Programming Software Agents as Designing Executable Business Processes: A Model-Driven Perspective

Jörg P. Müller[1], Bernhard Bauer[2], and Thomas Friese[3]

[1] Siemens AG Corporate Technology
Intelligent Autonomous Systems
Otto-Hahn-Ring 6, D-81739 München, Germany
joerg.p.mueller@siemens.com

[2] Programming of Distributed Systems,
Institute of Computer Science, University of Augsburg
D-86135 Augsburg
bernhard.bauer@informatik.uni-augsburg.de

[3] Dept. of Mathematics and Computer Science
University of Marburg
D-35032 Marburg, Germany
friese@informatik.uni-marburg.de

Abstract. The contribution of this paper is fourfold. First, we sketch an architecture of agent-enabled business process management that cleanly separates between agent capabilities, business process modeling, and the modeling of services that are employed to actually implement processes. Second, we demonstrate how the Model-Driven Architecture (MDA) paradigm can be beneficially employed at all three layers of the agent-enabled business process architecture. Third, we describe an instance of a platform independent model based on the new UML2 standard, and sketch a mapping to a platform dependent model based on the Business Process Execution Language for Web Services (BPEL4WS). Fourth, we point out the relationship between the programming of multiagent systems, and the design of agent-enabled business processes. Our key thesis is that designing business processes based on an agent-enabled business processes modeling and enactment framework is a useful way of programming agents – a way that may help agent technology to gain much wider acceptance in industry.

1 Introduction

Over the past few years, enterprises have undergone a thorough transformation in reaction to challenges such as globalization, unstable demand, and mass customization. A key to maintaining competitiveness is the ability of an enterprise to describe, standardize, and adapt the way it reacts to certain types of business events, and how it interacts with suppliers, partners, competitors, and customers. Today, virtually all larger enterprises describe these procedures and interactions in terms of *business processes,* and invest huge efforts to describe and standardize these processes.

The trend to process-centered modeling and operation of enterprises brings new opportunities for software technologies that support the monitoring, management, and optimization of processes. On the other hand it requires software technologies to relate to business processes, to understand them and to hook into them where required.

Software agents are computer systems capable of flexible autonomous action in a dynamic, unpredictable and open environment [16]. These characteristics give agent technology a high potential to support process-centered modeling and operation of businesses. Consequently, there have been various research efforts of using agent technology in business process management, one of the earliest examples being ADEPT [15]. Recently, agent technology has also started to attract business analysts in the context of adaptive supply network management [20].

However, the focus of ADEPT was essentially focused on communication and collaboration in business process management. Its outcome was a research prototype, not a usable business process support platform. Migrating agent technology successfully to business applications requires the provision of end-to-end solutions that integrate with standards, that preserve companies' investment in hardware, software platforms, tools, and people, that enables the incremental introduction of new technologies (see also [18]), and that can deal gracefully with moving to new technology platforms.

One main objective of this paper is to sketch an architecture of agent-enabled business process management that cleanly separates between agent capabilities, business process modeling, and the modeling of services that are employed to actually implement processes. We regard software agents as software components primarily responsible for

- the intelligent evaluation, selection and customization of business processes and activities matching service requests;
- and for the robust and flexible enactment of these processes or activities, including monitoring, exception and failure handling, and decision support.

Enactment of business processes relies on an underlying service layer, an information and communication runtime infrastructure providing the possibility of registering and discovering services and of communicating service requests and responses.

A second objective is to demonstrate how the Model-Driven Architecture (MDA) [17] paradigm can be beneficially employed at all three layers of the agent-enabled business process architecture, in order to allow system designers to focus on the functionality and behavior of a distributed application or system, independent of the technology or technologies in which it will be implemented.

Based on this model-driven approach, our third objective is to demonstrate an instance of a platform independent model based on the new UML2 standard [22], extending the scope of the Unified Modeling Language to the design of processes, and sketch a mapping to a platform dependent model based on the Business Process Execution Language for Web Services (BPEL4WS) [13]. We discuss design-time and run-time aspects of a methodology for agent-enabled business process modeling.

Finally, the fourth objective of this paper is to point out a relationship between the programming of multiagent systems, which is the key topic of the volume at hand, and the design of agent-enabled business processes. Our key thesis here is that designing business processes based on an agent-enabled business processes modeling and enactment framework can be regarded as one way of programming agents – one way that may help agent technology to gain much wider acceptance in industry.

This paper is not a primarily a technical paper although it does contain technical elements. Our primary intention is to bring forward a position sketching the need and requirements for, and key aspects of, an architecture and development framework for programming agents for industrial applications, which in our opinion is not sufficiently reflected in most existing research on agent-oriented software engineering and agent programming. Many of the details remain to be fleshed out in future research, but we feel it is worth pointing them out. We hope that the reader agrees with this.

The structure of the paper is as follows: In Section 2, we provide a conceptual architecture for agent-enabled business processes and set the technological background. Section 3 provides an application example that will serve as a demonstration vehicle throughout the paper. In Section 4, a model-driven design methodology for agent-enabled business processes is provided including examples for platform-independent and platform-specific models. Section 5 then outlines a runtime architecture for enactment of agent-enabled business processes. We conclude in Section 6 with some discussion of future research opportunities.

2 Background

In this section, we describe a conceptual architecture defining the relationship between web services, business processes, and software agents.

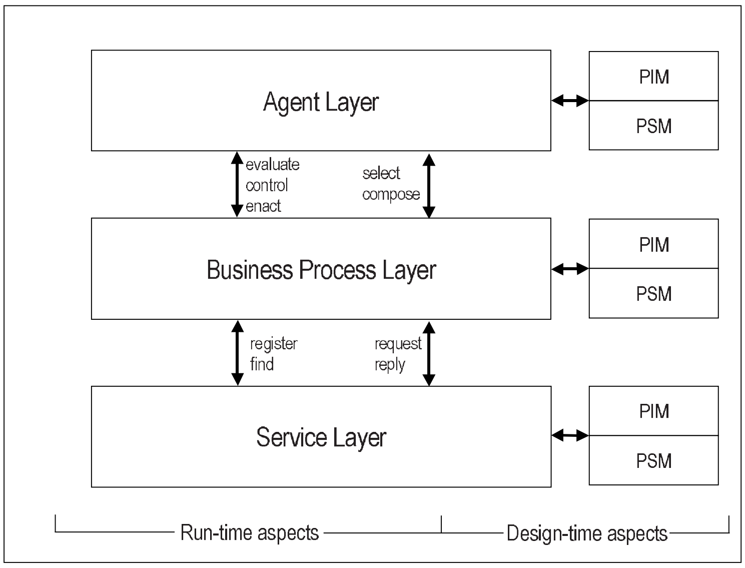

Fig. 1. Conceptual architecture for agent-enabled business processes

2.1 Conceptual Architecture

Figure 1 illustrates the essence of our conception in an abstract three-tier architecture. We regard software agents as software components primarily responsible for the intelligent evaluation, selection and customization of business processes and activities matching service requests and for the robust and flexible enactment of these processes or activities, including monitoring, exception and failure handling, and decision support. Enactment of business processes relies on an underlying service layer, an information and communication runtime infrastructure providing the possibility of registering with and discovering services and of communicating service requests and responses. For grounding this conceptual architecture in a real IT environment, we propose the Model-Driven Architecture (MDA) approach (see [17] and [25]) in order to achieve a separation of business and application logic from underlying platform technologies. The core idea of the MDA is to first describe a software artifact by a platform independent model (PIM), and then provide mappings to a number of platform-specific models (PSM). Our claim is that MDA can be applied beneficially at all three layers of our conceptual architecture. At the agent layer, it can be used to specify the behavior of agents, there strategies and objective functions in an abstract manner. At the business process layer, it can be used to accommodate the use of different business process representations (such as SCOR, event-driven process chains, or activity diagrams of UML 2.0). Finally, at the service level, it can be used to ground the architecture on different service-oriented platforms, such as web services/.NET, Java RMI, Peer-to-Peer Platforms, or agent communication platforms available as part of FIPA-compliant agent infrastructure (e.g., Jade/Leap).

