

# Programming SOA/WS Systems with BDI Agents and Artifact-Based Environments

Michele Piunti<sup>1</sup>   Andrea Santi<sup>1</sup>   Alessandro Ricci<sup>1</sup>

<sup>1</sup>Università degli studi di Bologna, DEIS - Cesena

Mallow-Awesome, 9/09/2009

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArTAgO-WS working environments
- 3 Case study
- 4 Conclusions

# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArtAgO-WS working environments
- 3 Case study
- 4 Conclusions

# Motivation and objective of our work

## The current situation

Agents and MAS are recognized as a suitable paradigm for engineering SOA/WS applications but in few works agent programming languages – in particular those based on a strong notion of agency such as BDI ones – are exploited for concretely developing such systems

## Our goal

We aim to define a programming model for developing SOA/WS applications based on:

- Jason BDI agents [Bordini et al., 2006, Bordini et al., 2007]
- Artifact-based environments built on top of the CArTAgO-WS platform

# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArtAgO-WS working environments
- 3 Case study
- 4 Conclusions

# SOA/WS applications in A&A models

SOA realized in CArtaGO-WS working environment where a set of Jason agents work together and interact creating, sharing and exploiting a dynamic set of artifacts

- **BDI Jason Agents** encapsulate the execution and the control of the business activities/tasks that characterize the SOA scenario
  - Discovery, orchestration, coordination, composition ...
- **Working environments** used as a first class abstraction for:
  - Encapsulate the business resources and tools needed by agents to operate in the application domain
  - Allow the design of a world aimed at the agent's use

# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArtAgO-WS working environments
- 3 Case study
- 4 Conclusions

# The pivotal role of the BDI agents 1/3

## The idea

The introduction of BDI agents into a agent-based SOA/WS programming model can be used for tackle and manage a lot of complexities and high-level aspects of SOA/WS applications

## Benefits relates to the use of abstractions provided by the BDI model

High-level abstractions such as goals and plans allow

- An ad-hoc decomposition and structurization of the business logic
  - through plans hierarchy, definition of long term-activity, plans/sub-plans etc.
  - concepts that are widely used in BPM [Burmeister et al., 2008]
- Easier business process modeling - relates to a WS - and seamless integration to practical implementation

# The pivotal role of the BDI agents 2/3

- The WS's behavior is governed through practical reasoning
  - Pro-activeness (towards goals and task achievement)
  - Adaptive and flexible behavior of the WS's towards changes into an uncertain environment
    - new events
    - change in context informations
  - Using practical reasoning agents have the capability to perform different course of actions execute due to context informations or their mental attitudes
- Reactive (even-driven) behavior
  - such in the case of Event-Driven Architectures (EDA), which are meant to be a main aspect of forthcoming SOA

# The pivotal role of the BDI agents 3/3

Errors and unexpected situations can be managed in an easier and straightforward way

- Plan failure are directly manageable in BDI languages such as Jason
  - the course of action to follow due to failure can be specified in the same way as the normal activities/tasks the agent has to perform
- Proper repair plans can be spawned as soon as a failure is detected
- Developer can manage smoothly failure policies at design time

# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CARTAgO-WS working environments
- 3 Case study
- 4 Conclusions

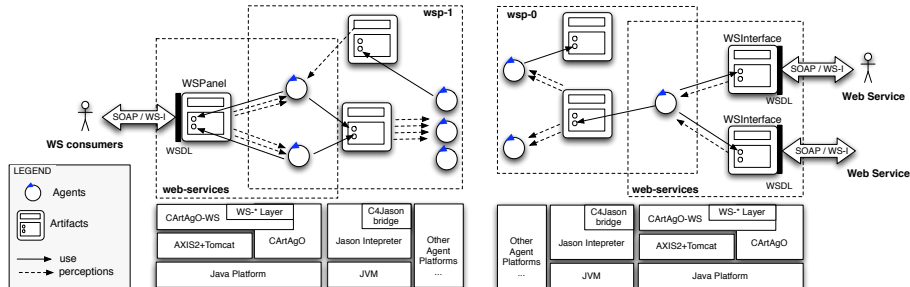
# CARTAgO platform / infrastructure

- The CARTAgO platform
  - Provides a runtime environment for executing (possibly distributed) artifact-based environments
  - Java-based programming model for defining artifacts
  - Set of basic API for agent platforms to work within artifact-based environment
  - Integration with agent programming platforms via ad-hoc bridges (C4Jason, C4Jadex, ...)
- Distributed and open MAS
  - Workspaces distributed on Internet nodes
  - Agents can join and work in multiple workspace at a time
  - Role-Based Access Control (RBAC) security model
- Open-source technology
  - available at <http://cartago.sourceforge.net>

