

Virtual Environment

Since the physical environment of AGVs is very restricted, it offers little opportunities for agents to use the environment for coordination purposes. The local virtual environment (LVE) is a software entity that represents and maintains relevant state of the physical environment and state that is used by the AGV agent to exchange information with other agents and to coordinate their behavior.

The VE reflects the real AGV environment: the physical environment is mapped onto a *graph representation*. The nodes of the graph represent *station* (any point on which the AGV can stand still, and from which other segments start or arrive in) and *location* (a station where an AGV can perform an action - picking or dropping a packet) while *segments* the paths on which AGVs can drive.

Resource Agent

At every station and location a *resource agent* (RA) is associated. AGV agents can book road elements via the RAs. Bookings must be refreshed regularly to maintain the reservation. In particular RA provide the *transit function* to the AGV agent, informing the agents if they can pass for the specific location or the station represented by RAs.

Moreover, considering that *transport agents* (TAs) are not created when a new task is inserted in the system but is already associated to all the locations, when a new task is inserted in the system the correspondent TA is switch on. This way the scenario is:

- at every station a RA is associated
- at every location or a RA is associated (no task) or a RA and a TA if in that node a new task has been added

6.3.2 Functionalities to Support AGV Transportation System

The main functionalities AGV agents require for task assignment are (i) functionality to *explore relevant paths* in the environment, and (ii) based on the evaluation of the possible alternatives, functionality to *choose one path* and record it as an intention.

AGV agents need to **explore** the relevant paths to reach their destination. This results in different alternatives the AGV agent can select to achieve its goal. Based on the evaluation of the alternatives, the AGV agent chooses one path as its **intention**. Adopting an intention obviously has implications on the possibilities of other AGVs. Therefore, the AGV agent needs to communicate with the corresponding resource agents and inform them of the time they intend to occupy the corresponding segment, i.e. the information agents need to book these reservations.

6.4 Delegate MAS for AGV Transportation System

The use of Delegate MASs allow exploration of alternatives and propagation of intentions toward the infrastructure agents. In classical approaches AGV agents and resource agents use direct communication protocols to achieve the functionalities for AGV Transportation System.

Delegate MASs instead, being inspired by food foraging in ant colonies [37], exploit the environment to achieve the AGV System functionalities. Food foraging ants execute a simple procedure: in absence of any signs in the environment, ants walk around randomly in search for food. When an ant discovers a food source, it drops a smelling substance - a *pheromone* - on its way back to the nest while carrying some of the food. This pheromone trail evaporates over time, and disappears if no other ant deposits fresh pheromone. Another ant in search for food will use pheromones in the environment as a source of information to direct its own behavior. Pheromones indicate *possible routes* to a food source, and ants are urged by instinct to follow this trail to the food source. When the ant finds the food source, it will return with food, while depositing pheromone itself. When the ant discovers that the food source is exhausted, it starts a random search for food again. As the pheromone trail is no longer reinforced, it disappears over time [37].

These simple behavioral patterns result in an emergent behavior of the ant colony that is highly ordered and effective at foraging food while being *robust against dynamics in the environment*. An important property of this type of collective behavior

is that **global information is made available locally**.

This leads to the following engineering principles: *(i)* place relevant information as signs in the environment *(ii)* limit the lifetime of this information - evaporation - and refresh the information as long as it remains valid (allows the system to cope with *changes* and *disturbances*).

Delegate MAS model takes into account these principle, specifying three types of light-weight agents 4.2, which represent three different Delegate MASs and which share a common virtual environment for indirect communication: Feasibility ants (FAs), Exploration ants (EAs) and Intention ants (IAs). It is important to notice that the ants of the delegate MASs can explore alternative paths orders of *magnitudes faster* as AGV can drive.

Feasibility ants An AGV agent generates FAs at a *certain frequency* which explore the whole graph from the AGV's current position in order *(i)* to discover the feasible path in the graph, *(ii)* to have informations about all the nodes (location and station) in the graph and *(iii)* for each nodes to have informations from RAs if the AGV can transit in that node and if there is a task. This way FAs reports back all the feasible path of the graph and the status concerning the inner nodes.

Exploration ants An AGV agent, beyond FAs, generates exploration ants at a *certain frequency* which have to explore the feasible paths from the AGV's current position (according to the previous explanation, a feasible path gives also informations about its inner nodes and their status - *transit* and *task* issues). These exploration ants are scouts that each explore a feasible paths through the underlying system and evaluate this route. To evaluate a path, an exploration ant follows the path through the virtual environment, and interacts with the resource agents at the different nodes. If a node is passable, two possibilities can be outlined:

- **if there is no task** an AGV agent asks to the RA connected to the node “*what* the travel time would be *if* I would arrive at a particular time in the near future?”. The answer by RA is a *delay time* value
- **if there is a task** an AGV agent asks to the RA connected to the node “*can* I execute the task *if* I would arrive at a particular time in the near future?”. The answer by RA is a *boolean* value

The EA collects this information, and then proceeds till all the nodes in the path have been evaluated. When it happens, the exploration ant returns and reports back to its base, i.e. the AGV agent that issued the exploration ant.

Figure 6.1 illustrates the exploration process for a simplified scenario. The AGV agent on the right hand side creates three exploration ants for scouting feasible paths. These ants bring back information on the current information about transition and task in the nodes. This information are refreshed regularly as exploration ants are sent out regularly.

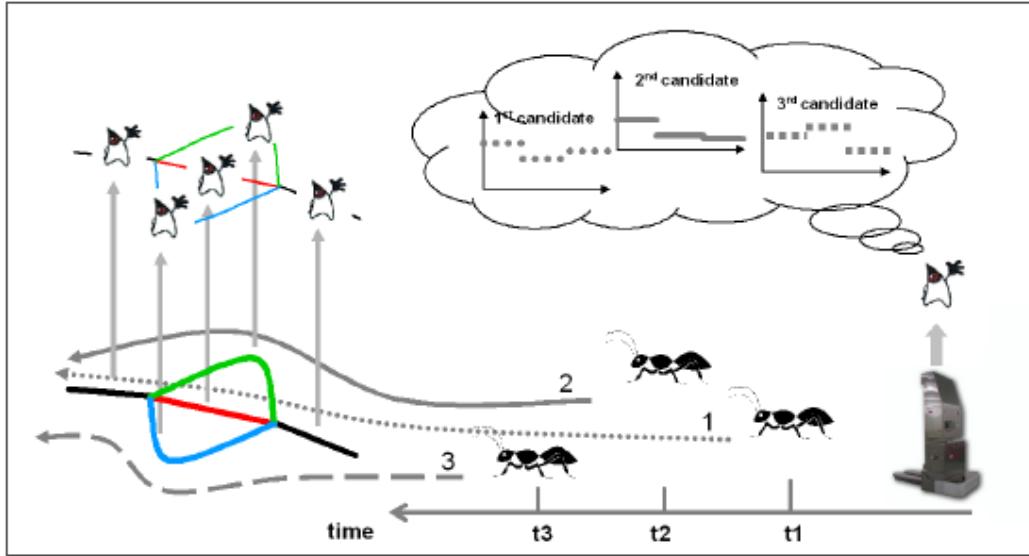


Figure 6.1: Exploration ants, issued by a AGV agent, scout feasible paths

Intention ants When an AGV agent has constructed a set of valid paths to follow, the vehicle agent selects one candidate path to become its intention. This way, AGV agents generate IAs, which propagate the intentions of AGV agents through the virtual environment. The criteria used for the path selection depend on quality factors, based on:

- the whole time to travel a path (sum of the delay time)
- number of available tasks

An AGV agent try to select the path with the smaller travel time and the maximum number of tasks. After the decision, the AGV agent creates AIs, at a *certain frequency*, to inform the resource agents that are involved in this intended path.

The IAs follow the selected path, and virtually travel the route of their selected candidate solution. On their virtual journey, the IAs acquire travel and

queuing informations from RAs on their path.

Changes which occurred since the exploration, immediately become visible when these RAs provide the information. In contrast to the EAs, IAs inform RAs that their AGV agent is likely to visit them at the estimated time. In this way, IAs make a booking on the road segment. As a consequence, RAs are able to predict what is the AGV agent that want to pass through a node (where a task can be there or can not be there), more accurately to their visiting EAs and IAs. Similar as EAs, IAs report back to their AGV agent to inform the agent about the quality and schedule of the bookings.

The intention information at the AGV agent - the booking - evaporates. *Evaporation of bookings* reflects truth maintenance of the intention information in the environment. Vehicle agents must create intention ants to refresh their intention at a frequency that is sufficiently high to maintain their bookings for the task to execute. While refreshing, a vehicle agent observes the evolution of the task landscape through the EAs issuing: in fact, when an AGV agent decide for a particular path, beyond IAs, EAs are issued as well to trigger changes (one or more nodes are not passable or a new inserted task can be better - it is nearest).

This estimate is compared to the estimates of the candidate solutions that are found and refreshed by the exploration agents. When the quality of the estimation of the current intention drops significantly below the qualities of other candidate solutions, the vehicle agent may revise its intention.

Figure 6.2 illustrates an example scenario. Based on the information that was gathered by the three EAs, the vehicle agent decides that the path explored by the last EA (which explored candidate path 3) fits the goal best. To confirm this and align this decision with the involved road elements, an IA walks path 3 and makes bookings on its way.

It is important to underline that FAs, EAs and IAs are issued in parallel.

