A Model-driven, Agent-based Approach for the Integration of Services into a Collaborative Business Process[*]

Ingo Zinnikus, Christian Hahn, and Klaus Fischer
DFKI GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
{ingo.zinnikus, christian.hahn, klaus.fischer}@dfki.de

ABSTRACT

In cross-organisational business interactions, the most desirable solution for integrating different partners would suggest to integrate their processes and data on a rather low level. However, the internal processes and interfaces of the participating partners are often pre-existing and have to be taken as given. Furthermore, in cross-organisational scenarios partners are typically very sensitive about their product data and the algorithms that process it. In many cases, private processes are only partially visible and hidden behind public interface descriptions. This imposes restrictions on the possible solutions for the problems which occur when partner processes are integrated. In this paper, we describe a solution which supports rapid prototyping by combining a model-driven framework for cross-organisational business processes with an agent-based approach for flexible process execution. We show how the model-driven approach can be combined with semantic service discovery for flexible service composition.

Categories and Subject Descriptors

I.2.11 [**Multiagent systems**]; D.2.2 [**Design Tools and Techniques**]; D.2.12 [**Interoperability**]

1. INTRODUCTION

Business process modelling and execution in a collaborative environment requires a set of methodologies and tools which support the transition from an analysis to an execution level and integrate the process with a pre-existing IT infrastructure. In business-driven scenarios, a top-down approach is often applied [15]: a human-comprehensible analysis model which is used for communication among analysts is enriched with process and data details which yields a design model. For generating run-time artifacts, the design model is enriched by technical details leading to a technical model which in turn is transformed into executable code. Due to the rather static nature of many business scenarios, a rather fixed execution platform as e.g. BPEL4WS is used.

[*]The work published in this paper is (partly) funded by the E.C. through the ATHENA IP. It does not represent the view of E.C. or the ATHENA consortium, and authors are solely responsible for the paper's content.

Cite as: A Model-driven, Agent-based Approach for the Integration of Services into a Collaborative Business Process, Ingo Zinnikus, Christian Hahn and Klaus Fischer, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 241-248.

However, even in traditional collaborative settings, points or situations of choice occur, where the specific partner which delivers a specific task, can be selected even at execution time, thus allowing a certain flexibility.

To enable flexible business process execution, the application of agent-based systems has been proposed. Especially in semi-open processes where additional partners are dynamically integrated in the sales process (e.g. non-OEM manufacturers) the system needs to become robust against temporary unavailable partners.

The European project ATHENA (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications) provides a comprehensive set of methodologies and tools to address interoperability problems of enterprise applications in order to realize seamless business interaction across organizational boundaries.

We follow the approach outlined in [12] where business processes are modelled on different abstraction levels and transformed down from a business layer to a technical and an execution level. An agent-based approach is applied for modelling and transforming the technical layer to the execution layer. In [12], for the technical level a platform-independent metamodel for service-oriented architectures (PIM4SOA, an ATHENA result [2]) was used. We replace the PIM4SOA metamodel with a platform-independent metamodel for agents (PIM4Agents) which is more expressive and constitutes a generalization of the PIM4SOA.

In cross-organisational scenarios, the internal processes and interfaces of the participating partners are often pre-existing and have to be taken as given. Furthermore, partners are typically very sensitive about their product data and the algorithms that process it. In many cases, private processes are only partially visible and hidden behind public interface descriptions [23]. This imposes restrictions on the possible solutions for the problems which occur when partner processes are integrated. Since existing services have to be integrated, the agents are situated in a service-oriented and specifically, a Web service environment. Two tasks which involve interoperability problems have to be tackled:

- integrate services which are "fixed" i.e. known in advance when the process is specified.
- discovering/provisioning services at design time which are additionally required or could be beneficial for improving the overall result or reducing costs.

A model-driven approach is used for the integration of existing services and dynamic service selection and composition (if feasible) for enhancing flexibility. For discovery,

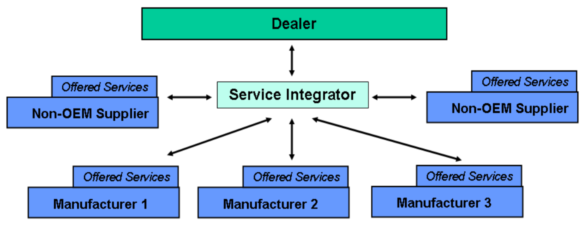


Figure 1: Overview over the architecture of the solution.

we support three of the most common Semantic Web Service Description Frameworks, namely, OWL-S [17], WSMO [5] and SAWSDL [8], which has just reached the status of a proposed recommendation within W3C.

The paper is organized as follows. In Section 2 we will sketch the business case of our pilot application which we use as motivational background, containing static interactions with dynamic features. Section 3 is devoted to our technical approach. Here, we present the approach developed in ATHENA and used within our pilot for the integration of cross-organizational processes. We show how the information contained in the PIM model can be used to discover required services using semantic web technologies in Section 4. We evaluate the approach by discussing advantages of agent-based SOAs in Section 5, related work in Section 6 and conclude the paper by taking a look at the lessons learned in Section 7.

