Design of SOA/WS applications using an Agent-based approach

Andrea Santi[1] Michele Piunti[1]

[1]Università degli studi di Bologna, DEIS - Cesena

Design of SOA/WS applications using a Agent-based approach, 12/06/2009

Outline

The need for a SOA/WS programming model

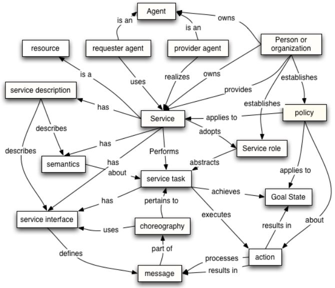
- SOA is a promising solution for building complex distributed systems, **but does not provide** a unifying programming model to support the core system design and development
- Actually the mainstream proposals are either object-oriented or component-oriented
    - In our opinion this models are deeply inadequate for the design of complex SOA system

Rationale

Agent concepts and technologies could be fruitfully exploited to define a general-purpose programming model to support the design, development, management of complex SOA/WS systems

Agent-based SOA/WS: Why?

- Agents and MAS are more and more recognised as a suitable paradigm for engineering SOA/WS systems[Hunhs, 2006, N. Huhns et al., 2005, Greenwood and Calisti, 2004]
- Actually agents *are* in service-oriented model described in the official W3C's document [Booth et al., 2004]:



A convergence of abstractions: Autonomy, Loose coupling, Message-based interactions

Outline

Complex Software Systems: Towards a Paradigm Shift

Programming in the large

Nextcoming computing will be:

- Anywhere: distributed, embedded in every environment item/object, cuncurrently operating in information-rich environments
- Always connected and always active: wireless technologies will make interconnection pervasive

Which impact on the design & development of software systems?

- Quantitative: in terms of computational units, software components, number of interconnections... **current processes, methods and technologies do not scale**
- Qualitative: new software systems are different in kind, will be introduced new features never modelled before

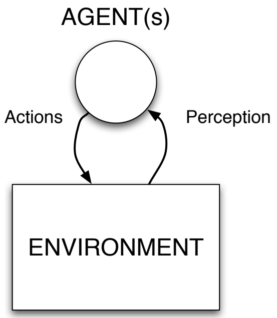
Requirements of Complex Software Systems

- Situatedness
  - Computations occur within an environment
  - Computations and environment mutually affect each other, and cannot be understood separately
- Openness
  - Systems and technologies are heterogenous *but* interoperable
- Locality in control
  - Components of a system are autonomous wrt other entities
  - Proactive loci of control
- Locality in interaction
  - Components of a system interact on a local basis based on some notion of spatio-temporal compresence
  - partial knowledge

Agents I

Intelligent Agents [Russell and Norvig, 2002]

An agent is anything that can be viewed as perceiving its ***environment*** *through sensors and acting upon that environment through effectors.*



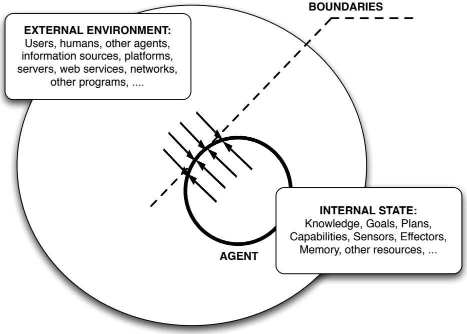
Agents II

Intelligent Agents [IBM, 1995]

*Software **entities** that carry out some set of **operations** on behalf of a user or another program with some degree of independence or **autonomy** and in so doing employ some **knowledge** or representation of user's **goals** or desires.*

Autonomy as Foundational Notion of Agency

- Agents are autonomous as they encapsulate (the processes of) their control and **the rules to govern it**
- Agents have no interface, cannot be controlled, nor can they be invoked
- Control does not pass through agent boundaries only information (readable data/events, messages) crosses agent boundaries


BOUNDARIES
EXTERNAL ENVIRONMENT: Users, humans, other agents, information sources, platforms, servers, web services, networks, other programs, ...
INTERNAL STATE: Knowledge, Goals, Plans, Capabilities, Sensors, Effectors, Memory, other resources, ...
AGENT

Other Relevant Agent Properties

Proactivity

- agents are designed with the aim to let them autonomously realize some objective (goal states)
- *making something happen*, rather than waiting for something to happen

Situatedness

- Any "ground" model of action is strictly coupled with the context where the action takes place
- Any agent is then located in the environment where it lives and (inter)acts

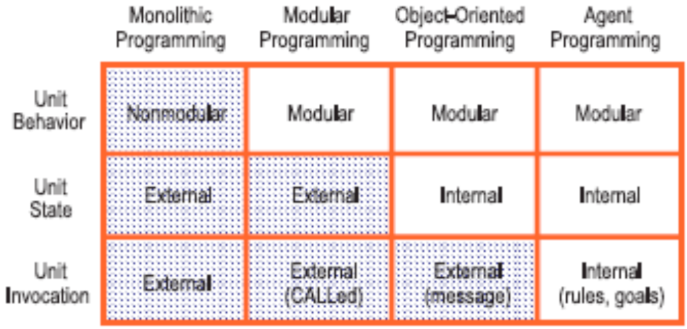
And many others

Regarding the context of interest a lot of other features can be considered

- Mobility, Intelligence, sociality, learning...