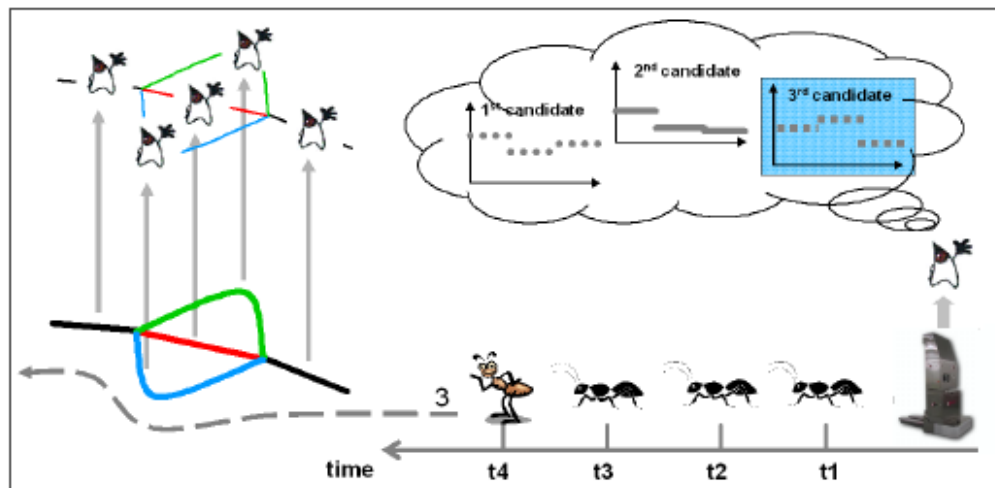


Figure 6.2: Intention ant, issued by an AGV agent at t_4 , communicate the intention of the vehicle agent through the environment

Part III

Horizontal Integration

Chapter 7

Integration: the theoretical issues

7.1 Introduction

Differently from FiTA and DynCNET approaches, in which both agents are directly involved in the task assignment, Delegate MAS can represent an innovative way to resolve the task assignment problem for the AGV Transportation System: the main idea is to exploit the *environment as a coordination infrastructure for cognitive agents* [10], separating, this way, the responsibilities in the system.

As reported in chapter 3, AGV Transportation System incorporate several functionalities like transport assignment, routing, collision avoidance, deadlock avoidance, traffic avoidance, battery recharging and more, all needed to fulfill main functionality AGV Transportation System has to perform and some quality requirement (see section 3.1.2). In particular, to achieve transport assignment functionality, FiTA and DynCNET has been developed in the system (which supports flexible integration of functionalities).

But, the question is: Does Delegate MAS represent an innovative approach, really improving the whole system behaviour and alleviating agent (AGV) complexity?

The best way to answer this question is to apply Delegate MAS in AGV Transportation System, comparing Delegate MAS with DynCNET and FiTA through some tests (like in [37]): in this mode is possible to know exactly if the new approach meets or not expectations.

According to the mapping of the Delegate MAS model into the A&A framework and its mapping into an implementation oriented architecture built up on TuCSon infrastructure [29], several ways to apply Delegate MAS in AGV Transportation System are possible, among them:

- completely re-design TuCSoN tuple centers to fit the AGV Transportation System reference architecture, including integration of topological aspects
- the support of the AGV Transportation System for flexible integration could suggest to add the Delegate MAS approach as a functionality that can be plugged-in and plugged-out
- to build a bridge between TuCSoN tuple centers and the AGV Virtual Environment, allowing tuple center of a TuCSoN node to be directly accessed from the AGV application

I opted for the latter approach, which (i) requires **no re-design** of the two infrastructures, this way resulting in more theoretical effort rather than practical one, and (ii) **promotes heterogeneity**, since adding the Delegate MAS as a service for the AGV AGV Transportation System, paves the way for integration of other kinds of services that could be interesting and useful for the AGV application (e.g. in this case, it allows AGV (BDI) agents to transparently interact and coordinate with any other agents accessing TuCSoN).

In next chapters, all the issues related to this approach will be discussed.

7.2 Infrastructure Landscape

Infrastructures, in Multi-Agent Systems, play a key role as support for complex agent (social) interplay, expressed in terms of services, convention and knowledge. Their services, indeed, typically cover critical system issues, and provide features that individual system components could not afford to provide or obtain elsewhere [1].

Moreover, infrastructures model and shape the agent environment from a twofold viewpoint: the **agents' viewpoint** in which agents access the environment through expressive runtime abstractions, possibly provided as services by the infrastructure and the **engineers' viewpoint** from which infrastructures are the most suitable place where to embed elements of control (constraints, coordination laws, norms) for open, unpredictable systems.

In this way, it is possible to conceive MAS engineering process as a continuum: from design down to development and deployment and thank to which abstractions used at design time should not disappear.

As in the human society people are different in their culture, skillness, in what they think and in what they are but can live well-integrated in the society, interacting with other people through well-known social artifact, as languages, laws (it is not always

so simple..) that we can conceive as being regulated by a sort of infrastructure depending on the role (e.g. State Rights) also in the Multi-Agent Systems this concept can fit.

Firstly, it is important to focus on the *agent interaction* notion for which different models, technologies and infrastructures have been defined: also for this issue infrastructure is important, providing abstractions and technologies to make it possible interaction among components.

Not only the pure interaction within the MAS boundaries is interesting but also *among MAS*: in fact MASs are applied in a wide set of real world application and are specialized to solve certain types of problems.

According with this point of view, it is easy to introduce the **Horizontal Integration** concept thanks to the role of the infrastructure: in fact Horizontal Integration means MAS Integration that is possible through Infrastructure Integration. Applications include not only the integration for legacy, but also for Web Services, for most of the existing technologies (i.e. FTP, HTTP, mail, GSM, etc.) and the new application scenarios like business-to-business and business-to-consumer activities on the Web [2].

In [2] the concept of Interoperability is introduced as the way to *assure interaction, communication and cooperation between heterogeneous agents and heterogeneous services*. The aim of their work is to provide transparency between user and services: joining application as services to the agentspace on the one hand and for using and integrating them by heterogeneous agents on the other hand.

In this case, also like in [3], the service abstraction can work as the bridge to embed *coordination artifact* in the system following the coordination as a service approach [4].

In the next chapters the main issue concerning the integration of the Delegate MAS service in the AGV Simulator will be argued, taking in account the key prerequisites discussed in the previous section.

7.3 Some Important Notions on Environments

In chapter 3 the reference architecture was presented, underlying the *key notion* of the environment around which the main concepts were built, consistently with the notion of environment as a “*first-class abstraction in multiagent systems*”.

Indeed, before Weyns, J. Odell and colleagues have remarked the *primary role* of the environment as well, pointing out that

“*without an environment an agent is effectively useless [30]*”

The environment is a key issue because:

- the agency definition subsume that an agent is operating in a environment through perception and action means
- environment as a coordination infrastructure for cognitive agents allow to *separate responsibilities* in the system and to manage the complexity, starting from the agents that are, obviously, more simple than putting the full complexity of coordination in them
- environment provides a *medium* for sharing information and mediating coordination among agents
- it is the glue for agents behaviour: indeed it mediates agent interaction and supports agent access to resources

From these brief properties stem that environment has a *crucial role* not only for Situated MASs but for the paradigm of MASs as a whole [31].

In general MASs represent a way to handle the complexity of modern Complex Software Systems that consist of a number of related subsystems, within which the constituent components work together to deliver the overall functionality: indeed, modelling software as a composition of agents and designing them to be goal-oriented, it is possible to suitably face the complexness.

These agents, referring to the environment, have one important feature that is the situatedness, that is the feature of being situated in a **common computational and/or physical environment** from both the spatial and temporal standpoint.

Complex MASs, however, can ask for multi-environments definition, because:

- openness is another critical and novel feature of Complex Software Systems for which systems are permeable and subject to change in size and structure: the different agents may need to use an environment as communication medium to interact
- the application domain can encapsulate some functions or services for the agents characterized by different ontologies: each environment can target a specific function or service of the application with one specific ontology
- the integration of new services can bring agent to access environments having different ontologies: one way can be the use of an environment as communication medium

The reference architecture presented in 3 embrace these issues: figure 7.1 presents the 3-Layer model in which three class of environment can be distinguished [24]:

- *application environment*, the environment of the MAS application layer
- the environment defined by the *execution platform*
- the environment defined by the *physical infrastructure*

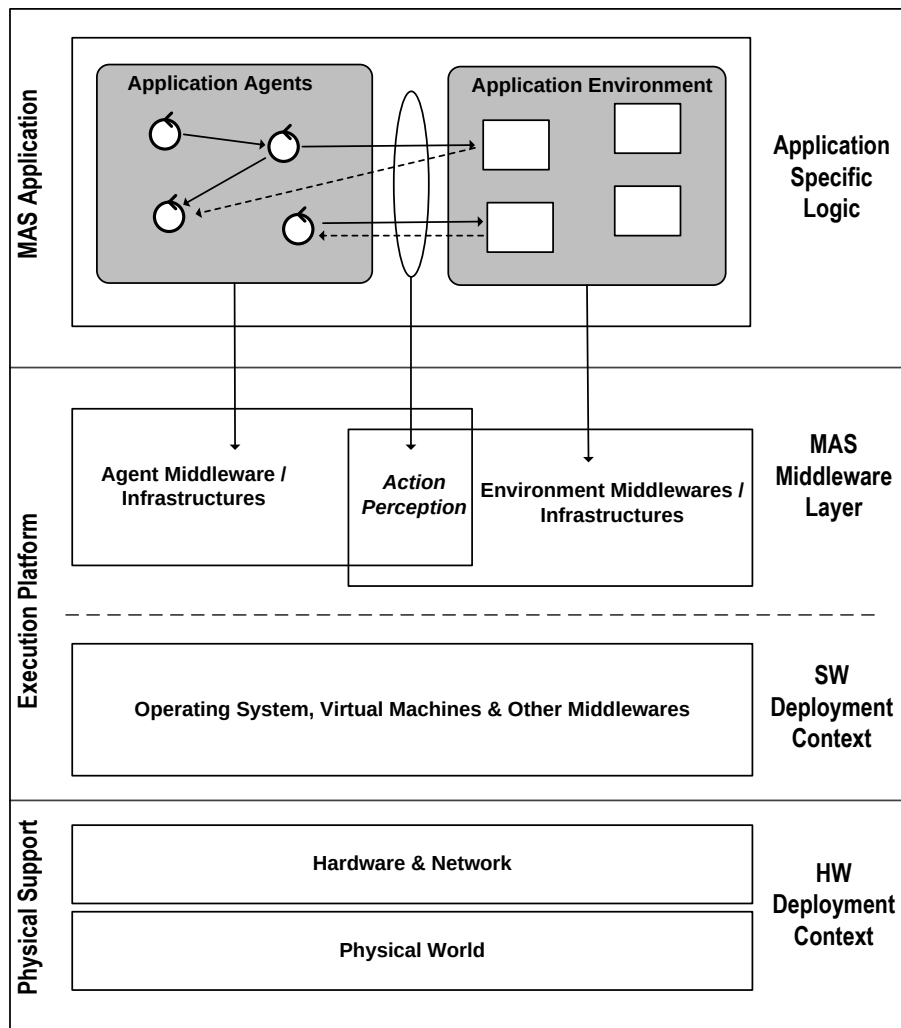


Figure 7.1: 3-Layer model for MASs [32]

As can be seen in figure 7.1, the application environment can be decomposed in building block called *environment abstractions*: indeed an environment embrace

and provides certain functionalities.