2. SCENARIO

In 2002, due to new laws in EU legislation, the market of car distribution changed fundamentally. Instead of being limited to selling only one brand, vending vehicles of different brands under one roof was facilitated. Dealers now can reach a broader audience and improve their business relations for more competitiveness. As a consequence, many so-called *multi-brand dealers* have appeared.

Today, multi-brand dealers are confronted with a huge set of problems. Rather than having to use the IT system of one specific car manufacturer, multi-brand dealers are now faced with a number of different IT systems from their different manufacturers. One specific problem is the integration of configuration, customization and ordering functionality for a variety of brands into the IT landscape of a multi-brand dealer.

We describe an integrated scenario, where multi-brand dealers use services provided by the different car and non-OEM manufactures and plug them into an integrated dealer system.

The desired to-be-scenario with its general architecture is depicted in Figure 1. The systems of the different car and non-OEM manufacturers are integrated via an integrator component. This integrator enables the dealer to access the software of the manufacturers in a uniform manner. The paper will focus on the manufacturer integration (Section 3) and present the model-driven, agent-based integration approach for cross-organizational processes modeling.

The dealer (software) interacts with the manufacturer systems by sending product information, configuration and ordering requests. The service integrator acts as an intermediary and adapts the dealer requests to the service interfaces provided by the manufacturers. The interaction protocol consists of either one-shot synchronous message exchanges or contract-net style interactions where a supplier is chosen based on non-functional criterias, thus introducing flexibility into an otherwise rather static scenario. The car manufacturers are preassigned, whereas the different part suppliers and non-OEM manufactures can eventually be selected at run-time. However, since car parts must adhere to norms, the list of potentially useful suppliers is based on a partner list defined and contracted at design time.

For the service integrator, the process is modelled on a PIM level and generated platform-specific models (PSM) are executed as software agents on Jack [1], an agent platform based on the BDI-agent theory (*belief-desire-intention*, [21]). In the following, we will describe this approach in detail.

3. MEDIATING SERVICES FOR CROSS-ORGANISATIONAL BUSINESS PROCESSES

The scenario involves a complex interaction between the partners. The design of such a scenario implies a number of problems which have to be solved:

- changing the protocol and integration of a new partner should be possible in a rapid manner (scalability)
- the execution of the message exchange should be flexible, i.e. in case a partner is unavailable or busy, the protocol should nevertheless proceed
- the different partners (may) expect different atomic protocol steps (service granularity)
- the partners expect and provide different data structures

These are typical interoperability problems occuring in cross-organisational scenarios which in our case have to be tackled with solutions for agents and SOAs. A core idea in the ATHENA project was to bring together different approaches and to combine them into a new framework: a modelling approach for designing collaborative processes, a model-driven development framework for SOAs and an agent-based approach for flexible execution.

Hence, the first problem is solved by applying a model-driven approach: the protocol is specified on a platform-independent level so that a change in the protocol can be made on this level and code generated automatically. Flexibility is achieved by applying a BDI agent-based approach. BDI agents provide flexible behaviour for exception-handling in a natural way (compared to e.g. BPEL4WS where specifying code for faults often leads to complicated, nested code). The problem of different service granularities is envisaged by specifying a collaborative protocol which allows adapting to different service granularities. Finally, the mediation of the data is tackled with transformations which are specified at design-time and executed at run-time by transforming the exchanged messages based on the design-time transformations. Two cases can be distinguished:

(i) integrating consortial partners and their services

(ii) integrating partners external to the collaboration and their services

i) Partners define the shared process together. The common information/data model may also be defined together. Roughly speaking, two alternatives are possible (analogous to the local-as-view vs. global-as-view distinction, cf. [16]): the common data structure is defined independently from the local data model of each partner. Each partner then defines a (local) mapping from the common data model to the local model. The mapping in turn can be executed (at run-time) either by the consumer of the service or the partner service itself. The first solution is the one preferred by Semantic Web service descriptions, e.g. OWL-S where the service provider describes the grounding to e.g. WSDL. The grounding is used by a service consumer who invokes the service. The second solution means that the service consumer always sends the same message (e.g. a SOAP message) to a partner service and does not care about the local data model. This is reasonable if specifying as well as testing the mapping is tedious and the mapping underlies many changes.

In a global-as-view approach, the common data model is defined as view on the local data models of the partners. A disadvantage of this approach is that the integration of a new partner requires changing the common data model.

ii) For integrating external partners service discovery as well as process and data mediation have to be realized. Service discovery can be based on the service requirements which are specified implicitly or explicitly for a service invocation task. Integrating the discovered services requires data transformations which are either provided by the service descriptions or based on a mapping to the common information model.

PIM4Agents: A Platform-Independent Model for Agents

The PIM4Agents [10] is a visual platform-independent model that specifies agent systems in a technology independent manner. It represents an integrated view on agents in which different components can be deployed on different execution platforms. The PIM4Agents metamodel defines modelling concepts that can be used to model six different aspects or views of an agent system that are listed below:

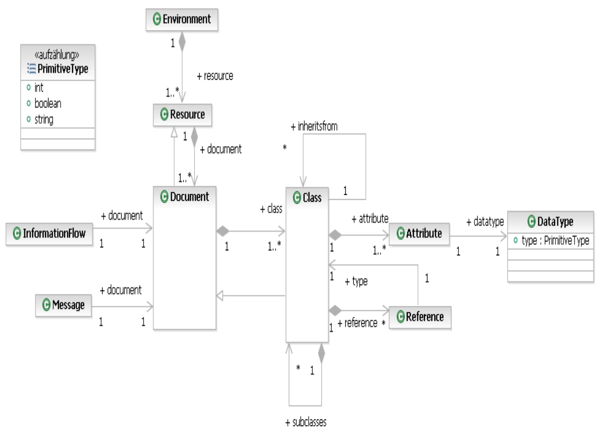


Figure 2: PIM4Agents environment metamodel.