Evolution of Programming Languages: The Picture

- Odell, 2002[Odell., 2002]



	Monolithic Programming	Modular Programming	Object-Oriented Programming	Agent Programming
Unit Behavior	Nonmodular	Modular	Modular	Modular
Unit State	External	External	Internal	Internal
Unit Invocation	External	External (CALLed)	External (message)	Internal (rules, goals)

Belief Desire Intention model of Agency

- Belief Desire Intentions (BDI) is a model of agency mimicking intelligent mental attitudes.
- Agents are conceived, designed and then programmed along with their *mental states*:

Beliefs	current (internal) state of the agent: beliefbase stores relevant information about environment, exogenous entities and other agents;
Goals	state that the agent desires to achieve and about which he brings about (Practical Reasoning) based on processing of internal and external stimuli;
Plans	recipes of procedural *means* the agent has in repertoire to *change the world* and thus achieve its goals

BDI Agents in Jason

A Jason[1] plan has the following general structure:

```
triggering_event : context <- body.
```

where:

- The triggering event denotes the events that the plan is meant to handle;
- The context represents the circumstances in which the plan can be used;
    - logical expression, typically a conjunction of literals to be checked whether they follow from the current state of the belief base (Belief Formulae)
- The body is the course of actions to be executed to handle the event if the context is believed true at the time a plan is being chosen to handle the event.
    - A sequence of actions may initiate (sub) goals to achieve the root goal

[1] Jason [Bordini et al., 2007] programming platform is available at: `jason.sourceforge.net/`

Example: a BDI Agent in Jason

```
 /* Initial Beliefs */
likes(radiohead).
resource(proxyUS, "http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl").
resource(proxyUK, "http://webservices.amazon.com/AWSECommerceService/UK/AWSECommerceService.wsdl").
 /* Initial Goal */
!look_for.

/* Goal Event */
+look_for  :  likes(Name)
  <-  !doRequest(Name).

/* Sub-Goal to make a query to web resources */
+!doRequests(Query) : resource(Loc, URL)
  <-  !prepareMsg(Query, URL, Msg);
  .println("Doing request: ", Msg, "on resource: " , Loc);
  !execute_query(Msg);
  -resource(Loc, _);
  !doRequests(Query).

/* Reaction to reply Messages */
+reply(Msg,_) [source("proxyUS")] : true
  <-  !process(Msg, Res);
  .print("Answer from US ",  Res);
+reply(Msg,_) [source("proxyUK")] : true
  <-  !process(Msg, Res);
  .print("Answer from UK ",  Res);

+!process(Msg, Res)
  <- // retrieve Res from Msg ...
```

Example: a BDI Agent in Jason

```
 /* Initial Beliefs */
likes(radiohead).
resource(proxyUS, "http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl").
resource(proxyUK, "http://webservices.amazon.com/AWSECommerceService/UK/AWSECommerceService.wsdl").
 /* Initial Goal */
!look_for.

/* Goal Event */
+look_for  :  likes(Name)
  <-  !doRequest(Name).

/* Sub-Goal to make a query to web resources */
+!doRequests(Query) : resource(Loc, URL)
  <-  !prepareMsg(Query, URL, Msg);
  .println("Doing request: ", Msg, "on resource: " , Loc);
  !execute_query(Msg);
  -resource(Loc, _);
  !doRequests(Query).

/* Reaction to reply Messages */
+reply(Msg,_) [source("proxyUS")] : true
  <-  !process(Msg, Res);
  .print("Answer from US ",  Res);
+reply(Msg,_) [source("proxyUK")] : true
  <-  !process(Msg, Res);
  .print("Answer from UK ",  Res);

+!process(Msg, Res)
  <- // retrieve Res from Msg ...
```

Agents to Web Services Back and Forth

- Besides Agent Comunication Languages (i.e. Agents to Agents message passing), agent based technolgies and metodologies provide a weak support for interaction with other environmental resources:
  - In particular, in the case of Web Services
- Typical solutions to integrate agents in WS applications make use of gateway agents [Greenwood et al., 2007, Nguyen and Kowalczyk, 2007, Shafiq et al., 2005, Poggi et al., 2007, Greenwood and Calisti, 2004, Varga and Hajnal, 2003]:
  - that aims at translating SOAP messages in an agent readable format
- In so doing some of the native agent capabilities are simply neglected:
  - Good Agent Oriented Support would provide *native* support for goal orientedness, state persitency, fault tolerance, situatedness and context awareness, adaptiveness, event handling etc.

Agents and Web Services: Related Works

Other approaches proposed intelligent agents to address specific problems:

- using BDI reactive planning to control *web-service invocation* [Dickinson and Wooldridge, 2005]
- agent behavior for deisign of *agile business processes* [Burmeister et al., 2008]
- *goal oriented business processes* [Rimassa et al., 2008]
- *goal oriented service orchestration* [van Riemsdijk and Wirsing, 2007]
- logic agents able to reason about WS though *protocol analysis* [Casella and Mascardi, 2006]
- semantic web, planning and *service composition* [Cardoso and Sheth, 2003, Pistore et al., 2005, Lécué et al., 2008]
- *service discovery*: adatptive systems shaped in terms of BDI that discover knowledge concerning how to solve a specific set of problems [Bozzo et al., 2005, Georgeff, 2006]

Outline

Agents and Artifacts in the A&A Meta-model

Artifacts

Besides Agents, a MAS in A&A meta-model is built using a dual abstraction: the **Artifact**.

- An A&A artifact is a computational entity, with a set of embodied functionalities, and readable information aimed at *beeing functionally exploited by agents*

- Artifacts are non-autonomous entities, designed to serve some agent's purpose
  - not to proactively achieve their own goals
- An artifact is a target, a tool, an external resource in the *hands* of agents:
  - Enlarging repertoire of agents available actions
  - Providding exogenous information that can be suitalby read by agents
  - An artifact does not need to be self-controlled, it just has to be *governed* by agents when they use it