It is important to underline some aspects about the agent-environment relationship. Russel and Norvig book has been presented a scenario in which sensors and actuators are situated on the agents' side: this means that the sensors and actuators are defined by the ontology of the agents and that is the environment that depends from the agents' ontology (figure 7.2). It is obviously presumed that the environment constituents are used by agents and never the opposite. Besides, in opposition with agents definition and properties, the environment is reactive, function-oriented design and behaviour, partially (or sometimes complete) controllability, and so on.

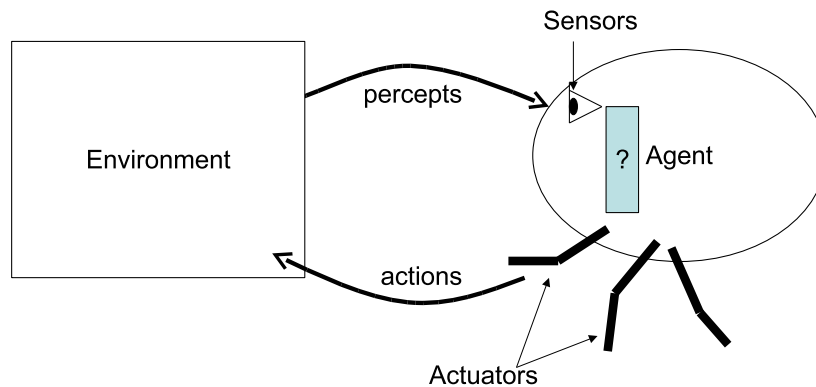


Figure 7.2: The original agent-environment diagram by Russel and Norvig [24]

Keeping to the diagram defined by Russel and Norvig it is clear that the environment is not independent from the specific model of an agent. Putting the actuators and sensors on the environment side, instead (figure 7.3):

- the *environment providing functionalities* makes sense
- an environment can host agents that use different ontologies and reasoning models
- the environment encapsulates significant portion of the system's complexity in terms of services, mechanisms and responsibilities that the agents can fruitfully be free of [32]

It is clear that environment offers to the agents actuators and sensors in order to act and perceive on it: so the environment ontology has to be explicitly defined.

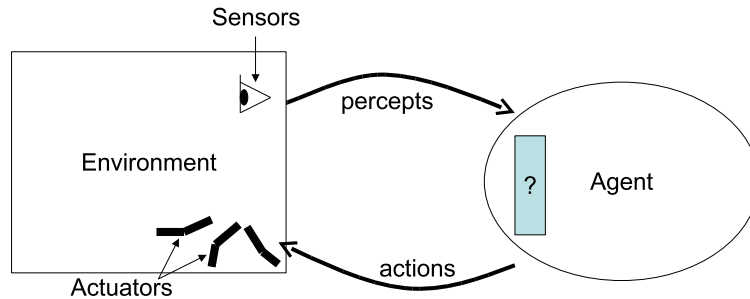


Figure 7.3: Environment provides actuators and sensors to the agents [24]

With these assumptions it is easy to deduce that in a multi-environment scenario the agent-environment interaction would have been more difficult since every environment would have to follow the ontology of each agent.

In particular, it is interesting the scenario in which one environment is used as communication medium between agents and the environment. Figure 7.4 depicts this case.

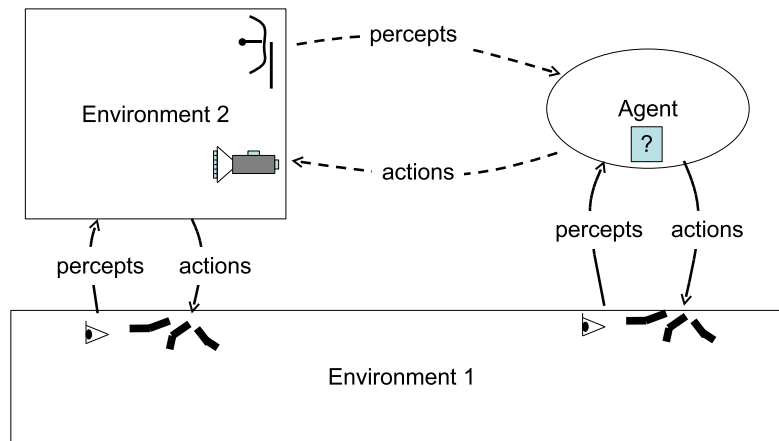


Figure 7.4: Environment as communication medium to access another environment [24]

Agents (indirectly) access environment 2, having (directly) access to the environment 1 that is the communication medium: they do not have to know anything

about the environment 2. So they are awareness of the environment 1's ontology: this makes integration of services and interaction between different agents reasoning models more easy.

7.4 Infrastructures for Environment

An important element related to the environment is the **infrastructure**, which encapsulates and supports critical features and properties of MAS expressed in terms of services, convention and knowledge. Its main role in MAS is to model and shape the *agent environment*, from the two points of view (i) of the agents living in the MAS; and (ii) of MAS designers [33].

It is possible to distinguish two important classes of infrastructures: *agent infrastructures* - like JADE , RETSINA or JACK - and *environment infrastructures* - like TOTA , TuCSoN or AMELI . The former provides the designer with the agent concept while the latter provide the designer with tool to form the environment. These are very useful abstraction to *model* and develop the environment of the multiagent systems.

Multiagent systems do not have always to resolve new types or classes of problems, so it is clear that building them from scratch each time hates *reuse* and *effectiveness*, pillars of software engineering principles.

To avoid this trend, it is almost a mandatory to develop a *middleware*, that Viroli et al. in [32] defines as *a software layer handling the life-cycle of environment abstraction and theri interaction with agents*. So, considering that a middleware for environments can be thought as an *infrastructure* that provide some class of environment abstractions at run-time, by analogy with agent infrastructure, a middleware of this kind can be seen as an *infrastructure of services* that provide certain environment abstractions to agent.

Also in this case it is possible to distinguish two kinds of MAS middleware:

- an infrastructure for agents (e.g. JADE)
- one or more environment infrastructures (e.g. TuCSoN)

that cannot be completely isolated. In figure 7.1 can be see that an intersection between them exists: actions and perceptions are *agent to environment interactions*. Not only agent-environment interaction but also interactions between environment abstraction can occur.

7.4.1 MAS Infrastructures for Environment: TuCSon

TuCSon is an *agent infrastructure* providing coordination services through *artifact* called *tuple centres* which are collected in nodes of the network and belong to specific organizations [34]. Each tuple center is a *programmable tuple space*: indeed is a space of logic tuples that can be programmed to react to certain interaction events (e.g. insertion or removal of certain tuples). Obviously, the state of the space is modified accordingly, through a chain of internal transitions [32]. Agents *interact* by writing, reading, and consuming *tuples* to/from tuple centres via simple communication operations (*out*, *rd*, *in*) that access tuples associatively. ReSpecT is the logic-based language used to program the tuple centres and to make them encapsulating *coordination laws*.

As a result:

- TuCSon infrastructure supports the creation, access and manipulation of tuple centres, distributed among the nodes of the multiagent system
- TuCSon makes no assumption on the model of individual agents: that is this agent infrastructure can be, in principle, exploited along with different agent platform

Figure 7.5 shows a logical view of a multiagent system, split in the three layers as reference architecture like, with TuSCoN used at the middleware layer. Over TuCSon, one application can be engineered as a multiagent system, where agents play a role and tuple centres represent some abstractions of the application.

One important aspect to underline is that agents can be built on **any middleware** and only *require the TuCSon API* (written in Java) in order to access tuple centres.

7.4.2 MAS Infrastructures for Environment: Virtual Environments in the AGV Application

In chapter 3, the reference architecture for AGV Transportation System has been deeply discussed. Here the goal is to focus the infrastructural support for the Virtual Environment in the AGV Application.

