An Agent-Based, Model-Driven Approach for Enabling Interoperability in the Area of Multi-brand Vehicle Configuration*

Ingo Zinnikus[2], Christian Hahn[2], Michael Klein[1], and Klaus Fischer[2]

[1] CAS Software AG, Karlsruhe (Germany)
`michael.klein@cas.de`
[2] DFKI GmbH, Saarbrücken (Germany)
{`ingo.zinnikus, christian.hahn, klaus.fischer`}`@dfki.de`

Abstract. With the change of EU regulations in the automotive market in 2002, multi-brand car dealers became possible. Despite the high economical expectations connected with them, the existing IT infrastructure does not provide satisfying support for these changes as it had been developed independently by each brand for many years. In this paper, we describe a solution which supports rapid prototyping by combining a model-driven framework for cross-organisational service-oriented architectures (SOA) with an agent-based approach for flexible process execution. We discuss advantages of agent-based SOAs and summarize the lessons learned.

1 Introduction

In cross-organisational business interactions such as multi-brand vehicle configuration, 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 [1]. This imposes restrictions on the possible solutions for the problems which occur when partner processes are integrated.

Thus, a service-oriented architecture (SOA) is the most appropriate approach. It enables partners to offer the functionality of their systems via a public service interface (WSDL) and hide the sensitive parts behind it. As usual in a SOA, the communication is performed by the exchange of messages between the partners.

A very important second advantage of SOA is the possibility of a loose coupling of partners. New partners can enter the system with little effort whereas

* 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.

obsolete partners are able to leave it easily. Especially in the case where additional smaller non-OEM manufacturers providing vehicle parts like radios or tires are integrated in the sales process, the system needs to become robust against temporary unavailable partners.

Despite the advantages of a SOA, several difficulties arise especially in the case where the systems of the partners have evolved independently for several years:

– The *philosophies* of the systems differ, e.g. one partner service uses a strict sequential run through the product space whereas another service allows e.g. randomly browsing through the products and product features.
– The *granularity* of operations of the various partner services differs.
– *Non-functional aspects* such as exception handling, session management, transactional demarcation, which differ from partner to partner, supersede the core functionality of the services.
– *Structural differences* in the payload data of the exchanged messages stemming from data models used by the different partners' sites are present.
– *Semantical misunderstandings* within the exchanged messages may arise due to different tagging of business data, different conventions etc.

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. In this paper, we present the results of a pilot application of the ATHENA approach to interoperability and the supporting technology in a real-world scenario of a multi-brand vehicle dealer.

The paper is organized as follows. In Section 2 we will sketch the business case of our pilot application and discuss the current and the to-be scenario for multi-brand vehicle dealers. Sections 3 and 4 are 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 discuss the advantages of this approach in Section 5 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.

In this paper, the business cases we are looking at are such multi-brand dealers. Multi-branding seamlessly offers products of different brands in one coherent sales process. This establishes a certain level of comparability among products of different brands and provides added value to the customers, thus strengthens the competitiveness of multi-brand dealers. However, multi-branding calls for an increased level of interoperability among the dealer on one side and the different manufacturers on the other side.

Today, however, systems for car configuration and order processing of different car manufacturers are isolated systems and not integrated into the dealer specific IT landscape. Thus, multi-brand dealers are faced with a simple multiplication of IT systems to support their pre-sales, sales and after-sales processes. As a consequence, one of the desired advantages of multi-branding, namely to seamlessly offer cars of different car manufactures and to establish comparability among the different products is seriously put at stake. We rather observe the phenomenon of what we call *early brand selection*, i.e. a customer has to choose his desired brand at the beginning and than go all the way through its brand-specific product configuration and order process. Changing the brand later means starting the process all over from the beginning.

In this paper, we propose 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. In the following section, we will describe our solution in more detail.