Agent aspect describes single autonomous entities, the capabilities they have to solve tasks and the roles they play within the MAS. The agent aspect is centered on the concept of *Agent*, the autonomous entity capable of acting in the environment. An *Agent* has access to a set of Resources from its surrounding *Environment*. These *Resources* may include information or ontologies the *Agent* has access to. Furthermore, the *Agent* can perform particular *Roles* that define in which specific context the *Agent* is acting and *Behaviours* that define how particular tasks are achieved. Furthermore, the *Agent* may have certain *Capabilities* that represent the set of *Behaviours* the *Agent* can possess. For the purpose of interaction, an *Agent* could be member in an *Organization* that represents the social structure *Agents* can take part in.

Organization aspect describes how autonomous entities cooperate within the MAS and how complex organizational structures can be defined. The *Organization* is a special kind of *Cooperation* that also has the same characteristics of an *Agent*. Therefore, the *Organization* can perform *Roles* and have *Capabilities* which can be performed by its members, be it agents or suborganizations. The multiple inheritance of the *Organization*, from the *Agent* and the *Cooperation*, also allows it to have its own internal *Protocol* that specifies (i) how the *Organization* communicates with other *Agents* (either atomic *Agents* or complex *Organizations*) and (ii) how organizational members are coordinated.

Role aspect covers feasible specializations and how they could be related to each other. Informally, a *Role* is an abstraction of the social behaviour of the *Agent* in a given social context, usually a *Cooperation* or *Organization*. The *Role* specifies the responsibilities of the *Agent* in that social context. It refers to (i) a set of *Capabilities* that defines the set of *Behaviours* it can possess and (ii) a set of *Resources* an *Agent* has access to.

Interaction aspect describes how the interaction between autonomous entities or organizations takes place. Each interaction specification includes the *Actors* involved and in which order *Messages* are exchanged between these *Actors* in a protocol-like manner. Furthermore, a *Protocol* refers (i) to a set of *TimeOuts* that define the time constraints for sending and receiving *Messages*, (ii) to a set of *MessageScopes* that defines the *Messages* and their order how these arrive, and (iii) to a set of *MessageFlows* that specify how the exchange of *Messages* is proceed.

Behavioural aspect describes how *Plans* are composed by complex control structures and simple atomic tasks like sending a message and how information flows between those constructs. A *Plan* specifies the agents' internal processes. It represents a super class connecting the agent aspect with the behavioural aspect. Informally, a *Plan* refers to a set of *Flows*, be them *ControlFlows* or *InformationFlows*, that are contained in the plan description. These different specializations of *Flow* link *Activities* to each other, either defining the control flow or information flow.

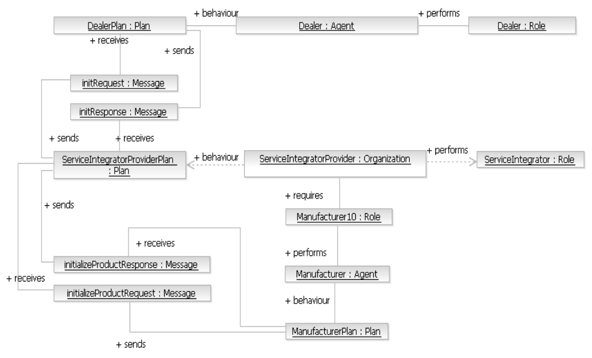


Figure 3: PIM4Agents Model for Pilot (part).

Environment aspect (depicted in Figure 2) contains any kind of *Resource* that is dynamically created, shared, or used by the *Agents* or *Organizations*, respectively.

Via model-to-model transformations, PIM4Agents models can be transformed into underlying platform-specific models such as Jack or JADE[1].

The business protocol between dealer (dealer software), integrator and manufacturers is specified as PIM4Agents model (see Figure 3 for the general collaboration between roles). The interaction between the roles is defined by interaction protocols (see Figure 4 and Figure 5). For the interaction between service integrator and manufacturers, a message flow *MF* which represents the action of sending a message to a collaboration partner is assigned to an interaction role (e.g. *ServiceIntegrator*). Specified messages are sent in parallel to the partners (e.g. *Manufacturer1* and *Manufacturer2* resp.) which reply by sending an answer message.

The interaction protocol in Figure 5 represents a contract-net style conversation between service integrator and part provider(s). Note that a role can be instantiated by several agents so that messages are actually send to all partners which are instances of the role *PartServiceProvider*.

In order to execute collaborative processes specified on the PIM level, the first step consists of transforming PIM4Agents models to agent models that can be directly executed by specific agent execution platforms. In our case, the Jack Intelligent agent framework is used for the execution of BDI-style agents. The constructs of the PIM4Agents metamodel are mapped to BDI-agents represented by the Jack metamodel (JackMM). For detailed information on JackMM we refer to [11].