The AGV application is supported by the ObjectPlaces infrastructure, where unmanned vehicles controlled by agents transport various kinds of loads through a warehouse. A “virtual environment” (VE) is supported that keeps a *consistent* and *updated* map of the physical environment (including vehicles’ location and status, such as whether they are executing a job). This is realised through a number of

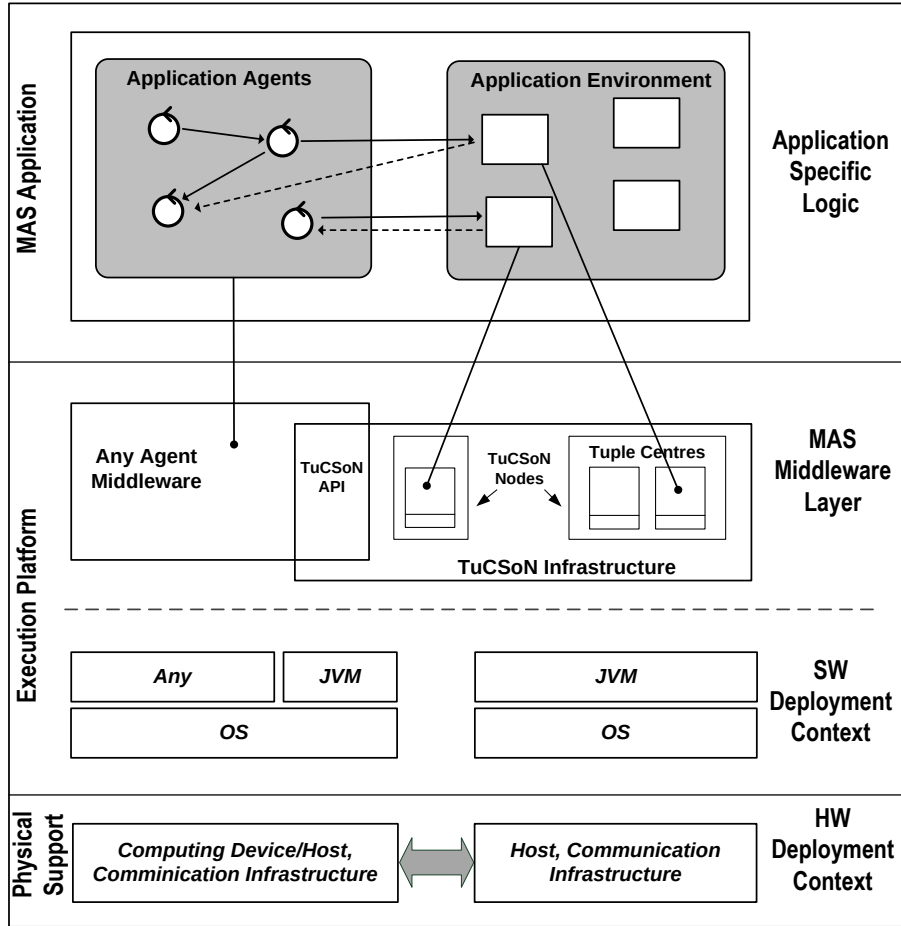


Figure 7.5: MAS layers with TuCSoN as agent infrastructure [32]

partial representations called *local VEs* (LVE): each AGV keeps a LVE used as the information to share.

Besides AGVs, the system contains one or more *transport bases*, where orders are generated and assigned to AGV agents by a number of Transport Agents. Each Transport Agent is responsible for assigning one transport to an AGV, and for making sure it is executed. To achieve this goal, a Transport Agent uses a *LVE* to get an up to date view on the AGVs position and status. Other than mere sharing, the VE encapsulates important functionalities to support agent cooperation. Agents can e.g. put marks in the VE to avoid collisions, and the VE takes on the burden of handling the interaction protocol between the various AGVs in the mobile network to synchronise these marks.

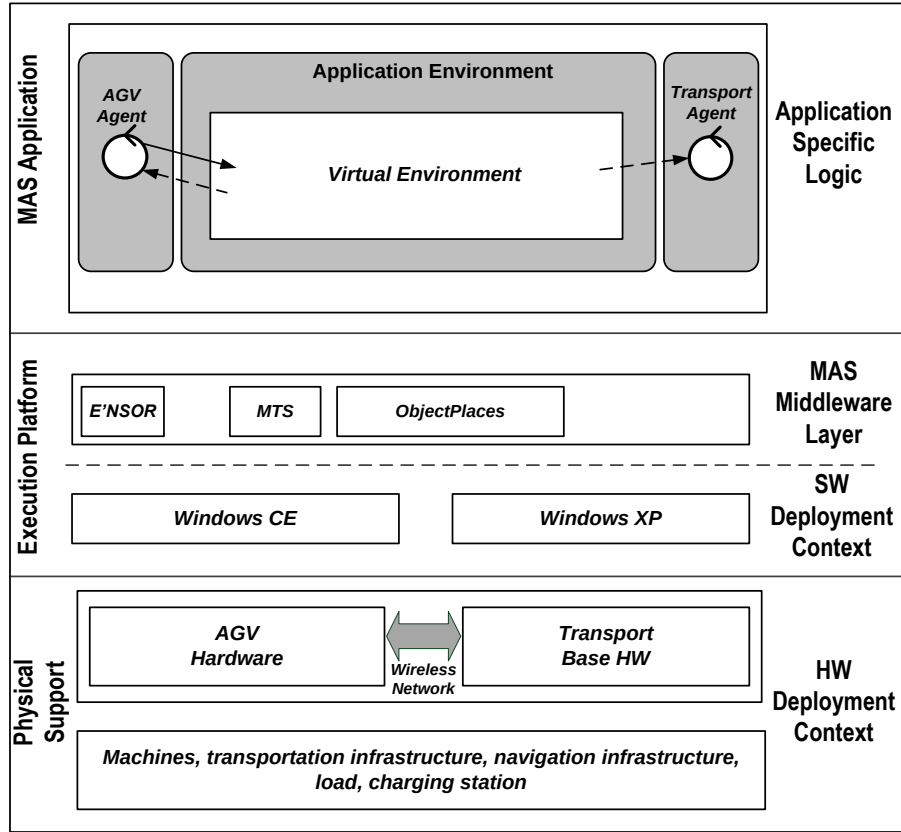


Figure 7.6: AGV Application layers [32]

Figure 7.6 represents the AGV application, with the various layers in evidence.

- the *ObjectPlaces* middleware implements the distributed environment for the agents
- the *Message Transfer System* (MTS) is a middleware used to enable communications between AGVs and Transport Agents
- E'nsor ® (Egemin Navigation System On Robot) is a S/W layer running on AGV robots which handles their low-level control

In this application, the two middlewares *ObjectPlaces* and *MTS* both contribute in devising an infrastructure providing a shared environment to agents: the global VE can hence be considered here as the global and distributed environment abstraction made available to agents.

7.5 Towards Integration

7.5.1 Delegate MAS as Coordination Service?

In the previous sections, important issues, such as infrastructure and environment, has been explored to achieve the horizontal integration goal. However, other ingredients are needed: one of these is the coordination issue.

Coordination in multiagent systems has been focused as natural result of autonomous agents living and interacting within agent societies & MAS. Two main viewpoints are interesting for Delegate MAS model:

- the subjective viewpoint, for which coordination can be conceived as an agent activity: is required when defining individual agents' architecture and inner dynamics [29]
- the objective viewpoint, for which coordination can be conceived as an activity over agents: in order to have observers' goals achieved (objective viewpoint) it is necessary to affect somehow agents' interaction space [29]

Subjective and objective coordination were defined as coordination *inside* and *outside* the agents respectively: in the former the components are the *coordinating* entities while in the latter the components are the *coordinated* entities.

Even if the differences between these two approaches has led to radically different way of building multi-agent systems, a unifying view of these approaches is not only possible but also *necessary* in complex scenarios where intelligent autonomous entities are required to interact in a system achieving global goals that do not simply reduce to single agents' individual goals [29].

Delegate MAS support both viewpoints:

Subjective viewpoint the use of Feasibility and Exploration Ants can be considered a sort of delegated environment sensing (resulting in agent's perception), while Intension Ants, by spreading agents' intensions execute some kind of agent actions, whose effects influence both the issuer agent and the other members of the society: the resulting coordination arises from single agents' reasoning and perception of the environment (intra-agent issues) [29]

Objective viewpoint In the case dependencies among tasks are present, Delegate MAS can have in Delegate-d Ants the fundamental environment abstractions contributing to an objective coordination. In that they are immersed in the environment and, being enhanced with more responsibilities than actually are,

they can act by modifying the agent environment or using environment resources (other than topological ones) such to satisfy global goal [29]

Moreover, Tedeschi in [29] has underlined that is possible to think about Delegate MAS as a *new coordination pattern* where two kinds of agents, namely Task Agents and Resource Agents, exploit this coordination pattern to achieve “entering tasks” execution, alleviating them from the task assignment responsibilities.

Following the **coordination as a service approach** [4], the Delegate MAS can be conceived as a *service*, where the service abstraction can work as the bridge to embed *coordination artifacts* inside a multiagent system for which Delegate MAS can be a suitable means to achieve global goals.

This way, conceiving Delegate MAS as a coordination service, Task Agents can exploit the service both if agents and Delegate MAS share the same system (*internal*) and if the Delegate MAS is *external* to Task Agents’ system.

So, it is now clear that the final objective is to capture the concepts of Delegate MAS’s service elements, thus enabling the integration with the AGV Simulator.

7.5.2 Delegate MAS as an AGV Coordination Service

Looking towards integration, the main difficulty arising is AGV Simulator considers interactions both among agents, and between agents and the (logical/physical) environment, having no knowledge about the artifact abstraction.

So, the first point here is how to introduce coordination artifacts in the AGV Simulator world. This raises a couple of relevant issues:

Discovery and Awareness How do agents discover and become aware of the coordination artifacts available in a specific organization context? How can they understand the coordination services provided by the artifacts?

Use & Management How can interaction between agents and coordination artifacts be modeled and engineered?

In this thesis, the focus is on the *latter issue*.

In [29], a mapping of the Delegate MAS model into the A&A framework has been proposed. This mapping is one of the most important pillars for the final integration because it allows to define the bridge between AGV agents, BDI agents, and the A&A meta-model, in which agents, artifacts and workspaces are the main abstractions.