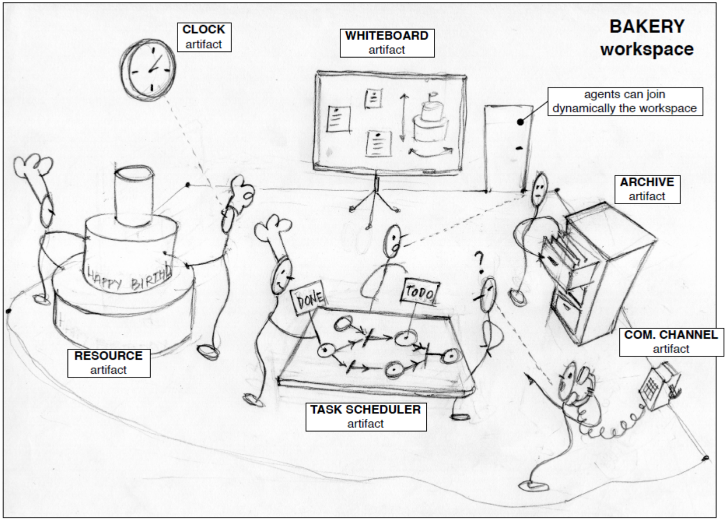
The A&A meta-model

A&A: A conceptual framework for MAS modelling & engineering

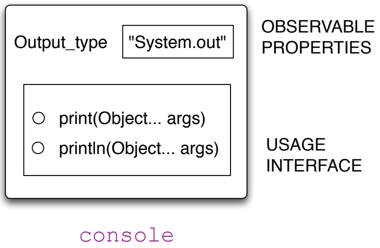
According to the A&A meta-model, a Mutli Agent System consists in a number of entities built upon the following three basic abstractions: [Omicini et al., 2008]

Agents	represent pro-active components of the systems, encapsulating the autonomous execution of some kind of activities inside some sort of environment
Artifacts	represent passive components of the systems such as resources and media that are intentionally constructed, shared, manipulated and used by agents to support their activities, either cooperatively or competitively
Workspaces	are the conceptual containers of agents and artifacts, useful for defining the topology for the environment and providing a notion of locality in work environments

Agents & Artifacts model: basic idea in a picture



Artifact Computational Model in CArtAgO



CArtAgO IMPLEMENTATION OF
ARTIFACT CONSOLE:

```
package alice.cartago.util;
import alice.cartago.*;

public class Console extends Artifact {

     @OPERATION void init(){
        defineObsProperty("Output_type","System.out");
     }

     @OPERATION void print(Object arg){
        internalPrint(arg);
     }

     @OPERATION void println(Object arg0){
        internalPrintln(arg);
     }

     private void internalPrint()   { ... }
     private void internalPrintln() { ... }
}
```

JASON AGENT USING CONSOLE ARTIFACT:

```
+!say_hello : true
<- cartago.joinWorkspace("web-services","localhost");
   cartago.lookupArtifact("console", ConsoleID);
   cartago.observeProperty(ConsoleID, "Output_type", Type);
   !write(Type).
```

```
+!write("System.out") : true
<-  cartago.use(ConsoleID, println("Hello,", "World!")).

+!write(_) : true
<-  .print("Hello,", "World!")).
```

Basic Actions for interacting in CArtAgO Working Environments

(1)	`joinWorkspace(+Workspace`*`[,Node]`*`)`
(2)	`quitWorkspace`
(3)	`makeArtifact(+Artifact,+ArtifactType`*`[,ArtifactConfig]`*`)`
(4)	`lookupArtifact(+ArtifactDesc,?Artifact)`
(5)	`disposeArtifact(+Artifact)`
(6)	`use(+Artifact,+UIControl(`*`[Params]`*`)`*`[,Sensor] [,Timeout] [,Filter]`*`)`
(7)	`sense(+Sensor,?PerceivedEvent`*`[,Filter] [,Timeout]`*`)`
(8)	`focus(+Artifact`*`[,Sensor] [,Filter]`*`)`
(9)	`stopFocussing(+Artifact)`
(10)	`observeProperty(+Artifact,+Property,?PropertyValue)`

Table: Basic agent actions managing workspaces (1–2), creating, disposing and looking up artifacts (3–5), using artifacts (6–7), and observing artifacts (8–10). Syntax is expressed in a logic-like notation, where italic items in square brackets are optional [Ricci et al., 2008].

Outline

Sahping SOA/WS applications in A&A

SOA modelled in one or multiple workspaces where a (possibly open and heterogeneous) set of agents work together and interact both via direct communication and by producing, consuming, sharing and cooperatively using a dynamic set of artifacts

- **Agents** encapsulate the responsibility of the execution and control of the business activities and tasks that characterise the SOA-specific scenario,
  - business logic of service providers/requestors is encapsulated in the application by agents
  - task as service discovery, orchestration, coordination, composition etc. are delegated to agents
- **Artifacts** encapsulate the business resources and tools needed by agents to operate in the service application domain.
  - exploited (also) to model and engineer those parts encapsulating Web Services, to be used, changed and adapted by agents on the need.
  - encapsulate and govern the *message processing logic*, which typically includes header processing, validation, and possible transformation

Sahping SOA/WS applications in A&A

SOA modelled in one or multiple workspaces where a (possibly open and heterogeneous) set of agents work together and interact both via direct communication and by producing, consuming, sharing and cooperatively using a dynamic set of artifacts

- **Agents** encapsulate the responsibility of the execution and control of the business activities and tasks that characterise the SOA-specific scenario,
  - business logic of service providers/requestors is encapsulated in the application by agents
  - task as service discovery, orchestration, coordination, composition etc. are delegated to agents
- **Artifacts** encapsulate the business resources and tools needed by agents to operate in the service application domain.
  - exploited (also) to model and engineer those parts encapsulating Web Services, to be used, changed and adapted by agents on the need.
  - encapsulate and govern the *message processing logic*, which typically includes header processing, validation, and possible transformation

Sahping SOA/WS applications in A&A

SOA modelled in one or multiple workspaces where a (possibly open and heterogeneous) set of agents work together and interact both via direct communication and by producing, consuming, sharing and cooperatively using a dynamic set of artifacts

- **Agents** encapsulate the responsibility of the execution and control of the business activities and tasks that characterise the SOA-specific scenario,
  - business logic of service providers/requestors is encapsulated in the application by agents
  - task as service discovery, orchestration, coordination, composition etc. are delegated to agents
- **Artifacts** encapsulate the business resources and tools needed by agents to operate in the service application domain.
  - exploited (also) to model and engineer those parts encapsulating Web Services, to be used, changed and adapted by agents on the need.
  - encapsulate and govern the *message processing logic*, which typically includes header processing, validation, and possible transformation

Agent based SOA in CArtAgO-WS

CArtAgO-WS is a programming platform providing artifact based facilities that enable agents holding to heterogenous and distributed platforms to *natively* interact with Web Services in their Worksapeces.
In particular, a couple of programmable artifacts are provided:

- `WSPanel` artifact: used by provider agents for build up and set up a new WS:
    - Provides basic functionalities to manage service requests, including receiving and sending messages according to the specific MEP as described in the WSDL, and basic controls to configure security and reliability policies.
- `WSInterface` artifact: used by user agents for consuming existing WS:
    - Provides basic functionalities to interact with the specified Web Service, in particular to send messages for executing operations and to get the replies sent back by the service, according to the message exchange patterns defined in the WSDL and to the quality of service specified by the service policies (in particular, security and reliability).