In the following, we provide a short overview of the major technological aspects at the different conceptual layers.

2.2 Model-Driven Architecture

The Model Driven Architecture (MDA) (for details see [25]; this section is also based on this reference) is a framework for software development driven by the Object Management Group (OMG). Key to MDA is the importance of models in the software development process. Within MDA the software development process is driven by the activity of modeling the software system. The MDA development process does not look very different from a traditional lifecycle, containing the same phases (requirements, analysis, low-level design, coding, testing, and deployment). One of the major differences lies in the nature of the artifacts that are created during the development process. The artifacts are formal models, i.e. models that can be understood by computers. The following three models are at the core of the MDA:

- **Platform Independent Model (PIM):** This model is defined at a high level of abstraction; it is independent of any implementation technology. It describes a software system that supports some business. Within a PIM, the system is modeled from the viewpoint of how it best supports the business. Whether a system will be implemented on a mainframe with a relational database, on an agent platform or on an EJB application server plays no role in a PIM.
- **Platform Specific Model (PSM):** In the next step, the PIM is transformed into one or more PSMs. It is tailored to specify a system in terms of the implementation constructs available in one specific implementation technology. E,g. an agent

PSM is a model of the system in terms of an agent platform. A PIM is transformed into one or more PSMs. For each specific technology platform a separate PSM is generated. Most of the systems today span several technologies; therefore it is common to have many PSMs with one PIM.

- **Code:** The final step in the development is the transformation of each PSM to code. Because a PSM fits its technology rather closely, this transformation is relatively straightforward.

2.3 Service Layer: Web Services

Describing software architecture in a service-oriented fashion, while being increasingly popular, is not a new idea; it is what CORBA has been about more than a decade ago. Recently, the concept of web services has given new momentum to service-oriented architecture. Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web using existing web protocols and infrastructure. The core web service standards are the Web Service Definition Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery and Integration (UDDI). The combination of relative simplicity, platform independence, and leveraging of HTTP positions web services to become an important architecture for wide-scale distributed computing [8].

2.3.1 Web Service Definition

WSDL is an XML based specification for describing what a program module does (interface description), what the result of the module's activity is, and how to communicate with it. A WSDL document resides at a URL location, e.g. at a UDDI, and is linked to the module, which itself may reside at any location. Web Services exchange information through SOAP messages. SOAP is a protocol for Remote Procedure Call / Remote Method Invocation over HTTP. SOAP uses XML to define the message format and how variables and parameters required by the program are sent to it to invoke its methods. The program in turn, sends the results of its process back to the request originator in another SOAP message. Because HTTP is one of the transport mechanisms used by SOAP, a Web Service method call can be made to and from any Web enabled computer anywhere in the world.

2.3.2 Web Service Choreography

According to [23], web service choreography describes the specification of the interaction (i.e., ordering of messages) among a collection of services from the perspective of one service or a collection thereof. Web service choreography allows applications to combine existing web services in order to obtain more elaborated value-added web services. Note that choreography deals with the definition of these interactions, not with their execution. Several standards are currently under development for the definition of languages for Web Service composition or Web Service Choreography: For instance, IBM and Microsoft have developed process languages, the Web Services Flow Language (WSFL) and XLANG, respectively. These have been merged into the Business Process Execution Language for Web Service BPEL4WS, see Section 2.4.

2.4 Business Process Layer

The notion of service choreography logically leads to the notion of business processes. A business process is a group of business activities undertaken by an organization in pursuit of a common goal. The process-driven approach has become the predominant means of describing and structuring corporate IT and information systems. Hence, we argue that the notion of business processes is a mostly adequate basis for specifying agents and multiagent systems.

Driven by the availability of web services platforms (see Section 2.3), a number of architectures, presentations, and platforms for web-based business processes have been proposed, starting with the already mentioned BPEL4WS, but also including the Business Process Modeling Language (BPML) defined by the Business Process Management Initiative BPMI [4]. Another example is the ebXML *Business Process Specification Schema* [5], providing a standard framework by which business systems may be configured to support execution of business collaborations. We use BPEL4WS in the examples used throughout this paper; however, the adherence to MDA ensures transferability to other process models and infrastructure.

2.5 Agent Layer

From a sufficiently abstract perspective, a business process is essentially a computer program; composing a business process is very similar to program synthesis, and enacting a business process is program execution. Automated business process composition is not too different from artificial intelligence planning, which implies that AI planning techniques are applicable but that at the same time we should not hope for efficient general solutions of the business process composition problem, as they most probably do not exist. However, business processes are more than plans or programs: they incorporate knowledge about organizations, and they contain tasks that can only be solved by or in collaboration with humans. This is where we believe agents come into play. We claim that the role of agents in the context of business processes is threefold:

- Agents can be used to support the evaluation of existing business processes at request time, leading to recommendation regarding the assignment of a business process to an incoming service request.
- Agents can combine planning methods with knowledge about structure, authorities, and competencies of organizational units to modify existing business processes, or to create new instances of business processes from scratch.
- Agents can monitor, manage, and execute business processes in a robust and intelligent fashion, involving content- and capability-based routing of tasks, autonomous initiation and monitoring of process execution, situated recognition of and reaction to failures or other critical events, and longer-term self-optimization.

The agent layer in our conceptual architecture needs to provide methods and tools to model process-aware agents and to support (semi-)automated evaluation, selection, composition, adaptation, and robust execution of business processes.

Our perspective extends Huhns's [12] view on the relationship of agents and Web Services in the following way: While Huhns describes agents as intelligent web services, we claim that agents should be regarded primarily as driven by and responsible for business processes. While executing or monitoring business processes, they may interact or negotiate with (web) services that may again trigger agents responsible for the business process initiated by the service request. We believe that by decoupling the notion of process from the notion of service and by using the model-driven approach, we can achieve more modular, more re-usable and technology-independent specifications of multi-agent systems.

In this paper, we shall not provide a full specification of the agent layer of our architecture. This is beyond the scope of this overview paper and is a topic for future research. However, in Section 5 we sketch an agent-based process execution layer.

In the remainder of this section, we note a few requirements for an agent-enabled business process architecture.

- *Semantic process definitions*: Evaluating, composing and adapting processes and monitoring process execution require a machine-understandable description of the behavior of processes and their activities. The use of semantic markup languages, like DAML-OIL or OWL for the definition of ontologies and DAML-S for semantic service descriptions, is one obvious approach to add information on top of pure syntactical Web Services descriptions.
- *Self-organization, self-description and self-configuration:* The ability of organizations to adapt the design and execution of collaborative processes based on semantic information of the agents/organizations, the environment and the services available in its surroundings are other key functions that distinguish agents. In this context, we advocate a two-level approach where peer-to-peer concepts for resilience, de-central coordination, and self-organization are applied to achieve a short-term, reactive flexibility, and where multiagent coordination concepts such as multiagent planning are used for longer-term adaptation and self-organization (see Section 5).
- *Flexible interaction:* The availability of an open interaction model enables interaction among previously unknown parties and the tailoring of interactions to the partners capabilities and preferences by using automated negotiation. The big challenge in this context is how economically motivated agents can be equipped with domain-specific valuation models enabling them to efficiently engage in negotiations from the perspective of the business process or the organization(s) they represent.
- *Individualization*: The availability of domain-specific and extensible representations of context, profiles, and preferences is also a prerequisite for customization of e.g. services and goods to the needs or context of a person. Individualization takes the specific context or situation of a user into account and can be applied at different levels: User context, service level, interface level and infrastructure level (see e.g., [10]).