The result of the mapping in the A&A meta-model can be summarize as follows:

Agents Task Agent, Resource Agent, Feasibility Ants, Exploration Ants, Intention Ants

Individual Artifacts Scheduling Artifact, Task Management Artifact, Resource Management Artifact, Feasibility Artifact, Exploration Artifact, Intention Artifact

Environment Artifacts Node Artifact

Social Artifacts Delegation Artifact, Communication Artifact

Aggregates Delegation Aggregate: Feasibility Ants, Exploration Ants, Intention Ants, Delegation Artifact

Workspaces Node Workspace

Use interaction inherited from Architectural design phase

According to the A&A meta-model, interaction within MAS could occur in five ways [35]:

communication agents *speak* with agents (*speech act*)

operation agents *use* artifacts (*physical acts*)

composition artifacts *link* with artifacts

presentation artifacts *manifest* to agents

reaction (environment) artifacts *react* to events coming from the resources (of the MAS environment) [36]

For this work, two of these interaction are interesting: the *operation* and the *reaction* one. Obviously, this two different types of interaction entail different to AGV agents to conceive coordination service:

- in the first case the agents, in order to use artifacts, have to be aware of the set of operations which agents can execute to use them and, obviously, have to share the same environment

- in the last case the agents do not have *direct* knowledge of the artifacts but they *indirectly* use it through the environment: it has to know the operations which can be execute to use the coordination artifacts. In this mode agent actions spawn an event in its LVE at which the (environment) artifacts react.

In such a conceptual model, the coordination artifact is then to be exploited by BDI agents by accessing a specific service: the Delegate MAS service. Accordingly, embedding coordination artifacts as coordination services requires:

- the definition of a specific environment artifacts, as well as a coordination artifact: only an environment artifact is directly connected with an environment entity
- according to the mapping, the definition of an environment artifact and a coordination artifact for every Node Artifact and their relationship (artifact linkability)
- the definition of the physical act that the agent has to execute on its environment (LVE) in order to produce the proper event

7.5.3 Adding the Delegate MAS Service

According to the mapping of the Delegate MAS model into an implementation oriented architecture built up on TuCSoN infrastructure [29], adding the Delegate MAS Service means make AGV agents able to access and use tuple centers provided by TuCSoN infrastructure (as coordination artifacts) in an unaware way. In a unaware way because AGV agents can execute basic operation on a tuple center (*in*, *out* and *rd*) indirectly accessing tuple centers through their LVEs.

Figure 7.7 shows a logical view, at a high-level of abstraction, of the (TuCSoN-based implemented) Delegate MAS service integrated with AGV Transportation System, each supported by the own infrastructure: TuCSoN for Delegate MAS, ObjectPlaces, MTS, E'nsor middleware for the AGV application. To well understand how horizontal integration works, a fine-grain model description is needed.

It is possible to define a fine-grain model of the Horizontal Integration specifying some important issues.

LVEs allow to keep a *consistent* and *updated* map of the physical environment (including vehicles' location and status, such as whether they are executing a job). In order to make integration possible, a connection between LVE representation of the physical environment and Delegate MAS once is needed. Figure 7.8 shows

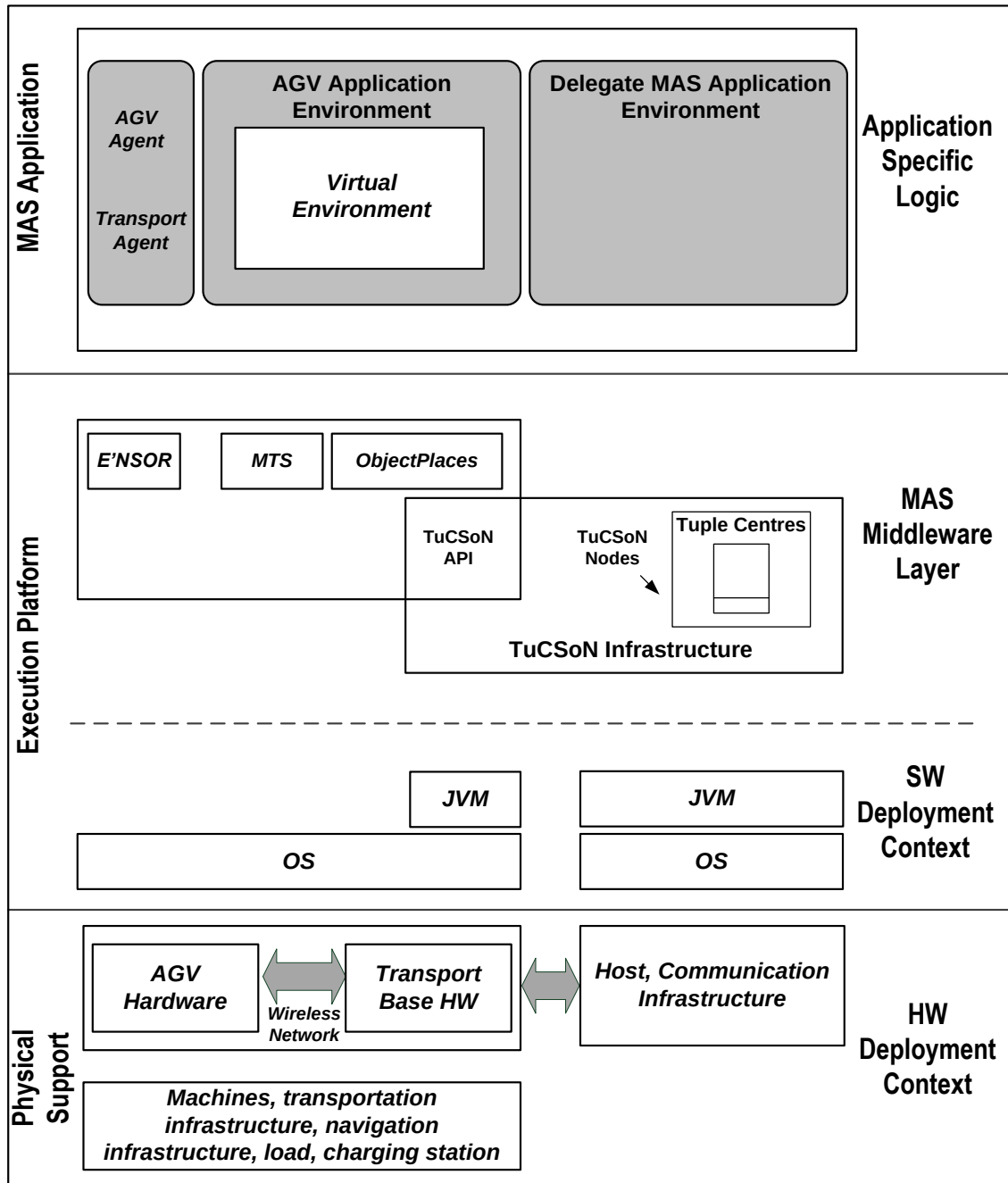


Figure 7.7: MAS layers representing Horizontal Integration

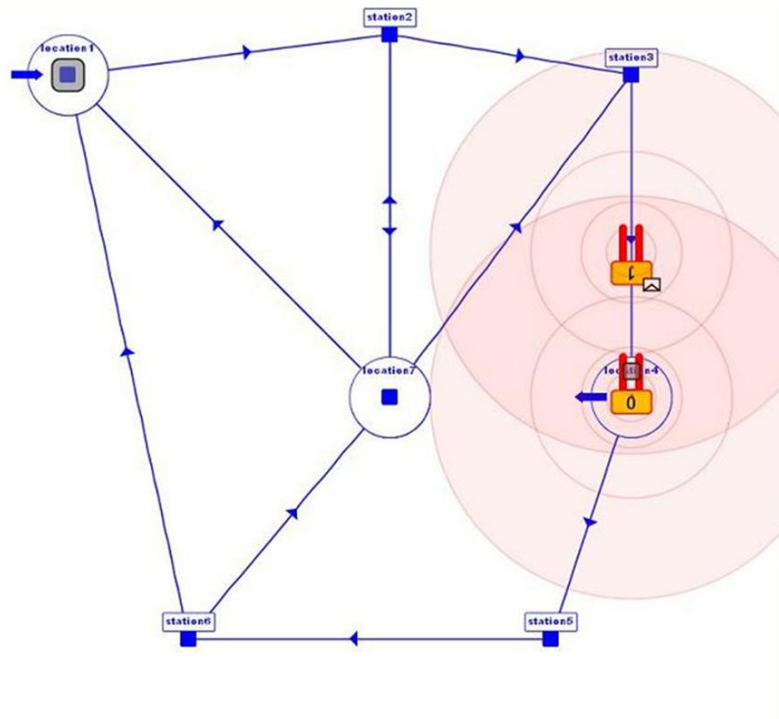


Figure 7.8: A representation of the layout

an example of a virtual representation of the real AGV environment. Important concepts to underline are:

Station any point on which the AGV can stand still, and from which other segments start or arrive in

Segments the paths on which AGVs can drive

Locations a station where an AGV can perform an action (picking or dropping a packet)

According to this topology and to the mapping of the Delegate MAS in the A&A meta-model, can be pointed out that at every Station and Location corresponds a Workspace (in Figure 7.9 a concrete topology for a coordination and control application is taken as an example). AGV and Transport Agents, in order to use Delegate MAS service, as argued in the previous sections, have to interact properly with their LVE. They can perform some predetermined actions on their LVE:

AGV Agents can (i) move over the layout, (ii) observe their immediate environment, (iii) communicate with each other, (iv) pick and drop packets, (v) wait a second