3 Our Solution

The desired to-be-scenario with its general architecture is depicted in Figure 1. The solution consists of two parts which are necessary to provide an integrated solution for a multi-brand dealer:

– *An integration of the manufacturers* (lower part of the figure). 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

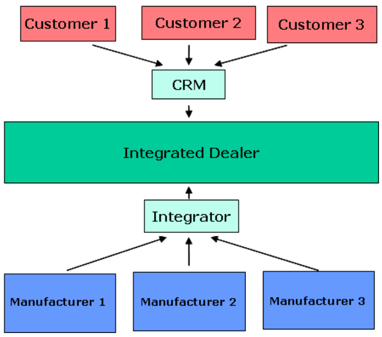

Fig. 1. Overview over the architecture of the solution

of the manufacturers in a uniform manner. For the sake of the pilot application, the car configurator CAS Merlin by CAS Software AG currently used for order processing and sales support applications by a leading German car manufacturer was used.

– *An integration of the customers* (upper part of the figure). The interaction of customers and the dealer is harmonized by integrating their different processes within a CRM system. In the pilot setting, the CRM system CAS genesisWorld was used.

In the following, we give an overview of our approach of the pilot application. However, the paper will focus on the manufacturer integration (see Section 3 and 4) and present the model-driven, agent-based integration approach for cross-organizational processes modeling. The customer integration has already been presented in detail in [2].

Manufacturer Integration

The integrator in the overall architecture in Figure 1 can be seen as a service integrator performing message transformations.

The messages that are exchanged between the dealer and the manufacturers are (conceptually) transformed in three steps. The resulting three layers of the architecture of our service integrator are shown in Figure 2 (left hand side).

In the top most component, a message entering the component from the dealer is analyzed by the CAS Instance Distributor and routed to the set of manufacturers that need to process this request. If the dealer e.g. wants to find a suitable family car for his customer, typically all (or many) of the manufacturers will receive the message. If the dealer however wants to configure and order a car of a certain brand, only this specific system will be addressed. In the inverse direction, i.e. when the results of the different manufacturers reach the component, the CAS Instance Aggregator comes into play: by applying metrics of equality, similarity

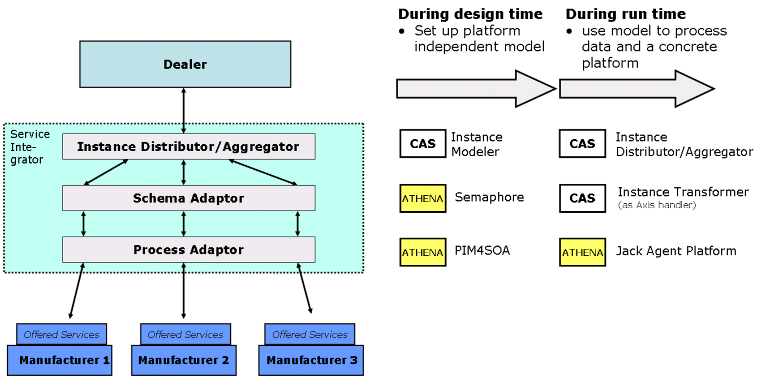

Fig. 2. Integration of car and non-OEM manufacturers

and equivalence, it tries to combine the different partial results to one meaningful integrated result.

The middle layer is responsible for harmonizing the data models of the different manufacturers with the common data model of the dealer. Thus, the business objects extracted from the incoming messages are remodeled and put into the outgoing messages. It is important to mention that for this step in certain cases several messages must be processed together in order to be able to rebuild a business object and its dependent objects.

The task of the lower layer is to mediate between the integrator and the different processes offered by the manufacturers, i.e. to adapt the sequence of messages that is expected by a manufacturer system with the sequence that is sent out by the dealer. Furthermore, the process adaptor reacts to unavailable services e.g. by invoking alternative services.

All three components have been developed with a model-driven approach. In a first step during design-time, platform independent models have been created for each component. E.g., for the process adaptor, the metamodel PIM4SOA (Platform Independent Model for Service Oriented Architectures) [3] was used to define a connection between the processes of the dealer and manufacturer; for the schema adaptor, Semaphore[1] was used to graphically map the entities and attributes of one data model to the corresponding entities and attributes in the other model. From these models, executables were generated, which have been applied in a second step during run-time to process the data on a concrete platform. E.g., for the process adaptor, the generated process models are executed as software agent on Jack [4], an agent platform based on the BDI-agent theory (*belief-desire-intention*, [5]).

In the following, we will describe this approach in detail and discuss advantages and problems.