CArtAgO-WS platform

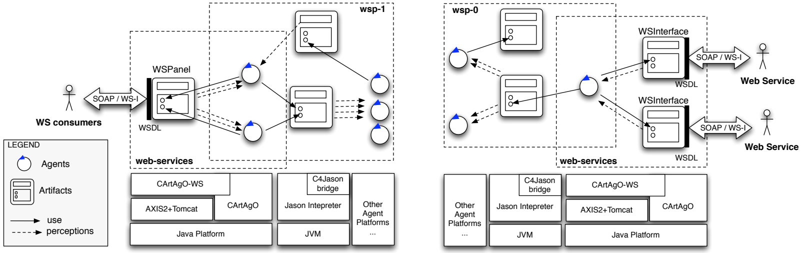
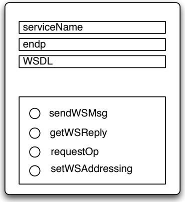


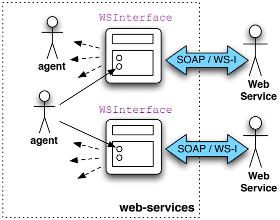
Figure: On the left, a CArtAgO-WS node running a Web Service, composed by two workspaces (`web-services` and `wsp-1`). In `web-services`, an instance of `WSPanel` artifact is shared and used by two agents to process WS requests and send replies. On the right, a CArtAgO-WS node running an application using existing Web Services. In `web-services` workspace two instances of `WSInterface` artifact are exploited by the same agent to interact (concurrently) with two distinct Web Services.

The `WSInterface` Artifact

- `WSInterface` is instantiated specifying the WSDL to interact with, name/port type (if the WSDL includes multiple port and services), a local endpoint to which the artifact receive messages (e.g. replies).
- Includes controls to send a message to the service (`sendWSMsg`), to get the reply to messages (`getWSReply`), and higher-level operations to directly support basic MEPs, such as the request-response (`requestOp`).
- Other operation controls configure the WS interaction (support for basic WS-* standards, i.e. WS-Addressing)

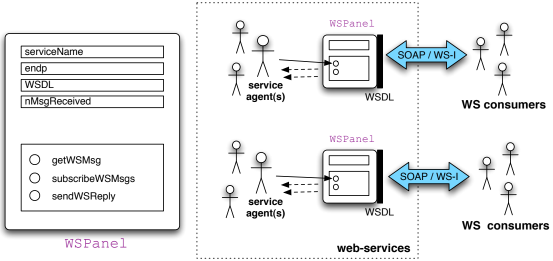


WSInterface

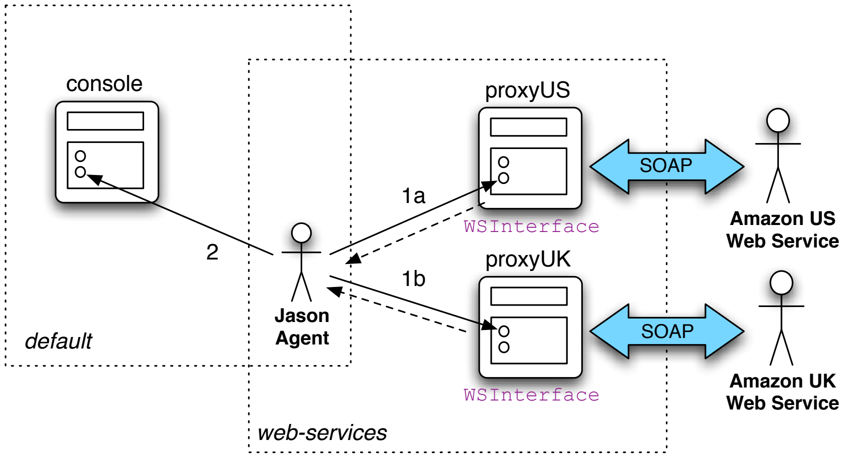


The `WSPanel` Artifact

- `WSPanel` is instantiated specifying a WSDL document for the service to be created
- Operation controls are provided to retrieve or be notified of requests/messages arrived at the Web Service (`getWSMsg` and `subscribeWSMsgs`), possibly specifying filters to select specific messages, and to send replies (`sendWSReply`).



Agents, Artifacts and WS in CArtAgO-WS Work Environments



Example: Jason agent exploiting WS in CArtAgO-WS

```
+look_for  : likes(Name)
  <-  cartago.joinWorkspace("web-services","localhost");
  !makeInterface(proxyUS,"http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl");
  !makeInterface(proxyUK,"http://webservices.amazon.com/AWSECommerceService/UK/AWSECommerceService.wsdl")
  // initialise proxies  (i.e., use WSInterface setWSAddressingSupport, setWSHeaderSupport operations)
  !doRequests.
/* Create a WSInterface related to WSDLURI */
+!makeInterface(Name,WSDLURI) : true
  <-  cartago.makeArtifact(Name,"alice.cartagows.WSInterface",[WSDLURI],Proxy);
  .print("service-interface ",Name," built.").

+!doRequests(Query) : resource(Loc, URL)
  <-  !prepareMsg(Query, URL, Msg);
  .println("Doing request: ", Msg, "on resource: " , Loc);
  cartago.use(Loc,requestOp("ItemSearch",Msg));        //no sensor is provided            (1.a & 1.b)
  -resource(Loc, _ );
  !doRequests(Query).

/* In CArtAgO-WS, artifact's observable events are signalled to the agents as +EventDescr[source(art_name)] */
+reply(Msg, _ )[source("proxyUK")] : true
  <-  .print("Answer from UK ",Msg);
  cws.buildDOM(Msg,Elem);
  cws.getDOMElemValue(Elem,"ItemSearchResponse.Items.TotalResults",Value);
  cartago.use("console@default",println("Total hits in UK Web Service: ",Value)).       (2)

+reply(Msg,_) [source("proxyUS")]: true
  <-  .print("Answer from US ",Msg);
  ....
```