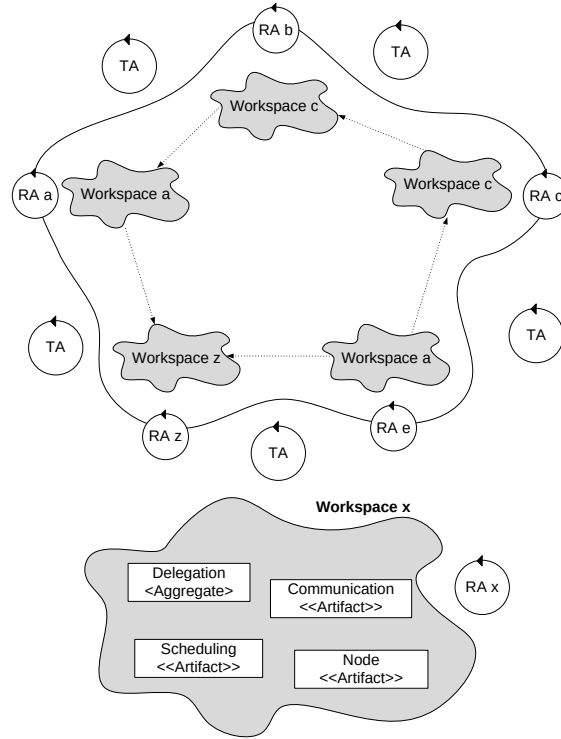


Figure 7.9: Example of concrete design scenario and node abstractions [29]

Transport Agents insert new task in the system, meaning that a new load to pick up is added in a specific location

To make AGV Agents able to use the Delegate MAS service, the actions they can perform have to be extended with the *ant issuing operation*, allowing AGV Agents to issue Feasible, Exploration and Intention Ants according to the Delegate MAS model: this new action, as well as the other, is obviously an action on its LVE.

On (TuCSoN -based implemented) Delegate MAS some important concerns have to be discussed as well. Firstly, it is possible to identify the delegation artifact as the key abstraction for the coordination issue: but how can it be exploited by AGV Agents? Part of the response to this key *dilemma* can be found in [36], in which *ReSpecT*, the language for MAS coordination has been extended according A&A meta-model providing a simple and solid grounding for MAS *situatedness* [36]. The key issue is that environment artifact, handling the interactions between an external entity of some sort belonging to the MAS environment and the MAS itself, can react to events coming from the environment resources. According to this issue, it is clear

that the definition of an environment artifact for every Workspace makes interaction between LVEs and delegation artifact possible: obviously the *composition* interaction, thanks to which delegation and environment artifacts are *linkable*, is necessary to guarantee the proper interaction behaviour.

To call integration of the Delegate MAS service quits, at last being able to answer the previous *dilemma*, only one issue need to be discussed. A sort of *transducer* between the LVE and the artifact is needed: it has to translate the action performed by (AGV and Transport)agents into events sent to the environment artifact (TuCSoN API) and to translate the invocation by environment artifact into a perception to the (AGV and Transport) agents. This transducer, being a real bridge between the two environment, is so called *bridge*. The bridge, according to the including vehicles' current location kept by LVE (in which it is defined), is able to access the proper workspace. Figure 7.10 shows the chain of interaction from the AGV application to the (TuCSoN -based implemented) Delegate MAS and vice versa.

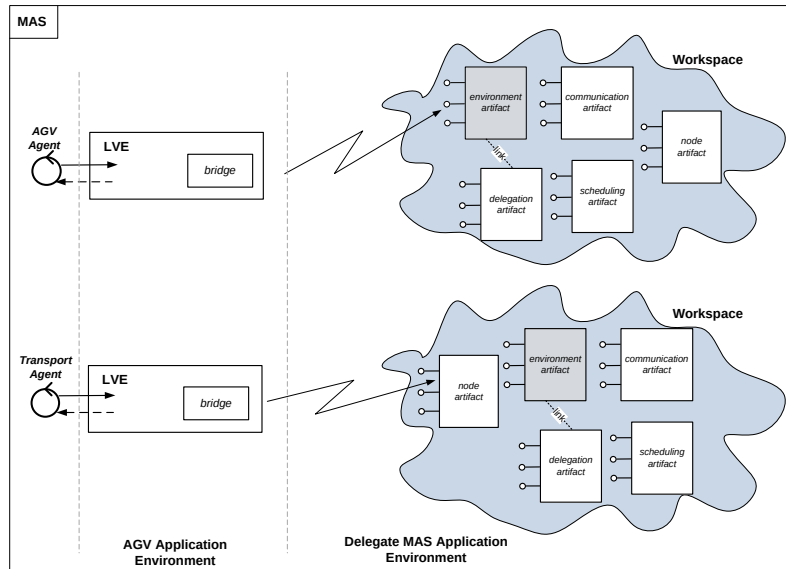


Figure 7.10: Fine-grain model of the Horizontal Integration

Part IV

Conclusion

Chapter 8

Conclusion

This thesis has discussed the integration of the Delegate MAS model into the AGV Transportation Systems. In particular, according to a mapping of the Delegate MAS model into the A&A framework and its mapping into an implementation oriented architecture built up on **TuCSon** , the keys issues have been discussed.

This work have led towards the theoretical definition of all concerns making the proper interaction behaviour, between the two different systems, possible.

The choice was to build a bridge between the Delgate MAS environment and the AGV Virtual Environment, allowing tuple center of a **TuCSon** node to be directly accessed from the AGV application. This way *(i)* requires **no re-design** of the two infrastructures, this way resulting in more theoretical effort rather than practical one, *(ii)* remark the key role of infrastructure for MASs for which this integration has been possible and *(iii)* **promotes heterogeneity**, since adding the Delegate MAS as a service for the AGV AGV Transportation System, *paves the way* for integration of other kinds of services that could be interesting and useful for the AGV application.

8.1 Future Works

In Chapter ?? a question was put: Does Delegate MAS represent an innovative approach, really improving the whole system behaviour and alleviating agent (AGV) complexity?

The best way to answer this question is to apply Delegate MAS in AGV Transportation System, comparing Delegate MAS with DynCNET and FiTA through some tests (like in [37]): in this mode is possible to know exactly if the new approach meets or not expectations.

This thesis paves the way to this direction. Obviously, an implementation of the theoretical integration discussed in this work is needed to compare Delegate MAS with DynCNET and FiTA through some tests.

Moreover, a number of parameters can affect the behaviour of Delegate MAS approach such as the ant issuing frequency, the number of tasks to execute that an AGV agent can book, etc: it could be interesting to tune up these parameters during the simulation. Not only the parameters to be tuned are interesting but also the way in which this can be done: in an *endogenous* way, for which the system components are aware of the tuning and cooperatively realize the tuning parameter, in an *exogenous* way, for which the system is adapted through an external entity (e.g. an agent) and engineering the tuning according to the integration conceptual model.

Bibliography

- [1] Omicini A. “*Trends & Experiences in MAS Infrastructures*”, Course on Multiagent Systems, Seminar, A.Y. 2006/2007, Alma Mater Studiorum-Università di Bologna a Cesena
- [2] Stanislaw Ambroszkiewicz and Tomasz Nowak “*Agentspace as a Middleware for Service Integration*”, pp. 134-159, ESAW '01: Proceedings of the Second International Workshop on Engineering Societies in the Agents World II, 2001
- [3] Andrea Omicini, Alessandro Ricci, Mirko Viroli, and Marco Cioffi DEIS, Università di Bologna, Cesena, Italy, Giovanni Rimassa, Research and Consulting Department, Whitestein Technologies AG, Zurich, Switzerland, “*Multi-Agent infrastructures for objective and subjective coordination*”, Applied Artificial Intelligence, Volume 18, Number 9-10, Pages 815-831, 2004
- [4] Viroli M. and Omicini A. “*Coordination as a Service*”, Fundamenta Informaticae 73(4), 2006
- [5] Weyns D. and Holvoet T. (2007) “*Architectural design of a situated multiagent system for controlling automatic guided vehicles*”, Int. J. Agent-Oriented Software Engineering (IJAOSE)
- [6] Delbaere W. and Lamberigts B., “*Flexible Integration of Functionalities in a Decentralized AGV Transportation System*”, Department Computer Science, Katholieke Universiteit Leuven, Master Thesis, 2007
- [7] Omicini A., “*Agents and Artifacts: The A&A Meta-model for Multiagent Systems*”, Multiagent Systems LS, Academic Year 20062007, Alma Mater Studiorum-Università di Bologna a Cesena
- [8] N.R. Jennings “*An agent-based approach for building complex software systems*” Communication of ACM, 44(6):35-41, April 2001

- [9] Ricci A. “*A. Engineering Agent Societies with Coordination Artifacts and supporting Infrastructures*”, Ph.D. Thesis, DEIS, Università degli Studi di Bologna
- [10] Weyns D. Omicini, A. Odell, J. “*Environment as a first class abstraction in multiagent systems*” *Autonomous Agents and Multi-Agent Systems* 14(1): 5-30 2007
- [11] Weyns D. “*An Architecture-Centric Approach for Software Engineering with Situated Multiagent Systems*”, PhD Dissertation, Katholieke Universiteit Leuven, 2006.
- [12] Bass L., Clements P., Kazman R. “*Software Architecture In Practice*”, Second Edition. Boston: Addison-Wesley, p. 21-24. ISBN 0-321-15495-9. 2003
- [13] Clements P., Kazman R. and Klein M. “*Evaluating Software Architectures: Methods and Case Studies*” Addison Wesley Publishing Comp., 2002
- [14] Reed P. “*Reference Architecture: The Best of Best Practices*” The Rational Edge, 2002. www-128.ibm.com/developerworks/rational/library/2774.html
- [15] Weyns D. and Holvoet T. “*Architecture-Centric Software Development of Situated Multiagent Systems*”, ESAW 2006, LNAI 4457, pp. 62-85, 2007.
- [16] Weyns D., Steegmans E., Holvoet T. “*Towards Active Perception in Situated Multi-Agent Systems*” *Applied Artificial Intelligence* 18(9-10), 867-883 2004
- [17] Ferber J., Muller J. “*Influences and Reaction: a Model of Situated Multiagent Systems*” In: 2nd International Conference on Multi-agent Systems, Japan. AAAI Press, Stanford, California, USA 1996
- [18] Weyns D., Holvoet T. “*Formal Model for Situated Multi-Agent Systems*” *Fundamenta Informaticae* 63(1-2), 125-158 (2004)
- [19] Weyns D., Steegmans E., Holvoet T. “*Integrating Free-Flow Architectures with Role Models Based on Statecharts*” In: Choren, R., Garcia, A., Lucena, C., Romanovsky, A. (eds.) *Software Engineering for Multi-Agent Systems III*. LNCS, vol. 3390. Springer, Heidelberg 2005
- [20] Steegmans E., Weyns D., Holvoet T., Berbers, Y. “*A Design Process for Adaptive Behavior of Situated Agents*” In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) *AOSE 2004*. LNCS, vol. 3382. Springer, Heidelberg 2005