3 Application Example

To illustrate our approach, we introduce an application example taken from the UML 2.0 Superstructure specification (see Figure 2); a similar example is given in [11]. The original example describes a Supply Chain scenario with different types of activities attributed to different organizational units. In this paper, agents represent the individual units.

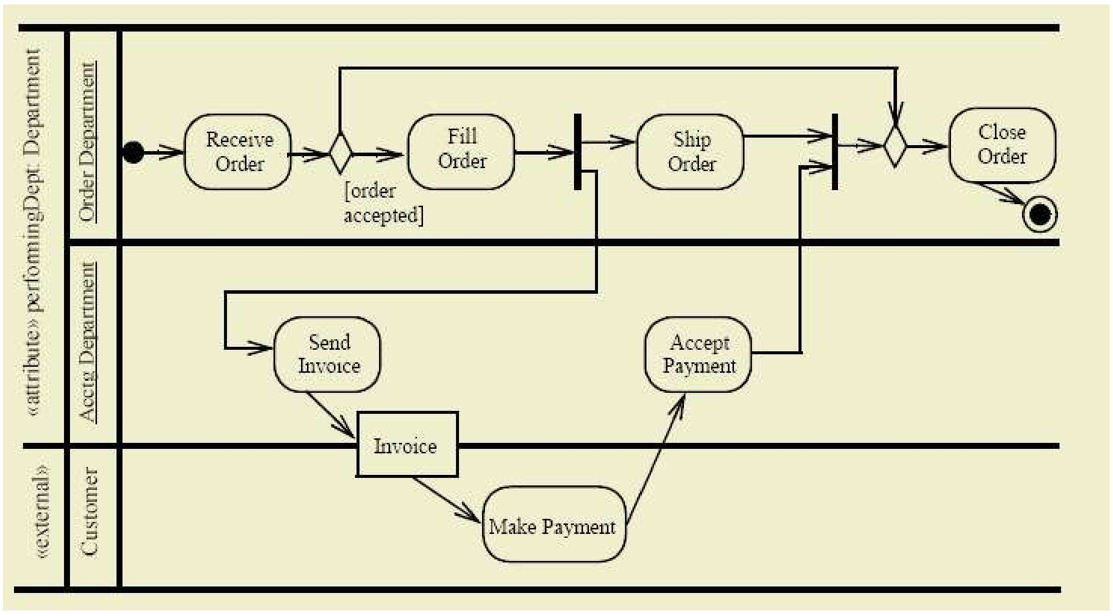

Fig. 2. UML activity diagram for the order processing example

Round-cornered rectangles denote actions, diamonds denote alternatives, rectangles denote comments, texts in brackets denote constraints. The order processing activity diagram is partitioned into "swim lanes." The three swim lanes denote the responsibilities for the different portions of the process.

The order agent initiates the process by receiving an order using *Receive Order*. This order can contain information about e.g. price and delivery date for the item corresponding to the Customer's constraints. The order agent checks the order and either rejects the order with the effect of closing the order (*Close Order*) or accepts it. In the latter case an order is filled (*Fill Order*) and two sub-processes are triggered, namely the order is shipped (*Ship Order*) and the accounting department produces the invoice and sends the invoice to the external customer (*Send Invoice*) who makes the payment (*Make Payment*). Payment is then accepted (*Accept Payment*) by the accounting department. If both sub-processes succeed the order is closed (*Close Order*).

4 Modeling Agent-Enabled Business Processes

In the remainder of this paper, we describe an instance of the conceptual architecture for agent-enabled business processes in Section 2.1. As indicated in, instantiating this architecture entails dealing with both design-time and run-time issues. In this Section, we focus on design-time aspects, i.e., at the modeling of agent-enabled business processes. In Section 5, we sketch a corresponding run-time model.

4.1 Overview

A straightforward method of implementing the conceptual architecture described in Section 2.1 would be to independently define three models corresponding to the three layers of the conceptual architecture: an agent model, a business process model, and a service model. We believe that this could be done by extending most existing methodologies, such as GAIA [24] or Tropos [9] by the business process layer, connecting agent model and service model. While doing this seems a worthwhile exercise, our approach in this paper is different. Instead of starting from the green field and describing a completely new unified framework to design the three models and their interrelationship, we set out from existing representation standards, platforms, and tools, and develop a pragmatic and simplified structure.

In this simplification, the original three-layer hierarchy is mapped into a two-stage procedure as illustrated in Figure 3. In particular, the agent layer is not explicitly represented in our development process. Rather, our idea is to enhance the service- and process-related elements with appropriate metadata and functionality. An instantiation of our model that fully complies with the three-tier architecture is left for future work.

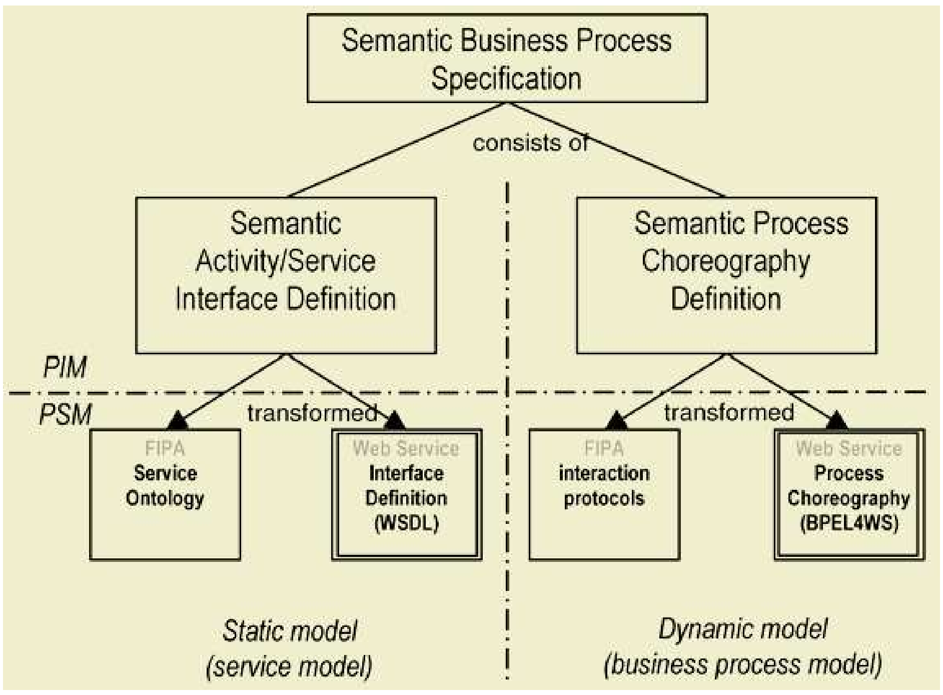
Semantic Business Process Specification
consists of
Semantic Activity/Service Interface Definition
Semantic Process Choreography Definition
PIM
PSM
transformed
transformed
FIPA Service Ontology
Web Service Interface Definition (WSDL)
FIPA interaction protocols
Web Service Process Choreography (BPEL4WS)
Static model (service model)
Dynamic model (business process model)

Fig. 3. Overview of model-driven development methodology for agent-enabled business processses

Figure 3 illustrates the top-down development process starting with a semantic business process specification using and extending UML 2.0 activity diagrams (similar to the one depicted in for our example). This specification consists of two models:

- a static model, which is essentially the service model in our conceptual architecture, even though enhanced with metadata, such as the description of pre- and post-conditions for service invocation, and with exception definitions;
- a dynamic model, which is essentially the business process layer in the conceptual architecture, even though enhanced with planning methods and exception handling capabilities based on the semantic service descriptions.