4 Mediating Services for Cross-Organisational Business Processes

As can be seen from the description of the scenario, the setting includes a complex interaction between the partners. The design of such a scenario implies a number of problems which have to be solved:

- the different partners (may) expect different atomic protocol steps (service granularity)
- the partners expect and provide different data structures
- 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

These are typical interoperability problems occuring in cross-organisational scenarios which in our case have to be tackled with solutions for SOAs. A core

[1] `http://www.modelbased.net/semaphore`

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. It turned out that these approaches fit nicely together, as e.g. the PIM4SOA metamodel and the agents' metamodel bear a striking resemblance to each other.

Hence, the first problem is solved by specifying a collaborative protocol which allows adapting to different service granularities. 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.

Scalability is envisaged 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.

Finally, 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).

PIM4SOA: A Platform-Independent Model for SOAs

The PIM4SOA is a visual platform-independent model (PIM) which specifies services in a technology independent manner. It represents an integrated view of SOAs in which different components can be deployed on different execution platforms. The PIM4SOA model helps us to align relevant aspects of enterprise and technical IT models, such as process, organisation and products models. The PIM4SOA metamodel defines modelling concepts that can be used to model four different aspects or views of a SOA:

Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity. Service architectures are composed of functions provided by a system or a set of systems to achieve a shared goal. The service concepts of the PIM4SOA metamodel have been heavily based on the Web Services Architecture as proposed by W3C [6].

Information is related to the messages or structures exchanged, processed and stored by software systems or software components. The information concepts of the PIM4SOA metamodel have been based on the structural constructs for class modelling in UML 2.0 [7].

Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc. The process concepts of the PIM4SOA metamodel have been founded on ongoing standardization work for the Business Process Definition Metamodel (BPDM) [8].

Non-functional aspects can be applied to services, information and processes. Concepts for describing non-functional aspects of SOAs have been based on the UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms [9].

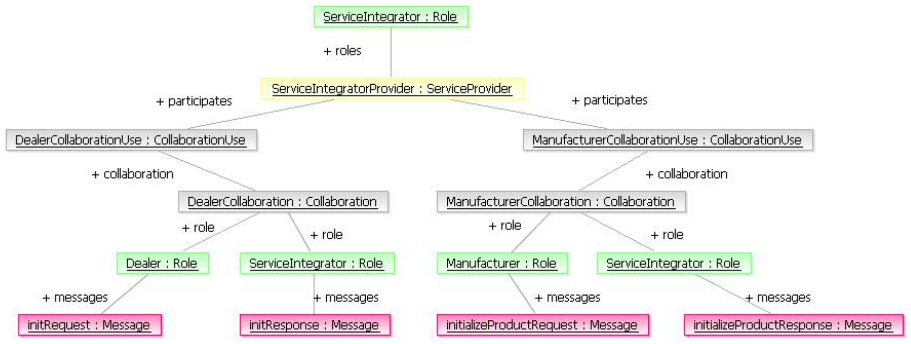
ServiceIntegrator : Role
+ roles
ServiceIntegratorProvider : ServiceProvider
+ participates
+ participates
DealerCollaborationUse : CollaborationUse
ManufacturerCollaborationUse : CollaborationUse
+ collaboration
+ collaboration
DealerCollaboration : Collaboration
ManufacturerCollaboration : Collaboration
+ role
+ role
+ role
+ role
Dealer : Role
ServiceIntegrator : Role
Manufacturer : Role
ServiceIntegrator : Role
+ messages
+ messages
+ messages
+ messages
initRequest : Message
initResponse : Message
initializeProductRequest : Message
initializeProductResponse : Message

Fig. 3. PIM4SOA Model for Pilot (part)

Via model-to-model transformations, PIM4SOA models can be transformed into underlying platform-specific models (PSM) such as XSD, Jack BDI-agents or BPEL.

The business protocol between dealer (dealer software), integrator and manufacturers is specified as PIM4SOA model (see Figure 3). In order to execute collaborative processes specified on the PIM level, the first step consists of transforming PIM4SOA 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 PIM4SOA metamodel are mapped to BDI-agents represented by the Jack metamodel (JackMM). For detailed information on JackMM we refer to [10].

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

The partner models are transformed to a Jack agent model with the model-to-model transformation developed in ATHENA. The following sketch outlines the metamodel mappings (see Figure 4, for more details, cf. e.g. [10]).