- [21] Weyns D., Steegmans E., Holvoet T. “*Protocol Based Communication for Situated Multi-Agent Systems*” In: 3th Joint Conference on Autonomous Agents and Multi-Agent Systems, New York, USA. IEEE Computer Society Press, Los Alamitos 2004
- [22] Kirsh D. “*Distributed cognition, coordination and environment design*”, In Bagnara, S., editor, 3rd European Conference on Cognitive Science (ECCS’99), pages 1-11, Certosa di Pontignano, Siena, Italy. Istituto di Psicologia, Consiglio Nazionale delle Ricerche, 1999
- [23] James Odell, H. Van Dyke Parunak, Mitchell Fleischer, Sven Brueckner: “*Modeling Agents and Their Environment*” AOSE 2002: 16-31
- [24] Abdelkader Gouaich, Fabien Michel: “*Towards a Unified View of the Environment(s) within Multi-Agent Systems*”, Informatica (Slovenia) 29(4): 423-432, 2005
- [25] FIPA: Foundation for Intelligent Physical Agents, FIPA Abstract Architecture Specification (8/2006), <http://www.fipa.org/repository/bysubject.html>
- [26] Bellifemine F., Poggi A., Rimassa G. “*Jade, A FIPA-compliant Agent Framework*”, In: 4th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK 1999
- [27] Smith R., “*The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver*, In *IEEE Transactions on Computers*”, C-29(12), 1980.
- [28] Holvoet T., Valckenaers P., “*Exploiting the Environment for Coordinating Agent Intentions*”, E4MAS 2006: 51-66
- [29] Tedeschi M., “*Mapping Delegate MAS model into the A&A framework*”, Master Thesis, Alma Mater Studiorum-Università di Bologna a Cesena, 2008
- [30] Odell J., Parunak H. V. D., Fleischer M. and Brueckner S., “*Modeling agents and their environment*”. In F. Giunchiglia, J. Odell, and G. Weiss, editors, Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002. Revised Papers and Invited Contributions, volume 2585 of Lecture Notes in Computer Science LNCS, pages 16-31. Springer, Berlin, 2003.

- [31] Weyns D., Parunak H. V. D., and Michel F., editors. “*Environments for Multi-Agent Systems*”, First International Workshop, E4MAS 2004, July 19, 2004, New York, NY, USA, Revised Selected Papers, volume 3374 of Lecture Notes in Computer Science LNCS. Springer, Berlin, 2005.
- [32] Mirko Viroli, Tom Holvoet, Alessandro Ricci, Kurt Schelfhout, Franco Zambonelli, “*Infrastructures for the environment of multiagent systems*”, Autonomous Agents and Multi-Agent Systems 14(1): 49-60, 2007
- [33] Omicini A., Ossowski S., Ricci A., “*Coordination infrastructures in the engineering of multiagent systems*”, In F. Bergenti, M.-P. Gleizes, & F. Zambonelli (Eds.), *Methodologies and software engineering for agent systems: The agent-oriented software engineering handbook, Vol. 11 of multiagent systems, artificial societies, and simulated organizations*, Kluwer Academic Publishers, Chapt. 14, pp. 273-296, 2004
- [34] Omicini A. and F. Zambonelli, “*Coordination for Internet application development*”, Autonomous Agents and Multi-Agent Systems 2(3): 251-269, 1999
- [35] Omicini A., and Ricci A. and Viroli M., “*Artifacts in the A&A meta-model for multi-agent systems*”, Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, vol. 17, num. 3, pp. 432-456, 2008
- [36] Casadei M. and Omicini A., “*Situating A&A ReSpecT for Pervasive Environment Applications*”, 7th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises. Workshop on Coordination Models and Applications (CoMA 2008), 2008
- [37] Danny Weyns, Nelis Boucké and Tom Holvoet, “*A field-based versus a protocol-based approach for adaptive task assignment*”, Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, vol. 17, num. 2: 288-319, 2008
- [38] Omicini A., Ricci A., Viroli M., “*Engineering MAS Environment with Artifacts*”, In: 2nd International Workshop Environments for Multi-Agent Systems, (E4MAS 2005), p.62-77, 2005.
- [39] Wooldridge. M., “*Intelligent Agents*”, In G. Weiß, editor, Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence, pages 27-77. MIT Press, Cambridge, Massachusetts, USA, April 1999.
- [40] Weyns D., Holvoet T., Helleboogh A., “*Anticipatory vehicle routing using delegate multi-agent systems*”, Intelligent Transportation Systems Conference, ITSC 2007. IEEE, pages 87-93, Seattle, USA, Sept. 30, Oct 2007

List of Figures

3.1	Deployment view of the AGV transportation System [5]	10
3.2	High-level model of an AGV Transportation System [6]	13
3.3	Top-level module decomposition of a situated MAS [15]	16
3.4	Collaborating Components of Agent [15]	17
3.5	Collaborating Components of Application Environment [15]	21
3.6	Module Decomposition View of the AGV Control System [6]	25
3.7	Collaborating Components View of the AGV Agent [6]	26
3.8	Collaborating Components View of the Decision Making Component of the AGV Agent [6]	28
3.9	Collaborating Components View of the Local Virtual Environment of the AGV [6]	29
3.10	Collaborating Components View of the Middleware of the AGV [6] .	31
3.11	Collaborating Components View of the Transport Base System [6] . .	32
3.12	Transport Assignment with FiTA	34
3.13	Transport Assignment with DynCNET	36
6.1	Exploration ants, issued by a AGV agent, scout feasible paths	54
6.2	Intention ant, issued by an AGV agent at t4, communicate the inten- tion of the vehicle agent through the environment	56
7.1	3-Layer model for MASs [32]	63
7.2	The original agent-environment diagram by Russel and Norvig [24] . .	64
7.3	Environment provides actuators and sensors to the agents [24]	65
7.4	Environment as communication medium to access another environ- ment [24]	65
7.5	MAS layers with TuCSon as agent infrastructure [32]	68
7.6	AGV Application layers [32]	69
7.7	MAS layers representing Horizontal Integration	74
7.8	A representation of the layout	75
7.9	Example of concrete design scenario and node abstractions [29]	76

7.10 Fine-grain model of the Horizontal Integration	77
---	----

Acknowledgments

This thesis is the end of almost six difficult years: thanks to all my parents, my dear friends, this work is dedicated to them.

Thanks to Professor Danny Weyns, Tom Holvoet, Alexander Helleboogh and all other members of the Distri-net research group for their hospitality, for their suggestions and knowledge, for their push towards the interaction and for their kindly regard: I'll bring this amazing experience and this dear person in my heart!

Thanks to all Professor of Bologna University for this difficult but stimulating years, in particular to the aliCE people that during the Master have taught very interesting topics. Naturally, thanks to Professor Andrea Omicini for his support during the abroad period and for the final work on Thesis.

Antonio Cauzo

Ringraziamenti

Finalmente si scrive un pò in italiano! Almeno i ringraziamenti!

Questa Tesi non è che un oggetto, un file informatico, elaborata serie di bit combinati tra loro. Eppure, data la loro essenziale natura, tipicamente priva di calore umano, questa collezione di elementi racchiude tanta parte di vita: quasi sei anni!

Pensare che numericamente non rappresenta una cifra troppo elevata ma nella vita di una persona questo raggio di tempo è sufficiente per vedere tante cose cambiare, noi stessi *in primis*.

Da zero a sei anni un bimbo appena in fasce non in neppure grado di parlare nè camminare, ma di strillare benissimo, si ritrova a scuola: incomincia il primo importantissimo cammino della sua vita, segnato da un primo acerbo inserimento nella società (da zero a sei anni l'asilo rappresenta una città!) e dal primo approccio con l'arte dell'apprendimento, fattori determinanti per tutta un'esistenza.

Poi, da sei a dodici è un attimo: non solo la scuola e l'apprendimento evolvono, ma iniziano ad affiorare i rapporti con le persone, si iniziano ad esplorare i mondi dell'amicizia, dell'infantile amore, del piacere e della passione per lo sport.

Da dodici a diciotto l'evoluzione corre ancora molto veloce: il bambino inizia a diventare un ragazzo, e la strada verso la maturità inizia a correre (per qualcuno di più, per altri meno) tra lunghissime salite e più o meno lunghe discese: si inizia a capire che oltre a noi stessi, intorno a noi esiste un Mondo che inizia ad apparirci piano piano.

Chi però decide di fare l'Università, deve rinviare la conoscenza e l'approfondimento con quel Mondo, per lasciare spazio allo studio dell'universo che ti si apre davanti pensando alla facoltà scelta: ecco, ogni punto di contatto con il Mondo si interrompe qui..e dell'universitario non si hanno più notizie..fino ad oggi però! :-P (naturalmente questa visione è molto autobiografica, quindi è normale che se il lettore non è il sottoscritto non si riconosce in questo breve racconto :-P)

Ciò che però voglio rimarcare è come, dietro al racconto di prima, ci siano tanti e tanti aspetti che non vengono citati perchè in qualche modo hanno fatto parte,

con mia grande fortuna, del mio cammino come una seconda ombra.