Each of these two models is described by one platform-independent model and one or more platform-specific models. We propose the usage of UML 2.0 for the Platform-Independent Model both for service definition and process choreography definition. In addition, we provide exemplary mappings to Platform Specific Models, using WSDL to specify the services/activities and using BPEL4WS for the process choreography. In the following, the different models and mappings are investigated in more detail. However because of lack of space we will mainly focus on WSDL and BPEL4WS.

This view is similar to that of DAML-S, however from a service specification perspective. The relevant key elements of DAML-S are:

- **Service Profiles** provide a means to describe the services offered by the providers, and the services needed by the requesters. Some properties of the profile provide human-readable information like service name, a textual description or contact information (phone, fax,…). Moreover the functionality is defined by inputs of the service, outputs of the service, preconditions of the service; and effects of the service. This is comparable with the Service Interface Definition in Figure 3.
- **Service Model** describes the processes and their composition with the related properties parameter, input, (conditional) output, participant, precondition, and (conditional) effect as well as the binding. This is comparable with the Semantic Process Choreography Definition in Figure 3.

Service Grounding specifies the details of how to access the service - details having mainly to do with protocol and message formats, serialization, transport, and addressing. A grounding can be thought of as a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service.

For the platform independent model we propose to use Activity Diagrams (as shown in) provided in UML 2.0 to model business processes. An activity diagram depicts behavior using a control and data-flow model. In particular it describes activities and flows in different details. They are applied e.g. for business process modeling, description of use cases and in particular defining implementation relations (i.e., refinements) of operations.

4.2 Semantic Activity/Service Interface Definition

In this section we shall present a platform-independent model and an example of a platform-specific instantiation of the semantic activity/service interface definition.

4.2.1 Platform-Independent Model

Activity diagrams allow the specification of activities, as depicted in Figure 4, as a specification of a parameterized behavior that is expressed as a flow of execution by sequencing of subordinate units (whose primitive elements are individual actions).

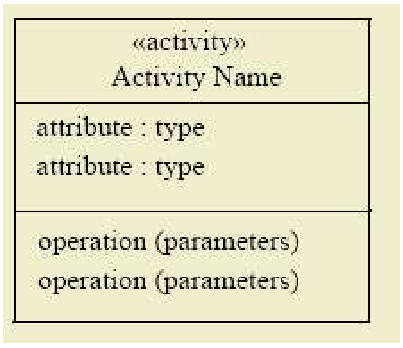

Fig. 4. Notation for activities in UML 2.0

We add the required semantic metadata information to the platform-independent model by using stereotypes in the activities, namely <<pre-conditions>>, <<post-condition>>, <<effects>> for defining the pre-conditions, post-conditions, and conditional affects of an activity. The underlying notation could be e.g. OCL from the UML specification or some Semantic Web language. This is illustrated in Figure 5.

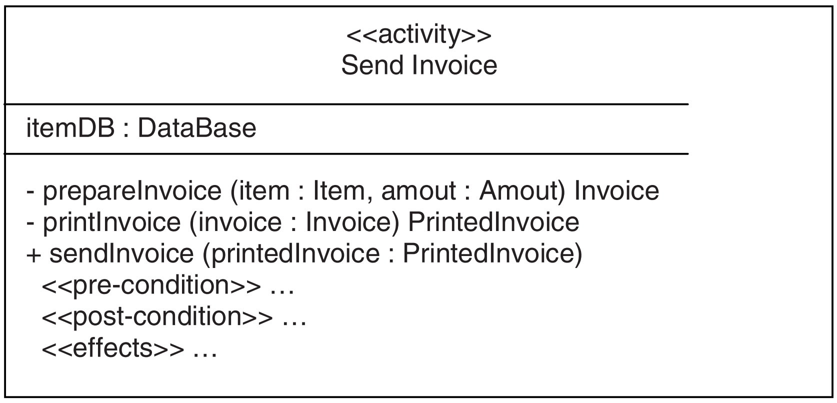
<<activity>>
Send Invoice
itemDB : DataBase
- prepareInvoice (item : Item, amout : Amout) Invoice
- printInvoice (invoice : Invoice) PrintedInvoice
+ sendInvoice (printedInvoice : PrintedInvoice)
<<pre-condition>> …
<<post-condition>> …
<<effects>> …

Fig. 5. Semantically enhanced activity definition

4.2.2 A Platform Specific Model Based on WSDL

4.2.2.1 Modeling Web Services Interfaces

The static description of a Web Service in WSDL is mainly concerned with defining its interfaces. A WSDL definition consists of the following parts: *Ports* define the concrete Internet address of a Web Service, i.e., its URL and communication port; *Services* cover several ports and define the physical address of an end point; *Messages* are the format for a specific information exchange, where `request` and `response` are two dedicated messages; *PortTypes* group messages to logical operations; *Bindings* bind PortTypes to their implementation, usually SOAP, and define the concrete interface of a Web Service. In [1] UML is applied for modeling WSDL descriptions.

We follow this approach for modeling WSDL descriptions. Since BPEL4WS does not require us to deal with concrete addresses and physical addresses, we will not cover *Ports* and *Services*.

One class is defined for an overall *WSDL description*. This class is stereotyped with <<wsdl description>> to mark it as a WSDL description. Each *element* of a non-complex type is modeled by stereotyped (<<element>>) attribute. *Complex types* of elements are modeled as separate classes with stereotype <<element>>.

Messages are depicted by <<wsdl message>> stereotyped operations in class diagrams; parameters denote the *part name:type* information of a message.

For each *PortType* a class is defined stereotyped with <<wsdl portType>>. This *PortType* is attached to a <<wsdl description>> class using aggregation; for each operation an operation of the class is used.

Again, for each *Binding* a class is defined stereotyped with <<wsdl binding>> and attached to a service by aggregation with <<port>> stereotype and for each operation an operation is applied.

Figure 6 illustrates an excerpt of our example specification in UML. The WSDL description has one element, i.e., *OrderDepartmentDescription*. It accepts different messages, including *receiveOrder*() and *sendInvoice*(). The parameters and their types are omitted in the figure, but can be found in the following WSDL description. The *portType sendInvoice* supports one operation, named *sendInvoiceToCustomer*(). This WSDL specification can be derived from the platform-independent model, giving additional information describing e.g. the interfaces and dependencies in detail. This example gives an idea of how the transformation can be performed.