## Agent based SOA in CArtAgO-WS

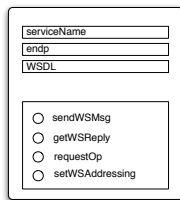
A CArtAgO-WS environment is instrumented with three kind of artifacts:

- Basic types artifacts that manages the Agent/WS interaction
- Artifacts that agents could exploit for get access to WS-\* functionalities
- Business level/application artifacts

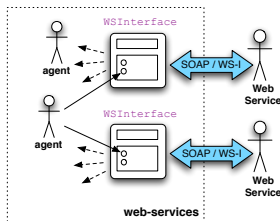


# The WSInterface Artifact

- Used for interact with an existing WS
- Instantiated specifying a set of parameters of the WS to interact with: the WSDL, the port-type ...
- Provides a set of operations to:
  - Send messages to the designed WS (sendWSMsg)
  - Get the replies to messages (getWSReply)
  - Support basic MEPs (i.e. request-response requestOp)
  - Configure the artifact with WS-\* specification's informations (i.e. setWSAddressing)

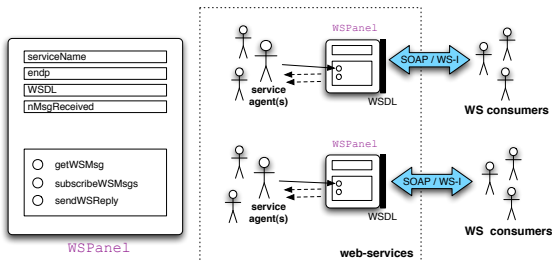


WSInterface



# The WSPanel Artifact

- Used for providing and set up a new WS
- Instantiated specifying the WS's WSDL
- Provides a set of operations to:
  - Retrieve or be notified of requests/messages arrived at the WS (getWSMsg and subscribeWSMsgs)
  - Send replies (sendWSReply)
  - Configure the artifact with WS-\* specification's informations



# WS-\* layer

## The SOA/WS evolution

The second generation of WS has introduced a set of specification (WS-\*) for the managing of advanced functionalities:

- WS-Coordination, WS-Security, WS-Trust, WS-ReliableMessaging ...

## WS-\* layer fundamentals

Provide a straightforward and uniform way for developers and agents, to get access, and manage, all the processing related to a WS's specifications thanks to a specification-independent architectural model

- All the functionalities of the WS-\* layer can be exploited simply using two different kind of artifact:
  - The `WSRequestMediator` (**RM**) artifact
  - The `Wallet` (**WA**) artifact

# WSRequestMediator and Wallet in a nutshell

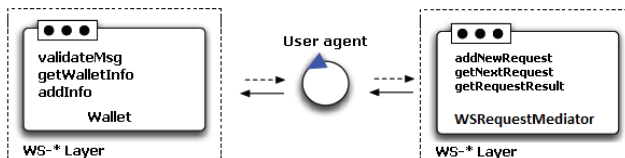
## WSRequestMediator (RM)

The **RM** mediates the interactions between the WS-\* layer and its utilizers in a request-response interaction: multiple **RM** can be instantiated to provide different functionalities

## Wallet (WA)

The **WA** is a sort of personal data container exploitable by agents for:

- Store/retrieve any kind of specification-based information
- Dynamically configure a `WSInterface`/`WSPanel` with the information contained in it



# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArtAgO-WS working environments
- 3 Case study
- 4 Conclusions

# Scenario's overview

## The scenario

A typical example used in SOA/WS contexts: a client WS wants to book an holiday by exploiting a series of WSs providing the required resources as hotel reservation, transport facilities, payment

## The actors

- The Booking Requestor (**BR**): the WS that wants to make an holiday booking for a specified period
- A set of WSs that compose the booking application (**BA**)
  - The Hotel Service (**HS**)
  - The Transport Service (**TS**)
  - The Payment Service (**PS**)

# Booking dynamics

The entire booking dynamics are managed through a WS-AtomicTransacion (**WS-AT**, one of the complex activities that can be coordinated on top of the WS-Coordination specification)

## Booking dynamics detail

- The Booking Requestor **BR** must contact each WS that compose the Booking Application **BA**
- If every WS reply to the **BR**'s request with the desired response message the **WS-AT** can be **committed** and then the holiday can be considered successfully booked
- Otherwise, if some wrong response is retrieved by the **BR**
  - The **BR** is forced to rollback the **WS-AT**
  - Ad-hoc failure handling can be performed...