The partner models are transformed to a Jack agent model with the model-to-model transformation developed in the ATHENA project. The following sketch outlines the metamodel mappings (see Figure 6).

An *Organization* (i.e. ServiceIntegratorProvider in Figure 3) is assigned to a *Team* (which is an extension of an *Agent*). The name of the *Organization* coincides with the name of the *Team*, its roles are the roles the *Team* performs. Furthermore, the *Team* makes use of the *Roles* required by the *Organization* (i.e. Dealer and Manufacturer), in which it participates. For each of these required *Roles*, we additionally introduce an atomic *Team*. The *Plan* of the *Organization* is mapped to the *TeamPlan* of the non-atomic *Team*. This *TeamPlan* defines how a combined service is orchestrated by making use of the services the atomic *Teams* (i.e. ManufacturerTeam in Figure 7) provide. Finally, *Messages* that are sent and received within *Plans* are mapped to *Events* that are handled and sent in JackMM.

[1]for detailed information see http://jade.tilab.com/

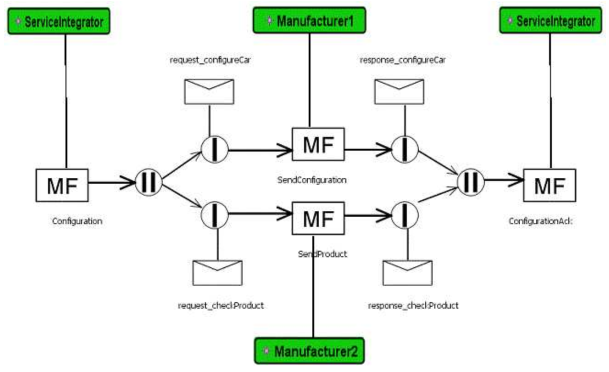


Figure 4: PIM4Agents Interaction Model (Service Integrator - Manufacturer part).

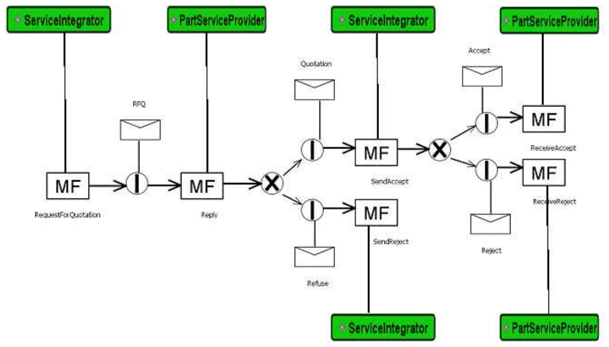


Figure 5: PIM4Agents Interaction Model (Service Integrator - Parts Provider).

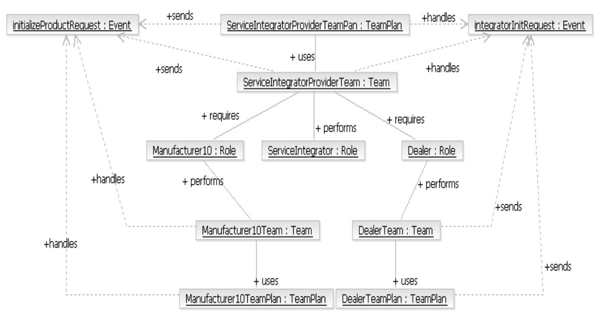

initializeProductRequest : Event
+sends
ServiceIntegratorProviderTeamPan : TeamPlan
+handles
integratorInitRequest : Event
+ uses
+sends
+handles
ServiceIntegratorProviderTeam : Team
+ requires
+ performs
+ requires
Manufacturer10 : Role
ServiceIntegrator : Role
Dealer : Role
+handles
+ performs
+ performs
+sends
Manufacturer10Team : Team
DealerTeam : Team
+handles
+sends
+ uses
+ uses
Manufacturer10TeamPlan : TeamPlan
DealerTeamPlan : TeamPlan

Figure 7: Jack Model generated from PIM4Agents (part).

In this service-oriented setting, the partners provide and exhibit services. Partner (manufacturer etc.) services are described as WSDL interfaces. The WSDL files are used to generate integration stubs for the integrator. We use a model-driven approach for mapping WSDL concepts to

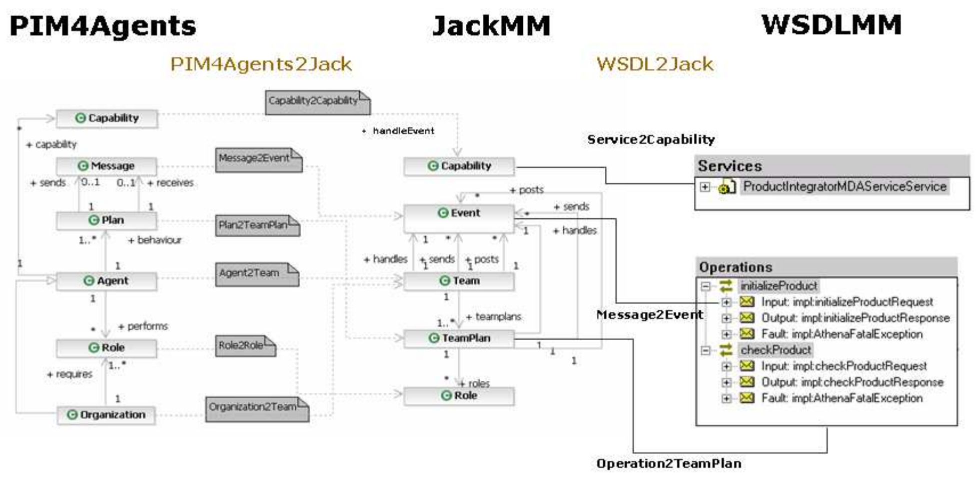


Figure 6: PIM4Agents and WSDLMM to JackMM transformation.

agent concepts, thereby integrating agents into an SOA and supporting rapid prototyping.