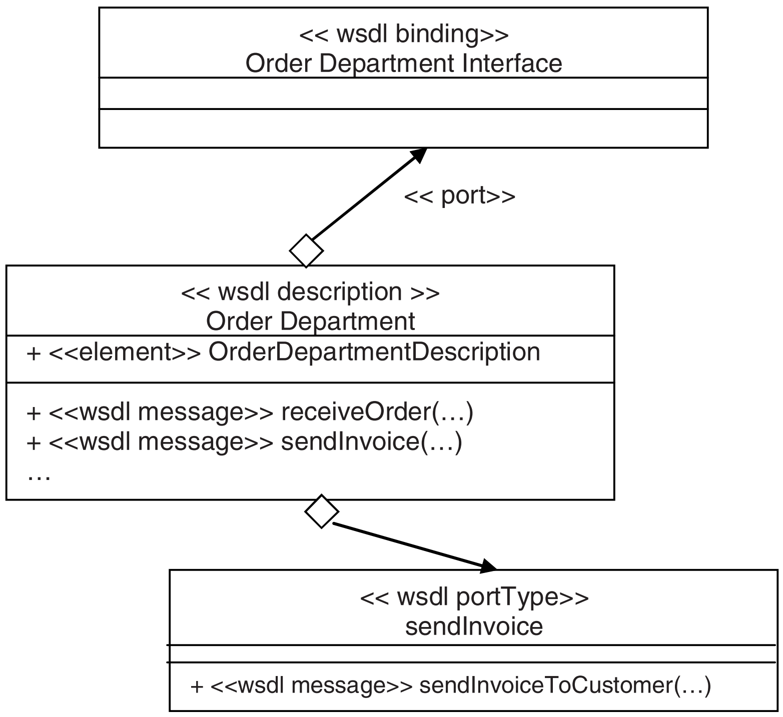
<< wsdl binding>>
Order Department Interface
<< port>>
<< wsdl description >>
Order Department
+ <<element>> OrderDepartmentDescription
+ <<wsdl message>> receiveOrder(…)
+ <<wsdl message>> sendInvoice(…)
…
<< wsdl portType>>
sendInvoice
+ <<wsdl message>> sendInvoiceToCustomer(…)

Fig. 6. UML and WSDL

4.2.2.2 Mapping UML Specifications to WSDL

The following listing shows an excerpt of the WSDL description obtained from the UML specification :

```
<definitions targetNamespace="…"
        xmlns:…
        […]
        <message name="receiveOrder">
                <part name="item" type="Item"/>
                <part name="amount" type="Amount"/>
        </message>

        <!-- portTypes supported by the Order Department -->
        <portType name="sendInvoice">
                <operation name="sendInvoiceToCustomer">
                        <input message="receiveOrder"/>
                        <output message="sendInvoice"/>
                </operation>
        </portType>
        […]
  </definitions>
```

4.3 Semantic Process Choreography Definition

Based on the service interface definitions described in the previous subsection, this section deals with dynamic aspects of semantic process choreography, i.e. with how processes are composed from activities. We start with the platform-independent modeling of semantic process choreography, and give an exemplary mapping of a PIM to a PSM based on BPEL4WS.

4.3.1 Platform Independent Model

UML 2.0 activity diagrams allow the definition of complex activities (sub-processes) defining the parameters, pre-conditions and post-conditions as shown in Figure 7.

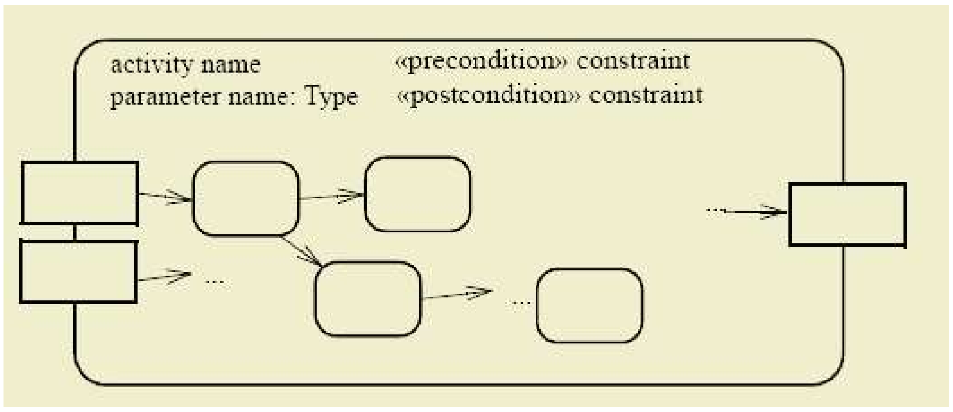

Fig. 7. Activities with parameters and pre- and post-conditions

Moreover, UML2 supports the representation of events that can be used to obtain a basic model of exceptions and exception handling behaviors. An example using the UML Accept Event Action construct is given in Figure 8.

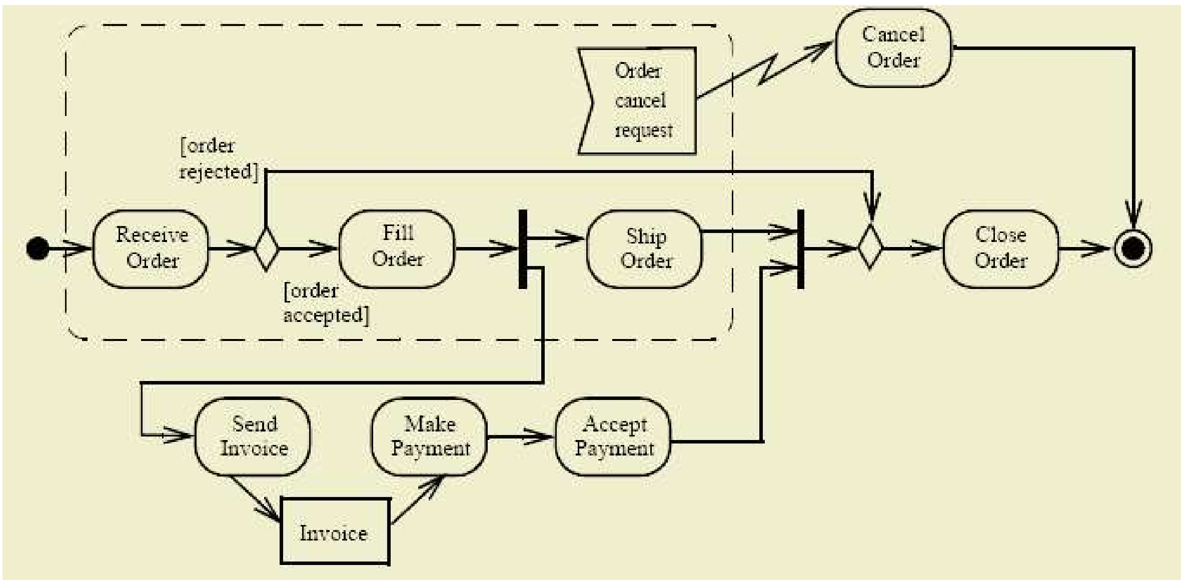

Fig. 8. Representation of events/exceptions in UML2.0

The example describes an event (denoted by the Accept Event Action *Order cancel request*) that can occur during the execution of a part of the process denoted by the dotted rectangle with rounded corners. In this case it comprises the activities *Receive Order*, *Fill Order*, and *Ship Order.* This results in cancellation of the order.

For a semantic grounding of complex activities and to describe meta-data we can add stereotypes <<effects>> and <<exceptions>> to the complex activity descriptions.