Outline

Considerations

SOA/WS evolution

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

WS-Coordination	provides the rules for coordinate complex activities (AtomicTransactions , BusinessActivities) between WSs
WS-Security framework	is a set of security specifications that provides authentication, authorization, data integrity and a so on...
WS-Trust	allows a WS to advertise its policies (on security, QoS..)

Consequences?

A platform for building SOA/WS application worthy of the name should:

- Provides support for the new specifications introduced
- Enable a developer an easy method to get access and use the functionalities provided by the WS-* specifications

The WS-* layer

The objective

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

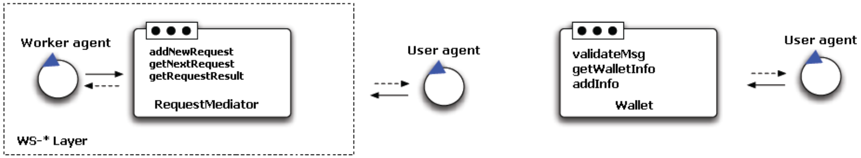
WS-* layer fundamentals

- All the functionalities of the WS-* layer can be exploited simply using two different kind of artifact:
  - The `RequestMediator` (**RM**) artifact
  - The `Wallet` (**WA**) artifact
- Proper **WorkerAgents** contained into the WS-* layer, working behind **RM** artifacts and not visible to application agents, are responsible of the managing and the material execution of the processing relates to the requests

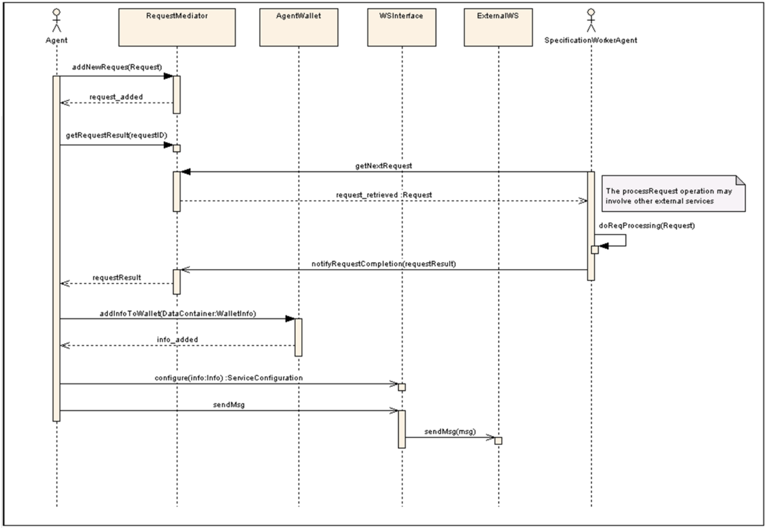
RequestMediator and Wallet in a nutshell

RequestMediator (RM) and Wallet (WA)

- The **RM** mediates the interactions between the WS-* layer and its utilizers in a request-response based interaction
- The **WA** is a sort of personal data container that the agents can exploits for:
  - Store/retrieve any kind of specification-based information
  - Dynamically configure a `WSInterface`/`WSPanel` with the information contained in it



WS-* layer dynamics


Agent
RequestMediator
AgentWallet
WSInterface
ExternalWS
SpecificationWorkerAgent
addNewReques(Request)
request_added
getRequestResult(requestID)
getNextRequest
request_retrieved :Request
The processRequest operation may involve other external services
doReqProcessing(Request)
notifyRequestCompletion(requestResult)
requestResult
addInfoToWallet(DataContainer:WalletInfo)
info_added
configure(info:Info) :ServiceConfiguration
sendMsg
sendMsg(msg)

Outline

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 date
- 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...

Transport Service (TS) and Payment Service (PS)

The Transport Service

The **TS** is the WS that manges 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