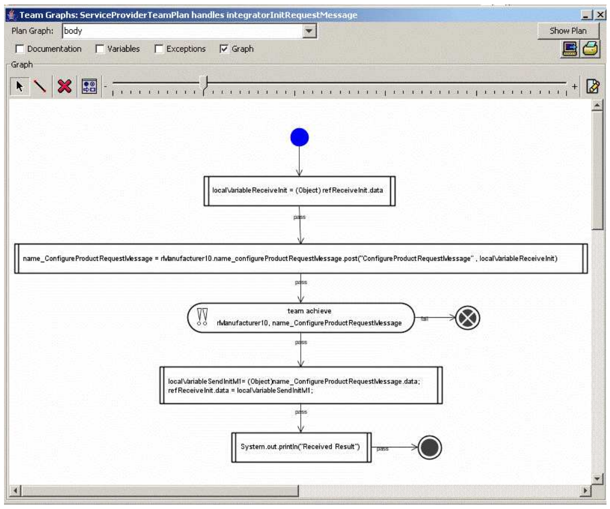
The process integrator and the manufacturers are modelled as Web services. Their interface is described by WSDL descriptions publishing the platform as Web service. In the pilot, only the process integrator is executed by Jack agents which are wrapped by a Web service, whereas the manufacturers and other partner services are pure Web services. For integrating Web services into the Jack agent platform, we map a service as described by a WSDL file to the agent concept *Capability* which can be conceived of as a module. A capability provides access to the Web services via automatically generated stubs. A capability comprises of plans for invoking the operations as declared in the WSDL (it encapsulates and corresponds to commands such as invoke and reply in BPEL4WS).

Applying these transformations, a PIM4Agents model can be automatically transformed into a PSM model, e.g. the Jack model illustrated in Figure 7. The PSM model is then serialized into Jack artifacts, e.g. *Teams*, *Events* (Messages), *Team Plans* (see Figure 8) and *Beliefs* which can be further modified if necessary.

It should be stressed that these model transformations and the respective code generation can be done automatically if (i) the PIM4Agents model is defined properly and (ii) the WSDL descriptions are available. The only interventions necessary for a system designer are the insertion of the proper XSLT transformations and the assignment of the capabilities to the agents/teams responsible for a specific Web service invocation.

4. DERIVING SERVICE REQUIREMENTS FROM THE PIM MODEL AND LOCATING EXTERNAL SERVICES

In order to discover and select additional services, we can use the information contained in the PIM4Agents model, especially those of the process and environment aspect. The environment aspect contains the data structure which is


Team Graphs: ServiceProviderTeamPlan handles integratorInitRequestMessage
Plan Graph:
body
Show Plan
Documentation
Variables
Exceptions
Graph
pass
fail

Plan Graph: body

[ ] Documentation [ ] Variables [ ] Exceptions [✓ Graph]

```
localVariableReceiveInit = (Object) refReceiveInit.data
```

```
name_ConfigureProductRequestMessage = rManufacturer10.name_configureProductRequestMessage.post("ConfigureProductRequestMessage" , localVariableReceiveInit)
```

```
team achieve
rManufacturer10, name_ConfigureProductRequestMessage
```

```
localVariableSendInitM1= (Object)name_ConfigureProductRequestMessage.data;
refReceiveInit.data = localVariableSendInitM1;
```

```
System.out.println("Received Result")
```

Figure 8: Generated Jack Team Plan for ServiceIntegrator.

commonly agreed upon among the collaborating partners and amounts to or (ideally) conforms to a shared ontology. It can furthermore be used to specify search requests for potentially usable services.

In our automotive scenario, there are a number of standards which can form the basis for the environment model (e.g. STAR standard for automotive retail industry[2]). The concepts of the standard and their relation to each other are either integrated into the common data model or used as annotation of the data model. If the local data model of the partners differs from the common model, the local partner is responsible for defining a mapping from the common model to the local model. If we assume that other external services use the same vocabulary for their service description (or their annotation), the concepts can be used to formulate

[2]www.starstandard.org/

service requests and search for relevant services which offer the required functionality. This assumption of a shared vocabulary among actors is reasonable, since in our scenario product data underlies strong standardization pressure.

In order to do that, we have to collect the description about a required service from the model, i.e. service operation, input and output of the service as well as pre- and postconditions of the tasks. In fact, this information about the service requirements is sufficient to formulate a query (at design time) to a semantic matchmaker, e.g. for an OWL-S, a WSMO, or a SAWSDL matchmaker.