Questi aspetti hanno dei volti, hanno degli sguardi, hanno un modo di esprimersi ed esprimere loro stessi, hanno delle movenze, hanno dei difetti particolari, hanno dei pregi insostituibili, hanno delle mani, un cuore ma soprattutto hanno un ruolo **fondamentale** nella mia vita..

Papà, Mamma, Enrico la mia famiglia, il mio sostegno unico e insostituibile: il vostro amore incodizionato e costante mi ha guidato fin qui e mi guiderà per sempre per tutti quei percorsi che ancora mi attendono. Senza di voi questo traguardo non sarebbe stato possibile, in tutto e per tutto. Siete la mia anima più profonda..Vi amo, ma non riuscirò mai a ricambiare tutto ciò che mi avete donato perchè il vostro è troppo grande: spero di poterlo fare io a mio volta

Papà GRAZIE. Mi hai insegnato la dedizione il sacrificio, l'ordine e la voglia di imparare, il non smettere mai di crederci, credersi e di migliorarsi. Porterò per sempre quelle ore in portineria come un ricordo indelebile

Mamma GRAZIE. Mi hai insegnato la calma e il rispetto, mi hai insegnato a guardare più in là, a ragionare e a non smettere di sognare. Mi hai accudito, coccolato, rimproverato, aiutato e io sono cresciuto amato, riverito :-P, maturato

Enrico GRAZIE. Non è descrivibile la gioia di un padre quando nasce un figlio ma nemmeno quella di un bambino quando nasce un fratello, perchè inconsapevole..anche se io l'ho tanto desiderato. Insieme siamo cresciuti e stiamo crescendo non solo fisicamente: in questo ultimo periodo un pò di Km ci hanno solo diviso materialmente, ma sono sicuro che tra non molto torneremo a condividere tutto come un tempo! Io lo voglio fortemente! Che bello averti visto crescere e maturare, venirti a prendere a scuola come a calcio, parlare di calcio ma anche di noi! Anche grazie a te sono migliorato!

Nonni e Zii Indescrivibili le sensazioni provate vivendo quel poco che ho potuto assaporare di loro, quante cose mi si sono stampate dentro guardando solamente i loro occhi, sentendo pronunciare poche parole giuste: un tesoro del mio cuore! L'elenco delle persone è molto ampio ma racchiude tutta la mia famiglia: Nonno Uccio, Nonno Giuseppe, Nonna Desdemona, Nonna Tetta, Zio Remo, Umberto, Cosimino, China, Claudio, Franco, Francesco, Zia Assuntina, Linda, Daniela, Anna Maria, Auri. Anche il vostro sostegno è stato importante e anche questo rimarrà impresso!

Un particolarissimo e affettuosissimo ringraziamento ad una Zia inevitabilmente speciale: Zia Antonia!! Non solo ho acquisito una persona speciale e da

un cuore grande un Mondo, ma ho anche ricevuto da lei amore, gioia, calore: non è una Zia per me, ma rappresenta la Zia!!!!!!

Rosy Come esprimere il mio affetto per te? Per la vicinanza in tutti i nostri anni a Forlì e per quando sei tornata poi a casa. Come descrivere la tua presenza nella nostra vita, i pranzi e le cene assaporando quel clima di serenità, gioia, gogliardia che in casa nostra portavi e scatenavi? Come raccontare che quel pezzo di Storia non voleva e non voleva entrare? “*Non è chi siamo, ma ciò che facciamo che ci nobilita*” ma tu sei, per me, una figura nobile. E quel nome, Rosa, espressione di dolcezza e di severità al tempo stesso (se la rosa non la prendi bene, ti pungi!) mi rimarrà sempre impresso nel cuore e nella mente!

Guerrina&Divo,Paolo&Daniela Il calore di una taverna, tra un boccone e altri quattro..intanto risate e serenità. Rappresentanti di una generosità incodizionata: grazie, perchè niente mi era dovuto!! Vi voglio bene!

Carolina&Marco Carolina e Marco, due persone che non avrei mai immaginato di incontrare: anche in questo una generosità, una dolcezza e un calore disumano! Siete un esempio e una presenza importante nel mio cammino!! Sarete sempre nel mio (piccolo, rispetto al vostro) cuore!

Ana&Paul Due persone speciali, due persone con le quali è bello poter discutere ma anche bere un bel bicchiere di vino davanti ad una bella pizza. Sapevo che potevo contare su di voi e di questo vi sarò sempre grato. La stima che si nutre è reciproca, e questo mi riempie di orgoglio e di felicità! Mi avete insegnato tanto, spero che questo seme possa germogliare davvero! Vi voglio bene!

Frens Schivo e sbrigativo, gioviale e pieno di vita, intelligente e un pò pazzo: Frens, unico!!! E visto che troppe cose dette non ti piacciono, sai cosa scrivo? Ciao amico, ti abbraccio fortissimo!! :-P

Merilin Se l'amicizia avesse un nome e cognome la chiamerei Merilin Tedeschini (anche perchè l'amicizia è donna :-P). Nell'ambiente ostile e infido della vita che iniziavamo ad esplorare, hai rappresentato un pilastro del mio quotidiano, irrinunciabile e fondamentale sempre di più! Le difficoltà di questi anni non si può dire che si siano attutite: poter condividere le fatiche, ti fa guardare avanti con ansia e preoccupazione comunque, ma ti rincuora quanto basta per poter continuare a correre. In più, studentessa molto brava, dalla quale ho imparato tante cose molto importanti, e con la quale la cooperazione e il confronto è stato motivo di crescita individuale e di gruppo. Sarebbe stato tutto molto diverso se non ci fossimo incontrati. Ti voglio bene!

Marco Amico fidato e compagno di tante avventure nel corso degli anni, con il quale sono cresciuto e sono maturato. Sei entrato a far parte sempre più del mio quotidiano, a partire da quando, bambini, ci siamo detti amici, fino ad oggi, in cui lo facciamo in maniera sempre più consapevole. In questi anni hai rappresentato una presenza importante e spero continuerà ad essere così..anzi sempre meglio! :-P Abbiamo condiviso parti importanti della nostra vita, ma non per questo sei per me una persona speciale: la tua naturale ed infinita serietà, segna tutt'ora la nostra vita..e quella degli altri! Chiedere in giro per chiedere! :-D

Gabriele Ignazio, come posso non ringraziarti per la tua sensibilità e attenzione verso l'altro? Nel terreno arido della vita, forse è nata un'amicizia! Ed è quasi un tesoro!!! Grazie per i momenti di giovialità che mi hai sempre saputo trasmettere!

Sofia,Pamela,Michele Sofia e Michele, compagni di, purtroppo, non troppe uscite (ma la musica cambierà..almeno lo spero :-P) con i quali è veramente facile divertirsi. Grazie per i bei momenti insieme! Pamela, carissima, anche in questo caso la lucina dell'amicizia si sta accendendo..spero vivamente che possa continuare ad alimentarsi! Grazie per essermi stata vicina in questo ultimo periodo molto difficile!

Daniele,Francesco,Alex *Dulcis in fundo*, Carletto, amico!!! Io TI RINGRAZIO per avermi sempre tenuto compagnia con quella bellissima musica che ascolti dal primissimo mezzogiorno (ora in cui ci degni della tua presenza) fino a tarda notte (quando la Forlì bene ti acclama a gran voce). Io TI RINGRAZIO per avermi degnato della tua presenza in casa: da un campione come te ho capito cosa voglia dire il sacrificio, la dedizione, la correttezza ma soprattutto ho capito che al mondo c'è qualcuno più furbo ma molto più furbo dei Cambogiani che si arrampicano sugli alberi! TE SALUTU! Francesco, cuoco giurato e guardia impeccabile. Da te ho imparato cosa voglia dire lavorare duro e sudarsi il posto ogni giorno! E quante cose mi hai offerto, quante cose mi hai insegnato!! A partire dall'accesso al Mondo 3 fino all'infinito..elencarli è impossibile!!!

Ragazzi, scherzi a parte, grazie per questi anni in cui mi avete risvegliato dai torpori dell'vecchiaia in cui ero precocemente caduto..grazie per avermi fatto ritrovare il sorriso e la dolce favella perchè sono entrato da Pierot..e ora sono rinato! Grazie a voi..sì VOI, VOI, VOI!

Alex, come esprimere il mio affetto nei tuoi confronti: sei un uragano, mi piace il tuo modo di affrontare le cose, sempre di petto, sempre con coraggio!

E poi la generosità e il senso di famiglia ti fanno una persona speciale! E “non mollare mai” è molto più che un semplice motto!!!

Grazie a Secondo Pantieri che mi ha trasmesso i più alti valori dello sport, valori che riapplicati alla vita di tutti i giorni mi hanno fatto da guida tra le innumerevoli salite e cadute di questi anni.

Per finire, volevo dire grazie a tutti i ragazzi Erasmus della Via Battuti Verdi, che in questi anni mi hanno fatto piacevolmente assaporare quell'Europa fino a prima praticamente sconosciuta..e ora mi è venuta fame! :-P

Grazie a tutti i professori di questa Università e ai professori di Leuven perchè in questi anni mi avete permesso di maturare come persona e (spero) come conoscenze! E' stata una battaglia dura, ma stimolante..certo che un pò di salita si potrebbe togliere! :-P

Ringrazio anche tutte le persone che non ho citato ma che hanno recitato un ruolo da comparsa: certe cose a volte si apprendono così velocemente!

Infine, “*..grazie alla vita, che mi ha dato tanto..*”!

Antonio Cauzo