Hotel Service (HS) overview

Functionalities provided

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

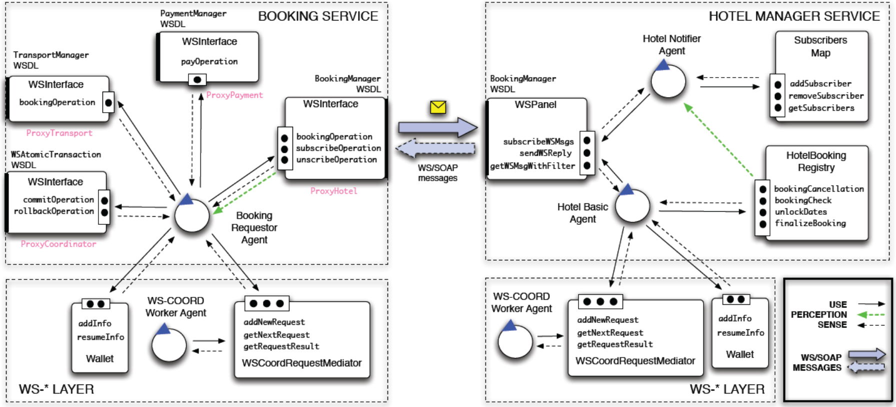
Implementation

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

Part of the scenario's structural architecture



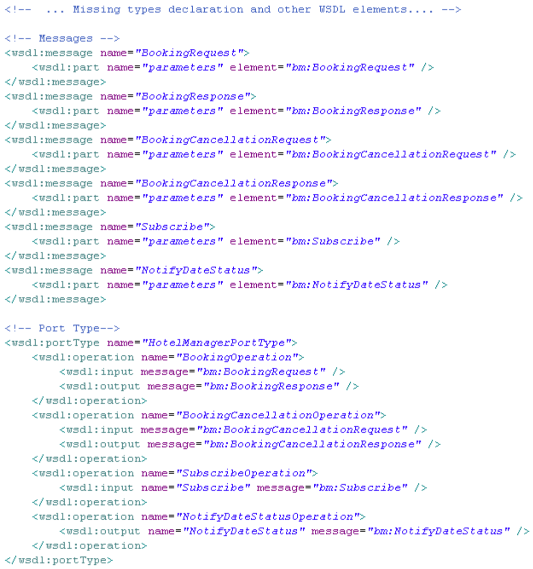
A generic WS-AT Participant's WSDL

```xml
<!--  ... Missing types declaration and other WSDL elements.... -->

<!-- Messages -->
<wsdl:message name="Prepare">
    <wsdl:part name="parameters" element="wsat:Prepare" />
</wsdl:message>
<wsdl:message name="Commit">
    <wsdl:part name="parameters" element="wsat:Commit" />
</wsdl:message>
<wsdl:message name="Rollback">
    <wsdl:part name="parameters" element="wsat:Rollback" />
</wsdl:message>

<!-- Port Type-->
<wsdl:portType name="ParticipantPortType">
    <wsdl:operation name="PrepareOperation">
        <wsdl:input message="Prepare" />
    </wsdl:operation>
    <wsdl:operation name="CommitOperation">
        <wsdl:input message="Commit" />
    </wsdl:operation>
    <wsdl:operation name="RollbackOperation">
        <wsdl:input message="Rollback" />
    </wsdl:operation>
</wsdl:portType>

<!-- Binding -->
<wsdl:binding name="ParticipantBinding" type="bm:ParticipantPortType">
    <wsdl:operation name="PrepareOperation">
        <soap:operation soapAction="http://docs.oasis-open.org/ws-tx/bm/2006/06/Prepare" />
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="CommitOperation">
        <soap:operation soapAction="http://docs.oasis-open.org/ws-tx/bm/2006/06/Commit" />
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="RollbackOperation">
        <soap:operation soapAction="http://docs.oasis-open.org/ws-tx/bm/2006/06/Rollback" />
        <!-- Missing Operation details -->
    </wsdl:operation>
</wsdl:binding>

<!-- Service -->
<wsdl:service name="GenericParticipant">
    <wsdl:port binding="bm:ParticipantBinding" name="ParticipantPort">
        <soap:address location="http://localhost:8080/axis2/services/GenericParticipant" />
    </wsdl:port>
</wsdl:service>
```

Sketch of the Payment Service's WSDL

```
<!--  ... Missing types declaration and other WSDL elements.... -->

<!-- Messages -->
<wsdl:message name="PaymentRequest">
    <wsdl:part name="parameters" element="pm:PaymentRequest" />
</wsdl:message>
<wsdl:message name="PaymentResponse">
    <wsdl:part name="parameters" element="pm:PaymentResponse" />
</wsdl:message>

<!-- Port Type-->

<wsdl:portType name="PaymentManagerPortType">
    <wsdl:operation name="PayOperation">
        <wsdl:input message="pm:PaymentRequest" />
        <wsdl:output message="pm:PaymentResponse" />
    </wsdl:operation>
</wsdl:portType>

<!-- Binding -->
<wsdl:binding name="PaymentManagerPortTypeBinding" type="pm:PaymentManagerPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
    <wsdl:operation name="PayOperation">
        <soap:operation
            soapAction="http://localhost:8080/wsdl/PaymentManager/PayOperation" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>


<!--  Service -->
<wsdl:service name="PaymentService">
    <wsdl:port binding="pm:ParticipantBinding" name="ParticipantPort">
        <soap:address location="http://localhost:8080/axis2/services/PaymentService" />
    </wsdl:port>
</wsdl:service>
```

Sketch of the Hotel Service's WSDL 1/2

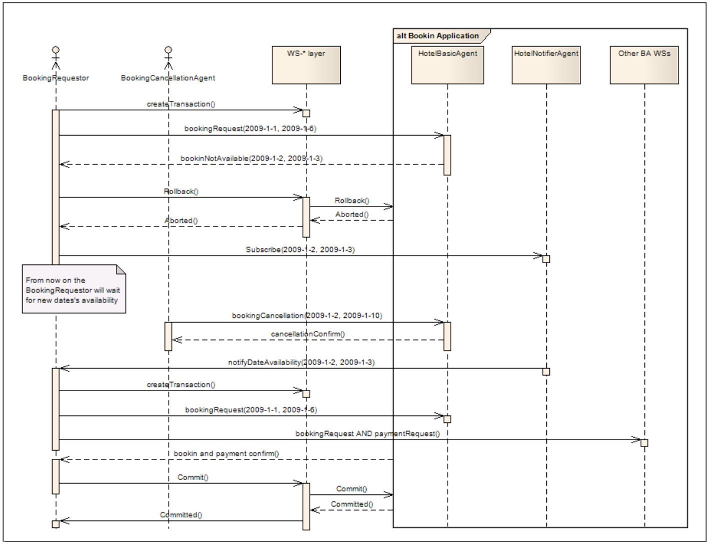


```
<!--  ... Missing types declaration and other WSDL elements.... -->

<!-- Messages -->
<wsdl:message name="BookingRequest">
    <wsdl:part name="parameters" element="bm:BookingRequest" />
</wsdl:message>
<wsdl:message name="BookingResponse">
    <wsdl:part name="parameters" element="bm:BookingResponse" />
</wsdl:message>
<wsdl:message name="BookingCancellationRequest">
    <wsdl:part name="parameters" element="bm:BookingCancellationRequest" />
</wsdl:message>
<wsdl:message name="BookingCancellationResponse">
    <wsdl:part name="parameters" element="bm:BookingCancellationResponse" />
</wsdl:message>
<wsdl:message name="Subscribe">
    <wsdl:part name="parameters" element="bm:Subscribe" />
</wsdl:message>
<wsdl:message name="NotifyDateStatus">
    <wsdl:part name="parameters" element="bm:NotifyDateStatus" />
</wsdl:message>

<!-- Port Type-->
<wsdl:portType name="HotelManagerPortType">
    <wsdl:operation name="BookingOperation">
        <wsdl:input message="bm:BookingRequest" />
        <wsdl:output message="bm:BookingResponse" />
    </wsdl:operation>
    <wsdl:operation name="BookingCancellationOperation">
        <wsdl:input message="bm:BookingCancellationRequest" />
        <wsdl:output message="bm:BookingCancellationResponse" />
    </wsdl:operation>
    <wsdl:operation name="SubscribeOperation">
        <wsdl:input name="Subscribe" message="bm:Subscribe" />
    </wsdl:operation>
    <wsdl:operation name="NotifyDateStatusOperation">
        <wsdl:output name="NotifyDateStatus" message="bm:NotifyDateStatus" />
    </wsdl:operation>
</wsdl:portType>
```