# Hotel Service (HS) overview

## Functionalities provided

The Hotel Service (**HS**) manages the hotel's booking tasks and also provides notification functionalities to interested subscribers

## Implementation

The **HS** has been designed using two specialized agents, sharing and exploiting an instance of `WSPanel` used to expose the service

**Hotel Basic Agent** manages the requests related to bookings and cancellations, exploiting to this end the functionalities provided by an `HotelBookingRegistry` artifact

**Hotel Notifier Agent** manages the **HS**'s notification functionalities: it uses a `SubscribersMap` artifact to keep track of the subscriptions requested and monitors the `HotelBookingRegistry` so as to notify interested subscribers as soon as changes regarding date availabilities are observed

# Transport Service (TS) and Payment Service (PS)

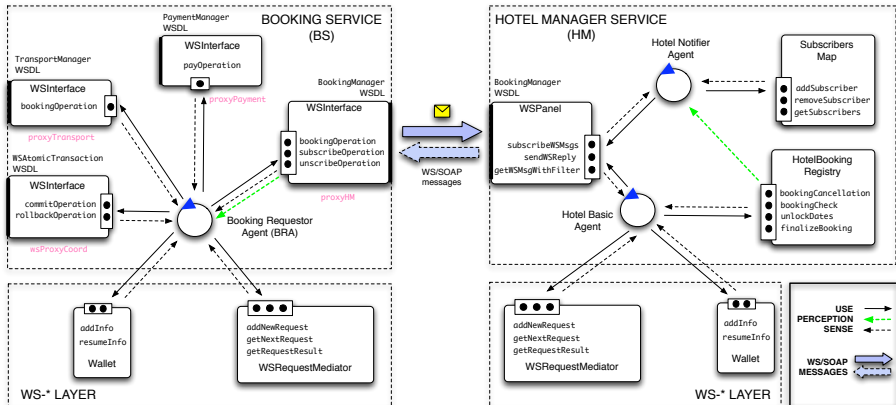
## The Transport Service

The **TS** is the WS that manages the booking for the transport used for arrive and leave from the designed destination. It maintains the transport's availability into its private registry

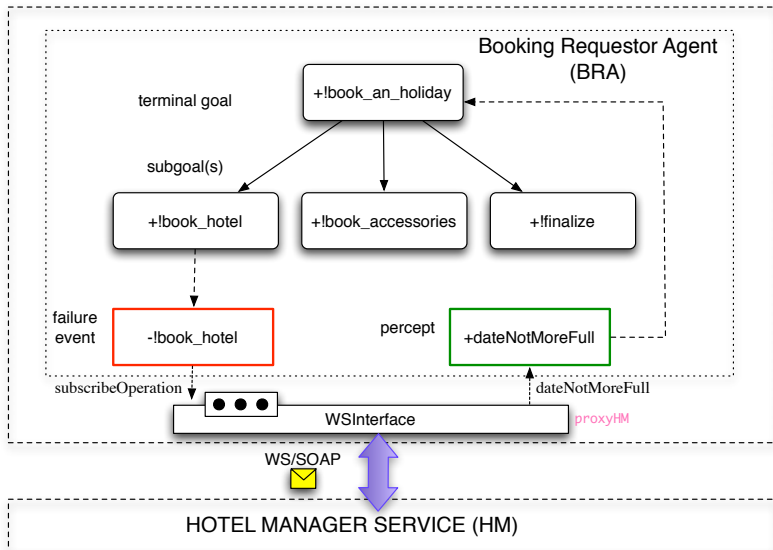
## The Payment Service

The **PS** manages bank accounts and can be used for settle payment requests. Obviously a payment request can succeed only and only if the specified bank account contains enough money

# Part of the scenario's structural architecture



# The Booking Requestor Agent business logic



# Cutout of the BR Agent 1/3

```

00  +!start_booking
01  <- !setupTools;
02      !retrieveDate;
03      !book_an_holiday.
04
05  /* Set up of the required artifacts (only if not already created)
06  for the communication with the designed WSs */
07  +!setupTools : true
08  <- !locate_artifacts;
09      //Use the RM to request a new WS-AT and add the related WS_AT_Context into the Wallet
10      ...
11      cartago.use(Wallet, addInfo(WS_AT_Context));
12      !makeInterface(proxyHM, "http://webservices.hotel.com/.../BookingManager.wsdl");
13      ?artifact_id(proxyHM, ProxyID);
14      cartago.use(ProxyID, configure(WS_AT_Context)).
15      ... //Further artifact constructions/configurations
16
17  //Top level goal
18  +!book_an_holiday : date(Dates)
19  <- !book_hotel(Dates, Res_Hotel);
20      !book_accessories(Dates, Res_Acc);
21      !finalize(Res_Hotel, Res_Acc).
22
23  +!book_hotel(Dates, Res_Hotel) : artifact_id(proxyHM, ProxyID)
24  <- !createBookingMessage(hotelBooking, Dates, MsgBookHotel);
25      cartago.doRequestResponse(ProxyID, bookingOperation(MsgBookHotel), HotelResponse);
26      !inspect_hotel_response(HotelResponse, Res_Hotel);
27      //The plan fails if the hotel is not available for the booking
28      Res_Hotel == "available".