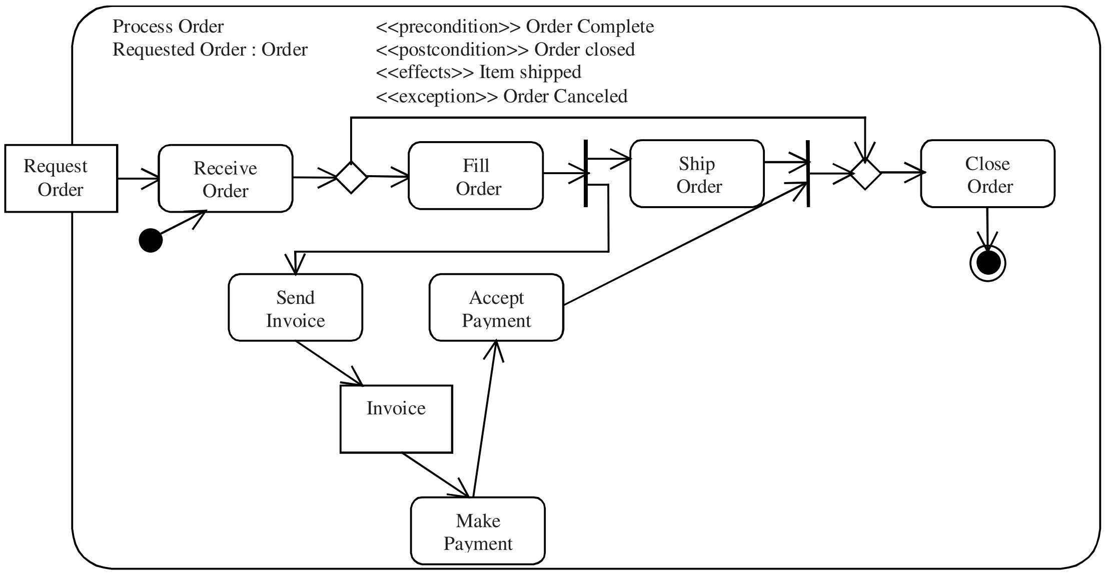

Fig. 9. The example (Figure 2) as a UML2.0 Activity Diagram; swim lanes omitted for simplicity

Figure 9 shows the UML2 activity diagram representation for the original example. The diagram contains the semantic metadata information required for use by the agent layer. It should be noted here that most of this information can actually be encoded by using existing UML2 language features. The only feature used in Figure 9 which is currently not available in UML2 are the <<effects>> and <<exceptions>> stereotype which we added for compatibility with DAML.

4.3.2 A Platform Specific Model: Mapping UML to BPEL4WS

In this section we show an exemplary instantiation of the PIM developed in Section 4.3.1 as a PSM. The basic idea is to implement the PSM by first performing a mapping from the UML activity diagrams into sequence diagrams defining the real message exchanges between the agents, and then to derive BPEL4WS process definitions from the sequence diagrams, which can be done in a relatively straightforward manner. Note, that for a fully automated transformation (code generation) additional information are necessary, which is beyond the scope of this paper. Figure 10 shows the corresponding sequence diagram (we omit the case where the order is rejected).

BPEL4WS (Business Process Execution Language for Web Services; [11]) defines a notation for specifying business process behavior based on Web Services. Processes in BPEL4WS export and import functionality via Web Service interfaces exclusively.

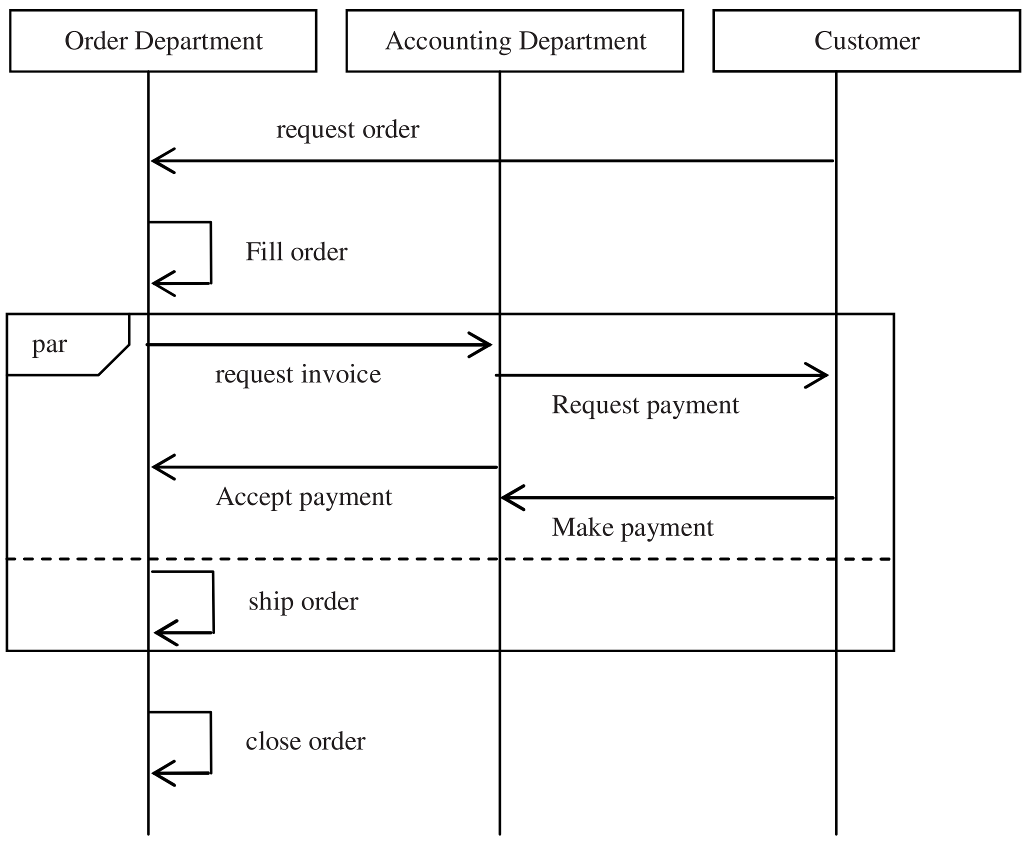

Fig. 10. PSM for Semantic Process Choreography Definition

The process specified in Figure 9 (including its representation as a sequence diagram shown in Figure 10) defines a re-usable process template. E.g., for the order department, it can be written in pseudo-code as follows:

```
while (ordering) do
     receive request order
     invoke fill order
     parallel
        seq
           receive accept-payment
           invoke payment-accepted
        invoke ship-order
     close order
od
```

```
<process name="Process Order"
  targetNamespace=[…]
  xmlns=[…]
  <partners>
      <partner name="AccountingDepartment"
          serviceLinkType="Accounting"
          partnerRole="accounting"
          myRole="order department"/>
  </partners>
  <assign> order-continue-yes=true </assign>
  <while condition="order-continue-yes"
      <sequence>
          <receive partner="Customer"
              portType="OrderInformation"
              operation="orderItem"
              inputContainer="item">
          </receive>
          <invoke partner="OrderDepartment"
              portType="OrderItem"
              operation="order">
          </invoke>
          <flow>
            <sequence>
              <receive partner="AccountingDepartment"
                  portType="SendInvoice"
                  operation="sendInvoiceToCustomer"
                  inputContainer="receivedOrder">
              </receive>
              <invoke partner="OrderingDepartment"
                  portType="PaymentAcception"
                  operation="acceptPayment">
              </invoke>

             </sequence>
              <invoke partner="OrderingDepartment"
                  portType="ShipingOrders"
                  operation="shipOrder">
              </invoke>
          </flow>
          <invoke partner="OrderingDepartment"
              portType="ShipOrdering"
              operation="shipOrder">
          </invoke>
      </sequence>
  </while>
</process>
```

Fig. 11. BPEL4WS Representation of the example process

Figure 11 illustrates the BPEL4WS definition of this example process. The business processes use the following components of BPEL4WS (following [13]):