Sketch of the Hotel Service's WSDL 2/2

```
<!-- Binding -->
<wsdl:binding name="HotelManagerPortTypeBinding" type="bm:HotelManagerPortType">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />

    <wsdl:operation name="BookingOperation">
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="BookingCancellationOperation">
        <soap:operation
            soapAction="http://localhost:8080/wsdl/BookingManager/BookingCancellationOperation" />
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="SubscribeOperation">
        <soap:operation
            soapAction="http://localhost:8080/wsdl/BookingManager/SubscribeOperation" />
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="UnscribeOperation">
        <soap:operation
            soapAction="http://localhost:8080/wsdl/BookingManager/UnscribeOperation" />
        <!-- Missing Operation details -->
    </wsdl:operation>
    <wsdl:operation name="NotifyDateStatusOperation">
        <soap:operation
            soapAction="http://localhost:8080/wsdl/BookingManager/NotifyDateStatusOperation" />
        <!-- Missing Operation details -->
    </wsdl:operation>

    <!-- Missing similar operations binding -->

</wsdl:binding>


<!--  Service -->
<wsdl:service name="HotelService">
    <wsdl:port binding="bm:HotelManagerPortTypeBinding" name="BookingManagerPort">
        <soap:address location="http://localhost:8080/axis2/services/HotelService" />
    </wsdl:port>
</wsdl:service>
```

The Booking dynamics of the considered example


alt Bookin Application
BookingRequestor
BookingCancellationAgent
WS-* layer
HotelBasicAgent
HotelNotifierAgent
Other BA WSs
createTransaction()
bookingRequest(2009-1-1, 2009-1-6)
bookinNotAvailable(2009-1-2, 2009-1-3)
Rollback()
Rollback()
Aborted()
Aborted()
Subscribe(2009-1-2, 2009-1-3)
From now on the BookingRequestor will wait for new dates's availability
bookingCancellation(2009-1-2, 2009-1-10)
cancellationConfirm()
notifyDateAvailability(2009-1-2, 2009-1-3)
createTransaction()
bookingRequest(2009-1-1, 2009-1-6)
bookingRequest AND paymentRequest()
bookin and payment confirm()
Commit()
Commit()
Committed()
Committed()

Simplified overview of the BR Agent in Jason 1/2

```
+!start_booking :
    <- !setupTools;
    !book_hotel.

+!setupTools : resources(Wallet)
    <-
    //Use the RequestMediator to create a new WS-AT and add the relates info to the Wallet
    ...
    cartago.use(Wallet, addInfo(WS-AT-INFO));
    /* Set up the required artifacts (only if not already created)
    for the communication with the designed WSs */
    !buildWSInterface(Name, WsdlURI, Op, Port, WSInt);
    //Add the WS-AT-INFO to the WSInterface configuration
    cartago.use(WSInt, configure(WS-AT-INFO)).
    ...

+!book_hotel : date(Date) & resources(ProxyHotel)
    <-
    //Hotel booking request
    !createBookingMessage(hotel, Date, MsgBookHotel);
    cartago.doRequestResponse(ProxyHotel, bookingOperation(MsgBookHotel), ResponseHotel);

     //Hotel's response inspection for checking the booking availability
    !inspect_hotel_response(ResponseHotel, Resp);

     //Continue the holiday booking operations
    !book_accessories(Resp).
```

Simplified overview of the BR Agent in Jason 2/2

```
//Plan that can be used if is possible book the hotel for the specified dates
+!book_accessories("dates_available") : date(Date) & resources(ProxyTransport, ProxyPayment)
          & hprice(HotelPrice) & tPrice(TransportPrice)
    <- //Transport booking and payment
    !createBookingMessage(transport, Date, MsgTransport);
    cartago.doRequestResponse(ProxyTransport, bookingOperation(MsgTransport), ResponseTransport);
    !createPayMessage("BankAccountID", (HotelPrice+TransportPrice), MsgPay);
    cartago.doRequestResponse(ProxyPayment, payOperation(MsgPay), ResponsePayment);
    !inspect_accessories_responses(ResponseTransport, ResponsePayment, Result);
     //Booking finalization
    !finalize(Result).

//Plan that can be used if is NOT possible book the hotel for the specified dates
+!book_accessories("dates_not_available"):wallet_entry(WS-AT-INFO) & resources(ProxyHotel) & dates(FullDates)
    <- //Transaction rollback and subscription to the interested dates
    !rollbackTransaction(WS-AT-INFO);
    !createSubscribeMessage(FullDates, MsgSubscription);
    cartago.use(ProxyHotel, subscribeOperation(MsgSubscription)).

//Booking and payment successfully done, transaction commit
+!finalize(operations_succeeded) : wallet_entry(WS-AT-INFO)
    <- commitTransaction(WS-AT-INFO).

//Booking and payment NOT successfully done, transaction rollback
+!finalize(operations_not_succeeded) : wallet_entry(WS-AT-INFO)
    <- rollbackTransaction(WS-AT-INFO).

//Notification of new booking availability, the booking operation can be retried
+dateNotMoreFull(FullDates) [source(ProxyHotel)] : resources(ProxyHotel)
    <- !start_booking;
```

Final Remarks

- Service modeling in SOA is still open issue
- Agent and MAS as a suitable modeling for complex systems in general, including aspects neglected by OO and component based systems
- A&Aand CArtAgO-WSenable design and development of true agent based SOA
- System design is not more constrained by low level mechanism but
- focused on important aspects as goal orientedness, fault tolerance, parallel execution, distributed resources, situatedness, etc.