```

# Cutout of the BR Agent 2/3

```

29  +!book_accessories(Dates,Res_Acc) : artifact_id(proxyTransport,TranID) & artifact_id(proxyPayment,PayID)
30    & hPrice(HotelPrice) & tPrice(TransportPrice) & bank_account_id(BankID)
31  <- !createBookingMessage(transportBooking, Dates, MsgTransport);
32    cartago.doRequestResponse(TranID, bookingOperation(MsgTransport), ResponseTransport);
33    !createPayMessage(BankID, (HotelPrice+TransportPrice), MsgPay);
34    cartago.doRequestResponse(PayID, payOperation(MsgPay), ResponsePayment);
35    !inspect_acc_responses(ResponseTransport, ResponsePayment, Res_Acc);
36    //The plan fails if some bad response has been retrieved
37    Res_Acc == "available".
38
39  /* Fail event handling due to hotel's unavailability.
40     The BR could subscribe itself for notifications regarding further hotel's availability */
41  -!book_an_holiday : artifact_id(proxyHM, ProxyID) & dates(Dates) & Res_Hotel /= "available"
42  <- !createSubscribeMessage(Dates, MsgSubscription);
43    cartago.focus(ProxyID);
44    cartago.doOneWay(ProxyID, subscribeOperation(MsgSubscription));
45    !finalize("not_available", "").
46
47  //Fail event handling due to accessories problems: no subscription can be done
48  -!book_an_holiday : Res_Hotel == "available" & Res_Acc /= "available"
49  <- !finalize("available", "not_available").

```

# Cutout of the BR Agent 3/3

```

50 //Notification received from HS: the holiday booking can be retried
51 +dateNotMoreFull(Dates) [source(ProxyID)] : artifact_id(proxyHM, ProxyID) & dates(Dates)
52 <- cartago.stopFocusing(ProxyID);
53   !start_booking.
54
55 //Booking finalization: commit or rollback can be performed due to context informations
56
57 //Both the hotel and the transport has been booked and the payment has been done: transaction commit
58 +!finalize(Res_Hotel, Res_Acc) : Res_Hotel == "available" & Res_Acc == "available"
59   & wallet_entry(atContext, WS_AT_Context) & artifact_id(wsProxyCoord, CoordID)
60 <- !createCommitMessage(WS_AT_Context, MsgCommit);
61   cartago.doOneWay(CoordID, commitOperation(MsgCommit)).
62
63 //Something wrong has happened: transaction rollback
64 +!finalize(Res_Hotel, Res_Acc) : (Res_Hotel /= "available" | Res_Acc /= "available")
65   & wallet_entry(atContext, WS_AT_Context) & artifact_id(wsProxyCoord, CoordID)
66 <- createRollbackMessage(WS_AT_Context, MsgRollback);
67   cartago.doOneWay(CoordID, rollbackOperation(MsgRollback)).

```

# Outline

- 1 Introduction
- 2 An agent-based SOA/WS programming model
  - BDI agents
  - CArtAgO-WS working environments
- 3 Case study
- 4 Conclusions**

# Drawbacks and future works

## Problem and drawbacks

Agent languages - like Jason used in our examples - currently don't provide the same set of functionalities and facilities provided by OO mainstream languages. Therefore aspects like the agile managing of complex data structure (like the XML-based messages exchanged in SOA/WS applications) are not trivial at all

## Future works

- Definition of a specific model for the organization of the business logic of the WS
  - works like [Burmeister et al., 2008] can provide a good background
- Semantic-based WS composition, discovery and orchestration [van Riemsdijk and Wirsing, 2007]
- Extensions of the WS-\* specification supported by the WS-\* layer
  - Actually only WS-Addressing and WS-Coordination

# Bibliography I



Bordini, R., Hübner, J., and Wooldridge, M. (2006).  
*Programming Multi-Agent Systems in AgentSpeak Using Jason*.  
John Wiley & Sons, Ltd.



Bordini, R., Hübner, J., and Wooldridge, M. (2007).  
BDI agent programming in AgentSpeak using Jason.  
*In Proceedings of 6th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI)*, volume 3900, pages 143–164.  
Springer.



Burmeister, B., Arnold, M., Copaciu, F., and Rimassa, G. (2008).  
BDI-agents for agile goal-oriented business processes.  
*In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pages 37–44. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC.

# Bibliography II

-  van Riemsdijk, M. and Wirsing, M. (2007).  
Using Goals for Flexible Service Orchestration.  
*Lecture Notes in Computer Science*, 4504:31.