- *Service Linking, Partners and Service References*: The relationship of a business process with a partner is typically peer-to-peer, requiring a two-way dependency at the service level. The notion of service links is used to directly model peer-to-peer partner relationships. Service links define the relationship with a partner by the message and port types used in the interactions in both directions. However, the actual partner service may be dynamically determined within the process.
- *Messages Properties*: The data in a message consists conceptually of two parts: application data and protocol-relevant data, where the protocols can be business protocols or infrastructure protocols providing higher quality of service, like security and transaction. The business protocol data is usually found embedded in the application-visible message parts, whereas the infrastructure protocols almost always add implicit extra parts to the message types to represent protocol headers that are separate from application data. Business processes might need to gain access to and manipulate both kinds of protocol-relevant data. The notion of message properties is defined as a general way of naming and representing distinguished data elements within a message, whether in application-visible data or in message context. Message properties are defined in a sufficiently general way to cover message context consisting of implicit parts, but the use focuses on properties embedded in application-visible data that is used in the definition of business protocols and abstract business processes. A property definition creates a globally unique name and associates it with an XML Schema type. The intent is to create a name that has greater significance than the type itself.
- *Data Handling:* Business processes model stateful interactions. The state involved consists of messages received and sent as well as other relevant data such as time-out values. The maintenance of the state of a business process requires the use of state variables, which are called containers. Furthermore, the data from the state needs to be extracted and combined in interesting ways to control the behavior of the process, which requires data expressions. Finally, state update requires the notion of assignment. BPEL4WS provides these features for XML data types and WSDL message types. In BPEL4WS, Data handling is performed by using the following features:
    - *Expressions*: BPEL4WS uses several types of expressions: Boolean-valued expressions used for transition conditions, join conditions, while conditions, and switch cases; deadline-valued expressions used with the "until" attribute of onAlarm and wait; duration-valued expressions used for "for" attribute of onAlarm and wait; general expressions based on XPath 1.0 used in assignments. Moreover, BPEL4WS provides an extensible mechanism for the language used in these expressions. The language is specified by the *expressionLanguage* attribute of the process element.
    - *Containers:* Containers provide the means for holding messages that constitute the state of a business process. Containers can hold messages, either received or temporary defined, that act as "temporary variables" for computa-

tion and are never exchanged with partners. Containers can be specified as input or output containers for invoke, receive, and reply activities. At the beginning of a process all containers are not initialized. Containers can be initialized by a variety of means including assignment and receiving a message. Containers can be partially initialized with property assignment or when some but not all parts in the message type of the container are assigned values.

  - *Assignments:* Copying data from one container to another is a common task within a business process. The *assign* activity can be used to copy data from one container to another, as well as to construct and insert new data using expressions.

- *Activities*: BPEL4WS distinguishes between two types of activities: basic and structured activities.
  - *basic activities*: The `receive` construct allows the business process to do a blocking wait for a matching message to arrive. The `reply` construct allows the business process to send a message in reply to a message that was received through a `receive`. The combination of a `receive` and a `reply` forms a request-response operation on the WSDL portType for the process. The `invoke` construct allows the business process to invoke a one-way or request-response operation on a portType offered by a partner. The `assign` construct can be used to update the values of containers with new data. An `assign` construct can contain any number of elementary assignments. The `throw` construct generates a fault from inside the business process. The `terminate` construct allows to immediately terminate a business process. The `wait` construct allows to wait for a given time period or until a certain time has passed. Exactly one of the expiration criteria must be specified. The `empty` construct enables insertion of "no-op" instructions into business processes. This is useful for synchronization of parallel activities, for instance.
  - *structured activities*: The `sequence` construct allows one to define a collection of activities to be performed sequentially. The `switch` construct allows selecting exactly one branch of execution from a set of choices. The `while` construct allows one to indicate that an activity is to be repeated until a certain success criteria has been met. The `pick` construct allows blocking and waiting for exactly a suitable message to arrive or for a time-out alarm to go off. When one of these triggers occurs, the associated activity is executed and the pick completes. The `flow` construct allows specifying one or more activities to be executed in parallel. Links can be used within parallel activities to define arbitrary control structures. The `scope` construct allows defining a nested activity with its own associated fault and compensation handlers. The `compensate` construct is used to invoke compensation on an inner scope that has already completed its execution normally. This construct can be invoked only from within a fault handler or another compensation handler.

5 Agent-Enabled Business Process Enactment

In this section, we sketch an infrastructure for agent-enabled business process enactment. This infrastructure is an instance of the model-driven architecture framework described in Section 4. Figure 12 shows the architecture of the run-time system.