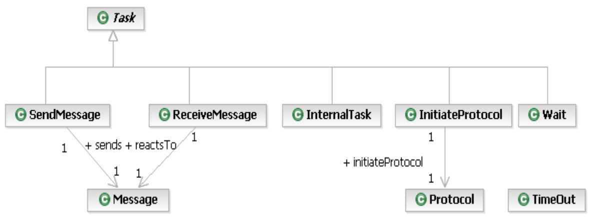
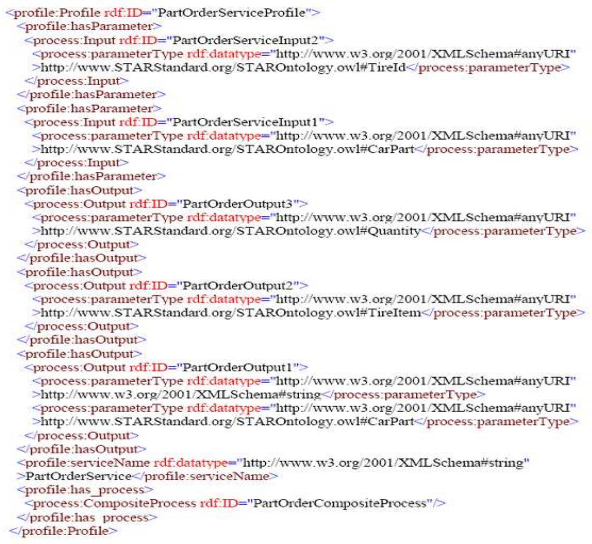


Figure 9: PIM4Agents Process Metamodel (part).

If a plan step, e.g. a Task (see Figure 9) involves a message exchange with an external partner, the requirements which the external partner must fulfil can be derived from the PIM model. The pre- and postconditions of a task and the message refer to the input and output parameters which the external service must provide. Since pre- and postconditions as well as input and output are the main features a matchmaker uses for checking service compliance, the service requirement contained in the PIM model can be transformed into service requests by using the pre- and postconditions of a task (for WSMO) or input and output concepts (for OWL-S or SAWSDL) of a message sent to a specific service. For matchmaking, we use the OWL-S [14] and WSMO matchmaker [13].

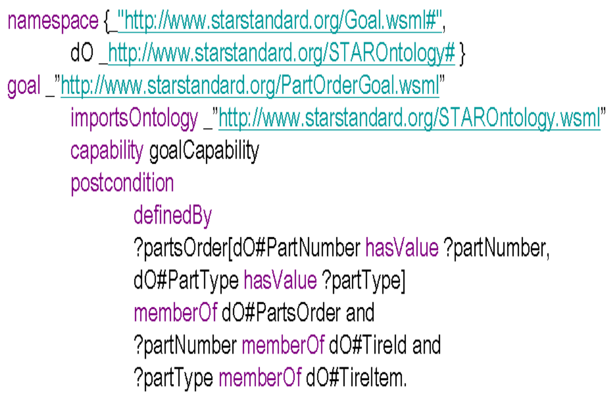


```
<profile:Profile rdf:ID="PartOrderServiceProfile">
  <profile:hasParameter>
   <process:Input rdf:ID="PartOrderServiceInput2">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.STARStandard.org/STAROntology.owl#TireId</process:parameterType>
   </process:Input>
  </profile:hasParameter>
  <profile:hasParameter>
   <process:Input rdf:ID="PartOrderServiceInput1">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.STARStandard.org/STAROntology.owl#CarPart</process:parameterType>
   </process:Input>
  </profile:hasParameter>
  <profile:hasOutput>
   <process:Output rdf:ID="PartOrderOutput3">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.STARStandard.org/STAROntology.owl#Quantity</process:parameterType>
   </process:Output>
  </profile:hasOutput>
  <profile:hasOutput>
   <process:Output rdf:ID="PartOrderOutput2">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.STARStandard.org/STAROntology.owl#TireItem</process:parameterType>
   </process:Output>
  </profile:hasOutput>
  <profile:hasOutput>
   <process:Output rdf:ID="PartOrderOutput1">
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
    <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"
    >http://www.STARStandard.org/STAROntology.owl#CarPart</process:parameterType>
   </process:Output>
  </profile:hasOutput>
  <profile:serviceName rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >PartOrderService</profile:serviceName>
  <profile:has_process>
   <process:CompositeProcess rdf:ID="PartOrderCompositeProcess"/>
  </profile:has_process>
</profile:Profile>
```

Figure 10: Generated OWL-S Service Profile for Parts Order Service.

Since the OWL-S matchmaker [14] essentially takes the service profile as requirement, we generate an OWL-S service description with a profile (see Figure 10 for a requirement definition for a part order service). An OWL-S matchmaker request consists of an OWL-S service description, including a service profile with input and output concepts referring to a shared ontology.

In WSMO, a Goal describes the requested functionality of a required service. Figure 11 shows an example of a goal which specifies that a parts order service must be able to provide parts of a specific type.



```
namespace {_"http://www.starstandard.org/Goal.wsml#",
     dO _http://www.starstandard.org/STAROntology# }
goal _"http://www.starstandard.org/PartOrderGoal.wsml"
     importsOntology _"http://www.starstandard.org/STAROntology.wsml"
     capability goalCapability
     postcondition
          definedBy
          ?partsOrder[dO#PartNumber hasValue ?partNumber,
          dO#PartType hasValue ?partType]
          memberOf dO#PartsOrder and
          ?partNumber memberOf dO#TireId and
          ?partType memberOf dO#TireItem.
```

Figure 11: Generated WSMO Goal for Parts Order Service.