References

CArtAgO and CArtAgO-WS are open-source software:

- Project HomePage:
  - `http://cartago.sourceforge.net`
  - `http://cartagows.sourceforge.net`
- SVN: `https://cartagows.svn.sourceforge.net/svnroot/cartagows`

Jason BDI Agent Platform:

- Project Homepage: `http://jason.sourceforge.net/`

Agents & Artifacts Theory, Design and Practice

- Google Group:
  `http://groups.google.com.au/group/agents-and-artifacts`

Bibliography I

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004).
Web services architecture - w3c working group note 11 february 2004.
http://www.w3.org/TR/ws-arch/.

Bordini, R., Hübner, J. F., and Wooldridge, M. (2007).
Programming Multi-Agent Systems in AgentSpeak Using Jason.
John Wiley & Sons, Ltd.

Bozzo, L., Mascardi, V., Ancona, D., and Busetta, P. (2005).
COOWS: Adaptive BDI agents meet service-oriented computing (extended version).
In *European Workshop on Multi-Agent Systems (EUMAS 2005)*.

Bibliography II

Burmeister, B., Arnold, M., Copaciu, F., and Rimassa, G. (2008).
BDI-agents for agile goal-oriented business processes.
In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Industry and Application Track*.

Cardoso, J. and Sheth, A. (2003).
Semantic e-workflow composition.
J. Intell. Inf. Syst., 21(3):191–225.

Casella, G. and Mascardi, V. (2006).
Intelligent agents that reason about web services: a logic programming approach.
In Polleres, A., Decker, S., Gupta, G., and de Bruijn, J., editors, *Proceedings of the ICLP'06 Workshop Workshop on Applications of Logic Programming n the Semantic Web and Semantic Web Services, ALPSWS2006*, pages 55–70.

Bibliography III

Dickinson, I. and Wooldridge, M. (2005).
Agents are not (just) web services : considering bdi agents and web services.
In *SOCABE 2005*.

Georgeff, M. (2006).
Service Orchestration: The Next Big Thing.
DM Review.

Greenwood, D. and Calisti, M. (2004).
Engineering web service-agent integration.
In *In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1918—1925, The Hague, Netherlands. IEEE Computer Society.

Bibliography IV

Greenwood, D., Lyell, M., Mallya, A., and Suguri, H. (2007).
The IEEE FIPA approach to integrating software agents and web services.
In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7, New York, NY, USA. ACM.

Hunhs, M. N. (2006).
A research agenda for agent-based Service-Oriented Architectures.
In Klusch, M., Rovatsos, M., and Payne, T., editors, *CIA 2006*, volume 4149 of *LNA*, pages 8–22. Springer-Verlag Berlin Heidelberg.

IBM (1995).
Intelligent agent strategy white paper.
Technical report, IBM.

Bibliography V

Lécué, F., Delteil, A., and Léger, A. (2008).
Towards the composition of stateful and independent semantic web services.
In *SAC*, pages 2279–2285.

N. Huhns, M., Singh, M. P., and Burstein, M. e. a. (2005).
Research directions for service-oriented multiagent systems.
IEEE Internet Computing, 9(6):69–70.

Nguyen, X. T. and Kowalczyk, R. (2007).
WS2JADE: Integrating web service with jade agents.
In *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4507 of *LNCS*, pages 147–159. Springer Berlin / Heidelberg.

Bibliography VI

Odell., J. (2002).
Objects and agents compared.
Journal of Object Technologies, 13(1):51–53.

Omicini, A., Ricci, A., and Viroli, M. (2008).
Artifacts in the A&A meta-model for multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 17 (3).

Pistore, M., Marconi, A., Bertoli, P., and Traverso, P. (2005).
Automated composition of web services by planning at the knowledge level.
In *IJCAI 2005*.

Poggi, A., Tomaiuolo, M., and Turci, P. (2007).
An agent-based service oriented architecture.
In *AI*IA Workshop From Object to Agents (WOA-07)*.

Bibliography VII

Ricci, A., Piunti, M., Acay, L. D., Bordini, R., Hübner, J., and Dastani, M. (2008).
Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms.
In *Proc. of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 225–232.

Rimassa, G., Kernland, M. E., and Ghizzioli, R. (2008).
Ls/abpm - an agent-powered suite for goal-oriented autonomic bpm.
In *AAMAS (Demos)*.

Russell, S. J. and Norvig, P. (2002).
Artificial Intelligence: A Modern Approach.
Prentice Hall / Pearson Education International, Englewood Cliffs, NJ, USA, 2nd edition.

Bibliography VIII

Shafiq, A. A., Ahmad, H. F., and Suguri, H. (2005).
AgentWeb Gateway - a middleware for dynamic integration of multi agent system and web services framework.
In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*.

van Riemsdijk, M. B. and Wirsing, M. (2007).
Using goals for flexible service orchestration - a first step.
In *Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE'07)*, volume 4504 of *LNCS*, pages 31–48. Springer-Verlag.

Varga, L. Z. and Hajnal, A. (2003).
Engineering web service invocations from agent systems.
In *In Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems*, pages pages 626–635, Prague, Czech Republic.