A *ServiceProvider* (i.e. ServiceIntegratorProvider in Figure 3) is assigned to a *Team* (which is an extension of an *Agent*). The name of the *ServiceProvider* coincides with the name of the *Team*, its roles are the roles the *Team* performs. Furthermore, the team makes use of the roles specified as bound roles in the *CollaborationUse* (i.e. Dealer and Manufacturer), in which it participates. For each of these roles, we additionally introduce an atomic *Team*. The *Process* of the *ServiceProvider* 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 and DealerTeam in Figure 5) provide. Finally, *Messages* that are sent by the roles we already have transformed are mapped to *Events* in JackMM.

The process integrator and the manufacturers are modelled as Web services. Their interface is described by WSDL descriptions publishing the platform as

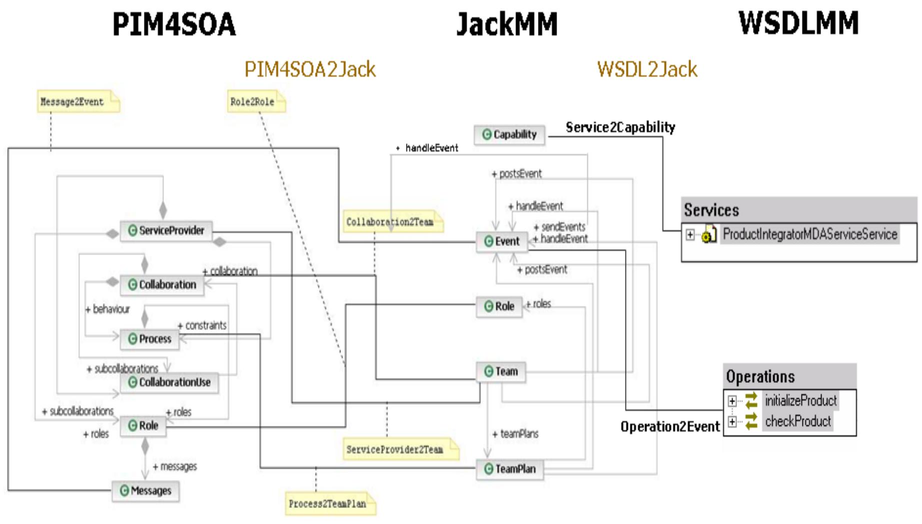

Fig. 4. PIM4SOA and WSDLMM to JackMM transformation

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 (using Apache Axis). 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).

By executing the model transformations we automatically derive the JackMM model illustrated in Figure 5 (for more details, cf. [10]). It should be stressed that these model transformations and the respective code generation can be done automatically if (i) the PIM4SOA 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.

5 Advantages of Agent-Based SOAs

The similarities between agent architectures and SOAs have already been recognized (e.g. [11]). In fact, the strong correspondence between the PIM4SOA and the JackMM confirms this observation. 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 [12] between *fixed*, *semi-fixed*, and *explorative composition* is useful. Fixed composition can be done with

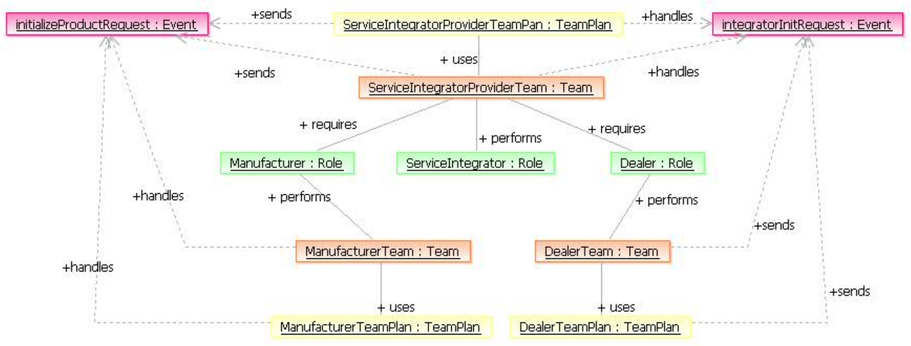

Fig. 5. Jack Model generated from PIM4SOA (part)

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 [13].