Integrating discovered services at design time can be done either by using the grounding contained in the service descriptions or by directly integrating the service using its WSDL description as described in Section 3.

5. ADVANTAGES OF AGENT-BASED SOAS

The similarities between agent architectures and SOAs have already been recognized (e.g. [24]). In the following we will briefly discuss advantages of applying BDI-agents in a service-oriented environment.

In order to compare an agent-based approach with other standards for Web service composition, the distinction introduced in [27] between *fixed*, *semi-fixed*, and *explorative composition* is useful. Fixed composition can be done with e.g. BPEL4WS, but also by applying BDI agents. Semi-fixed composition might also be specified with BPEL4WS: partner links are defined at design-time, but the actual service endpoint for a partner might be fixed at run-time, as long as the service complies with the structure defined at design-time. Late binding can also be done with the Jack framework. The service endpoint needs to be set (at the latest) when the actual call to the service is done. Explorative composition is beyond of what BPEL4WS and a BDI-agent approach offer (at least if they are used in a 'normal' way). To enable explorative composition, a general purpose planner might be applied which dynamically generates, based on the service descriptions stored in a registry, a plan which tries to achieve the objective specified by the consumer [25].

It might seem as if BPEL4WS and BDI-style agents offer the same features. However, there are several advantages of a BDI-style agent approach. An important question is how the availability of a partner service is detected. This might be checked only by actually calling the service. If the service is not available or does not return the expected output, an exception will be raised. BPEL4WS provides a fault handler which allows specifying what to do in case of an exception. Similarly, an agent plan will fail if a Web service call raises an exception, and execute some activities specified for the failure case.

However, the difference is that a plan is executed in a context which specifies conditions for plan instances and also other applicable plans. The context is implicitly given by the beliefs of an agent and can be made explicit. If for a specific goal several options are feasible, an agent chooses one of these options and, in case of a failure, immediately executes the next feasible option to achieve the desired goal. This means that in a given context, several plan instances might be executed, e.g. for all known services of a specific type, the services are called (one after another), until one of the services provides the desired result. An exception in one plan instance then leads to the execution of another plan instance for the next known service. Additionally, BDI-style agents permit 'meta-level reasoning' which allows choosing the most feasible plan according to specified criteria.

In our car configuration scenario, agents have to react to service unavailability and the protocols for e.g. selecting a non-OEM supplier involve auctions or *first come - first served* mechanisms which can be modelled in an very elegant manner with a BDI-agent approach. The BDI-agent approach supports this adaptive behaviour in a natural way, whereas a BPEL4WS process specification which attempts to provide the same behaviour would require awkward coding such as nested fault handlers etc.

Furthermore, since it is in many cases not possible to fully specify all necessary details on the PIM level, a system engineer must add these details on the PSM level. Hence, *customizing* the composition is facilitated since the different plans clearly structure the alternatives of possible actions. Since the control structure is implicit, changes in a plan do not have impact on the control structure, reducing the danger of errors in the code. Another advantage is that *extending* the behaviour by adding a new, alternative plan for a specific task is straightforward. The new plan is simply added to the plan library and will be executed at the next opportunity.

Finally, business process notations allow specifying unstructured processes. To execute these processes with BPEL, unstructured business process descriptions normally are transformed to block-structured BPEL processes. In doing so, most approaches restrict the expressiveness of processes by only permitting acyclic or already (block-) structured graphs [18]. In the case that any unstructured processes shall be executed, an approach like described in [19] has to be followed. The idea is to translate processes with arbitrary topologies to BPEL by making solely use of its Event Handler concept. The result is again cumbersome BPEL code, whereas the Jack agent platform naturally supports event-based behaviour.

6. RELATED WORK

Apart from the wealth of literature about business process modelling, enterprise application integration and SOAs, the relation between agents and SOAs has already been investigated. [24] cover several important aspects, [26] propose the application of agents for workflows in general.

[4] provide an overview of agent-based modelling approaches for enterprises. [20] describe the TROPOS methodology for a model-driven design of agent-based software systems. However, the problems related to integration of agent platforms and service-oriented architectures are beyond their focus. [7] map BPMN models to BDI agents but do not consider an integration of agents and Web services.

[9] and [6] present a technical and conceptual integration of an agent platform and Web services. [22] integrate Web services into agent-based workflows, [3] integrate BDI agents and Web services. However, the model-driven approach and the strong consideration of problems related to cross-organisational settings have not been investigated in this context. Furthermore, our focus on a tight and lightweight integration of BDI-style agents fits much better to a model-driven, process-centric setting than the Web service gateway to a JADE agent platform considered by e.g. [9]. The single agents interact directly with a service, without an intermediate gateway converting messages into Web service calls (and vice versa).

7. CONCLUSIONS AND SUMMARY

From a research transfer point of view, the following lessons could be learned:

- Evidently, a model based approach is a step in the right direction as design-time tasks are separated from run-time tasks which allows performing them graphically. Moreover, it is easier to react to changes of the different interacting partners as only the models have to be adapted but not the run-time environment.