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 PIM4SOA 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 [14]. In the case that any unstructured processes shall be executed, an approach like described in [15] 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. [11] cover several important aspects, [16] propose the application of agents for workflows in general. [17] and [18] present a technical and conceptual integration of an agent platform 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 tightly integrating 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. [17].

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.
- The PIM4SOA metamodel is sufficient for modelling basic exchange patterns but needs to be more expressive.
- 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.

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. We argued that agent-based SOAs provide additional advantages over standard process execution environments.

References

1. Schulz, K., Orlowska, A.: Facilitating cross-organisational workflows with a workflow view approach. Data and Knowledge Engineering 51(1), 109–147 (2004)
2. Klein, M., Greiner, U., Genßler, T., Kuhn, J., Born, M.: Enabling Interoperability in the Area of Multi-Brand Vehicle Configuration. In: I-ESA 2007. 3rd International Conference on Interoperability for Enterprise Software and Applications (2007)
3. Benguria, G., Larrucea, X., Elvesæter, B., Neple, T., Beardsmore, A., Friess, M.: A Platform Independent Model for Service Oriented Architectures. In: I-ESA 2006. 2nd International Conference on Interoperability of Enterprise Software and Applications (2006)
4. JACK Intelligent Agents: The Agent Oriented Software Group (AOS) (2006), `http://www.agent-software.com/shared/home/`
5. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents within a BDI-Architecture. In: Allen, J., Fikes, R., Sandewall, E. (eds.) KR91. 2nd International Conference on Principles of Knowledge Representation and Reasoning, pp. 473–484. Morgan Kaufmann publishers Inc., San Mateo, CA, USA (1991)
6. W3C: Web Services Architecture, World Wide Web Consortium (W3C), W3C Working Group Note (February 11, 2004), `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/`
7. OMG: UML 2.0 Superstructure Specification, Object Management Group (OMG), Document ptc/03-08-02 (August 2003), `http://www.omg.org/docs/ptc/03-08-02.pdf`
8. IBM: Adaptive, Borland, Data Access Technologies, EDS, and 88 Solutions, "Business Process Definition Metamodel - Revised Submission to BEI RFP bei/2003-01-06", Object Management Group (OMG), Document bei/04-08-03 (August 2004), `http://www.omg.org/docs/bei/04-08-03.pdf`
9. OMG: UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Object Management Group (OMG), Document ptc/04-09-01 (September 2004), `http://www.omg.org/docs/ptc/04-09-01.pdf`

10. Hahn, C., Madrigal-Mora, C., Fischer, K., Elvesæter, B., Berre, A.J., Zinnikus, I.: Meta-models, Models, and Model Transformations: Towards Interoperable Agents. In: Fischer, K., Timm, I.J., André, E., Zhong, N. (eds.) MATES 2006. LNCS (LNAI), vol. 4196, Springer, Heidelberg (2006)
11. Singh, M., Huhns, M.: Service Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Chichster, West Sussex, UK (2005)
12. Yang, J., Heuvel, W., Papazoglou, M.: Tackling the Challenges of Service Composition in e-Marketplaces. In: RIDE-2EC 2002. 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (2002)
13. Sirin, E., Parsia, B., Wu, D., Hendler, J.A., Nau, D.S.: HTN planning for Web Service composition using SHOP2. J. Web Sem. 1, 377–396 (2004)
14. Mendling, J., Lassen, K., Zdun, U.: Transformation Strategies between Block- Oriented and Graph-Oriented Process Modelling Languages. In: Lehner, F., Nekabel, H., Kleinschmidt, P. (eds.) Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Berlin (2006)
15. Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.H.M.: Translating Standard Process Models to BPEL. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, Springer, Heidelberg (2006)
16. Vidal, J.M., Buhler, P., Stahl, C.: Multiagent systems with workflows. IEEE Internet Computing 8(1), 76–82 (2004)
17. Greenwood, D., Calisti, M.: Engineering Web Service – Agent Integration. In: IEEE Systems, Cybernetics and Man Conference, the Hague, Netherlands, October 10-13, 2004, pp. 10–13. IEEE Computer Society Press, Los Alamitos (2004)
18. Dickinson, I., Wooldridge, M.: Agents are not (just) web services: Considering BDI agents and web services. In: SOCABE. AAMAS 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (2005)