- A model-driven, agent-based approach offers additional flexibility and advantages (in general and in the scenario discussed) when agents are tightly integrated into a service-oriented framework.

- The PIM4Agents metamodel is expressive enough for modelling service requirements which can be used for semantic service discovery.

In this paper, we presented a pilot developed within the EU project ATHENA in the area of multi-brand automotive dealers. For its realization, several integration problems on different levels had to be solved. We described a solution which supports rapid prototyping by combining a model-driven framework for cross-organisational service-oriented architectures with an agent-based approach for flexible process execution. The model-driven approach can be combined with semantic service discovery for flexible service composition. We finally argued that agent-based SOAs provide additional advantages over standard process execution environments.

8. REFERENCES

[1] JACK Intelligent Agents, The Agent Oriented Software Group (AOS). http://www.agent-software.com/shared/home/, 2006.

[2] G. Benguria, X. Larrucea, B. Elvesæ ter, T. Neple, A. Beardsmore, and M. Friess. A platform independent model for service oriented architectures. In *2nd International Conference on Interoperability of Enterprise Software and Applications (I-ESA 2006)*, 2006.

[3] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. Coows: Adaptive bdi agents meet service-oriented

computing – extended abstract. In *Proceedings of the Third European Workshop on Multi-Agent Systems (EUMAS'05)*, 2005.

[4] G. Cabri, L. Leonardi, and M. Puviani. Service-Oriented Agent Methodologies. In *5th IEEE International Workshop on Agent-Based Computing for Enterprise Collaboration (ACEC-07)*, 2007.

[5] J. de Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, U. Keller, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, H. Lausen, E. Oren, A. Polleres, D. Roman, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). W3C Member Submission, 3 June 2005. Available from http://www.w3.org/Submission/WSMO/.

[6] I. Dickinson and M. Wooldridge. Agents are not (just) web services: Considering BDI agents and web services. In *AAMAS 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE)*, 2005.

[7] H. Endert, T. Küster, B. Hirsch, and S. Albayrak. Mapping BPMN to Agents: An Analysis. In *First international Workshop on Agents, Web-Services, and Ontologies Integrated Methodologies (AWESOME'07)*, page 164, 2007.

[8] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. W3C Proposed Recommendation, 05 July 2007. Available from http://www.w3.org/TR/sawsdl/.

[9] D. Greenwood and M. Calisti. Engineering web service - agent integration. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1918–1925, 2004.

[10] C. Hahn, C. Madrigal-Mora, and K. Fischer. Interoperability through a platform-independent model for agents. In *3rd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2007)*, 2007.

[11] C. Hahn, C. Madrigal-Mora, K. Fischer, B. Elvesæter, A.-J. Berre, and I. Zinnikus. Meta-models, models, and model transformations: Towards interoperable agents. In *MATES*, pages 123–134, 2006.

[12] T. Kahl, I. Zinnikus, S. Roser, C. Hahn, J. Ziemann, J. Müller, and K. Fischer. Architecture for the design and agent-based implementation of cross-organizational business processes. In *3rd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2007)*, 2007.

[13] F. Kaufer and M. Klusch. WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In *4th European Conference on Web Services (ECOWS '06)*, pages 161–170, 2006.

[14] M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with OWLS-MX. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

[15] J. Koehler, R. Hauser, J. Küster, K. Ryndina, J. Vanhatalo, and M. Wahler. The role of visual modeling and model transformations in business-driven development. In *Fifth International Workshop on Graph Transformation and Visual Modeling Techniques*, April 2006.

[16] M. Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[17] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. W3C Member Submission. Available from http://www.w3.org/Submission/OWL-S/, 22 November 2004.

[18] J. Mendling, K. Lassen, and U. Zdun. Transformation strategies between blockoriented and graph-oriented process modelling languages. Technical Report JM-200510 -10, TU Vienna, 2005.

[19] C. Ouyang, M. Dumas, S. Breutel, and A. H. M. ter Hofstede. Translating Standard Process Models to BPEL. In *CAiSE*, pages 417–432, 2006.

[20] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. From stakeholder intentions to software agent implementations. In *CAiSE*, pages 465–479, 2006.

[21] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[22] B. T. R. Savarimuthu, M. Purvis, M. Purvis, and S. Cranefield. Agent-based integration of web services with workflow management systems. In *Fourth international joint conference on Autonomous agents and multiagent systems (AAMAS 05)*, pages 1345–1346, 2005.

[23] K. Schulz and M. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data & Knowledge Engineering*, 51(1):109–147, 2004.

[24] M. P. Singh and M. N. Huhns. *Service-oriented Computing - Semantic, Processes, Agents*. John Wiley & Sons, Ltd, 2005.

[25] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for Web Service composition using SHOP2. In *International Semantic Web Conference 2003*, volume 1, pages 377–396, October 2004.

[26] J. Vidal, P. Buhler, and C. Stahl. Multiagent systems with workflows, 2004.

[27] J. Yang, W. Heuvel, and M. Papazoglou. Tackling the Challenges of Service Composition in e-Marketplaces. In *Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE-2EC 2002), San Jose, CA, USA*, 2002.