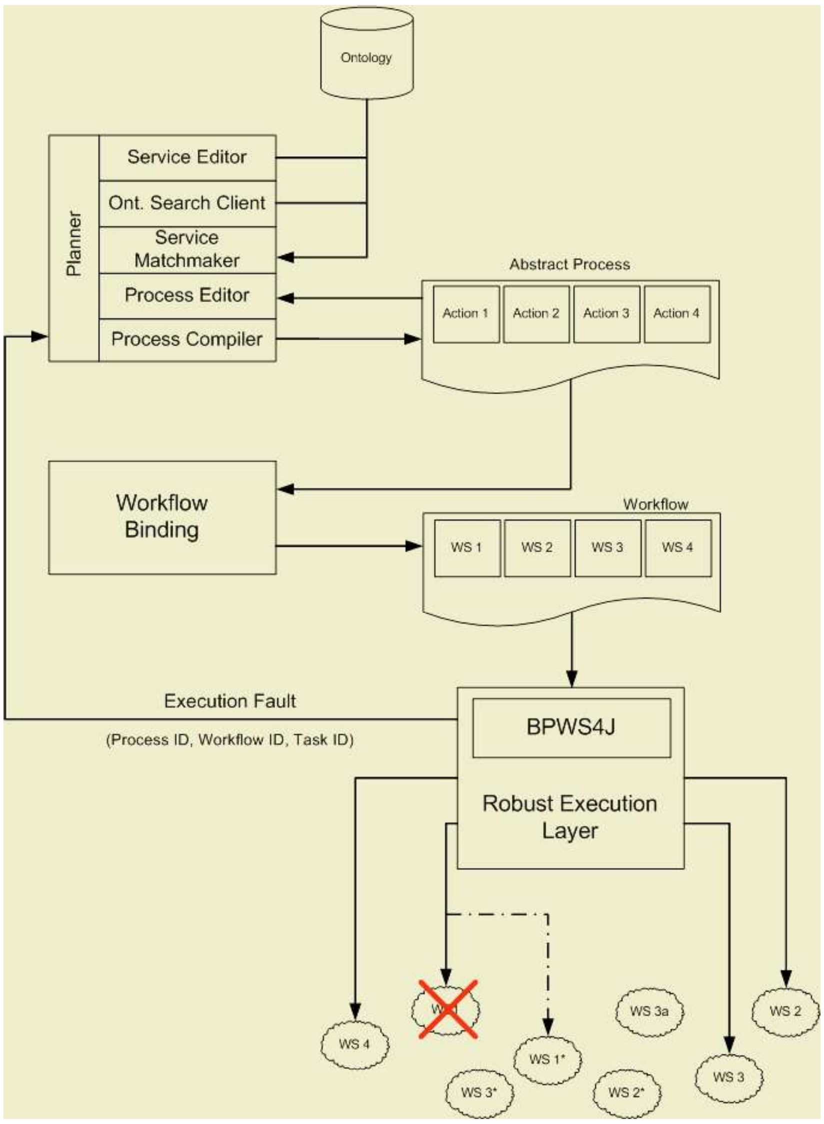

Fig. 12. Business process enactment architecture

The system is arranged in a two-tier hybrid architecture [19]. The upper tier provides a planning mechanism based on a library of process descriptions, the process repository. In the ideal case the planner starts from the platform independent description of the process descriptions, performs the planning and transforms this platform independent processes or workflows into platform specific processes and workflows, like BPEL4WS. Currently, the planner selects plans from the plan library based on a goal description and a precondition (both of which are first order formulae matched against plan metadata). The planning tier provides a number of additional tools such as a service editor, a service matchmaker used to find suitable service instances, and a process editor and compiler which are able to create abstract process descriptions to populate the process repository. When an entry in the process repository matching a pair `<goal, precondition>` has been found, a BPEL4WS workflow instance is generated by a workflow-binding module. This module instantiates variables in the abstract process by constants obtained by the matching with `<goal, precondition>`. Thus, each abstract action in the process description is instantiated by a task corresponding to a web service invocation. The corresponding workflow is then passed on to the business process execution engine. In our current implementation, we use IBMs BPWS4J business process execution engine to enact BPEL4WS business process descriptions which have been generated from UML2 representations. Existing fields of BPEL4WS process descriptions are used to transfer identifying information of processes, workflows, and tasks from the planning to the execution layer and back.

The business process execution engine now executes the workflow step-by-step. However, instead of directly calling individual web services, the calls from the execution engine go to the lower tier, the so-called Robust Execution Layer (REL), which acts as a proxy for the execution engine. The intention of the REL is to provide a higher level of execution reliability. This is achieved by means of an underlying peer-to-peer network maintained by the REL. In our implementation, we use the Resource Management Framework (RMF) developed at Siemens [7]. Individual web services are registered as resources in the peer-to-peer system. Services can be replicated using the RMF's replication mechanism. If a selected service fails, the Robust Execution Layer tries to transparently re-route the request to a replicated service and carries out compensation activities if necessary to prevent or resolve inconsistent transactional states. E.g., if the task represented by web service WS1 fails (see Figure 12), the RMF can transparently re-route the request to the replicated web service WS1*.

A simple rollback concept is provided to compensate in case of failure. For this, the BPEL4WS task descriptions contain links to compensation actions. If a task fails (e.g., WS3) and the REF is not able to find an alternative (replicated) service, the REL will call an appropriate fault handler in the execution engine. Then, the execution engine tries to compensate the previously executed tasks, starting with the most recent ones. If a task does not have a compensation routine, or compensation fails, the rollback stops and the process execution engine reports an execution fault to the planner (see Figure 12). For instance, assume that WS3 and WS1 can be compensated, but WS2 cannot. In this case, a failure would be reported indicating the identifier of the abstract process, of the workflow instance and of the failed task (in this case WS3).

The planner will then try and elaborate alternatives either by trying to locally amend the plan (e.g., suggesting an alternative task WS3a), or by looking up an alternative process in the business process repository, taking the effects of the performed tasks that could not be compensated into account. The amended process description is again instantiated and the corresponding BPEL4WS workflow is sent to the Robust Execution Layer.

6 Conclusions and Outlook

The foremost goal of this paper was to point out a relationship between the programming of multiagent systems, which is the key topic of the volume at hand, and the design of agent-enabled business processes. Our key thesis here is that designing business processes based on an agent-enabled business processes modeling and enactment framework should be regarded as a very practicable way of programming specific types of agents, i.e., agents that assist humans and organizations in monitoring, managing, and optimizing business processes.

The technical contributions of this paper were the (i) definition of a conceptual architecture of agent-enabled business process management that cleanly separates between agent capabilities, business process modeling, and the modeling of services that are employed to actually implement processes; (ii) a sketch of implementing the conceptual architecture based on using the Model-Driven Architecture (MDA) paradigm at all three layers of the agent-enabled business process architecture, in order to allow system designers to focus on the functionality and behavior of a distributed application or system, independent of the technology or technologies in which it will be implemented; and (iii) to outline an instance of a platform independent model based on the new UML2 standard, extending the scope of the Unified Modeling Language to the design of processes, and sketch a mapping to a platform dependent model based on the Business Process Execution Language for Web Services (BPEL4WS). We discuss design-time and run-time aspects of a methodology for agent-enabled business process modeling.

As noted in the introduction, the current paper deliberately leaves a number of questions and technical issues open. Firstly, this includes most of the agent layer of the conceptual architecture. Currently we have shown how semantic process and service information can be modeled in a technology-neutral way using industrial modeling standards. Obviously, it will also be necessary to model the agents that actually make use of this information. We believe that current agent design methodologies can be used at this point; however, certain extensions may be required to make these methodologies cope with the notion of business processes and to leverage the concept of a Model-Driven Architecture to the agent layer. Achieving this requires further research.

A second open issue is the extension of our model to enable code generation. In order to be able to generate executable code from platform-specific models, transformation rules will need to be designed and additional platform-specific information needs to be encoded. We believe that these transformation rules can be defined directly on the UML meta model. However, the verification of this hypothesis is open.

In summary, there is still some way to go to achieve the vision of business-process aware agents that can understand semantically enhanced business process definitions, that can help in the design of business processes, that can select appropriate business processes for execution, monitor distributed business process execution, recognize and fix problems in a collaborative manner. We hope that with this paper, we succeeded in setting a starting point and defining an overall approach that will help researchers and practitioners to ultimately build such technology based on existing standards for service-oriented computing, business process management, and software architecture.

References

1. Armstrong, Ch. (2002) ‘Modelling Web Services with UML’, Talk given at the OMG Web Services Workshop 2002.
2. Barbuceanu, M., Fox M.S. (1997) ‘Coordinating Multiple Agents in the Supply Chain’. Proceedings of WET-ICE 97, Boston, pp. 134-141.
3. B. Bauer, J. P. Müller, J. Odell: Agent UML: A Formalism for Specifying Multiagent Software Systems. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) 11(3): 207-230, 2001.
4. BPMI (2003), 10 June, http://www.bpmi.org/
5. ebXML (2003) ‘ebXML Business Process Specification Schema’, June 2003, http://www.ebxml.org/specs/ebBPSS.pdf
6. FIPA (2003), FIPA specifications, http://www.fipa.org/specs/fipa00030/
7. Friese T., Freisleben B., Rusitschka S., Southall A. A Framework for Resource Management in Peer-to-Peer Networks. In Proceedings of NetObjectDays 2002, Volume 2591 of Lecture Notes in Computer Science, pp. 4—21, Springer-Verlag.
8. Fuchs, I. (2002) ‘Web Services and Business Process Management Platforms – Understanding Their Relationship and Defining an Implementation Approach’, http://www.ebpml.org/ihf.doc
9. Giunchiglia F., Mylopoulos J, and Perini A. The Tropos Software Development Methodology: Processes, Models, and Diagrams. In Agent-oriented Software Engineering III. Lecture Notes in Computer Science, volume 2585, pp. 162—173. Springer-Verlag, 2003.
10. Guo Y., J. P. Müller, C. Weinhardt. Learning User Preferences for Multi-attribute Negotiation: An Evolutionary Approach. In Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems, volume 2691 of Lecture Notes in Artificial Intelligence, pages 303-310, Prague, Czech Republic. Springer-Verlag, 2003.
11. Huget M.-P. "An Application of Agent UML to Supply Chain Management". In Proceedings of Agent Oriented Information System (AOIS-02), Paolo Giorgini and Yves Lespérance and Gerd Wagner and Eric Yu (eds.), Bologna, Italie, July 2002. (Short presentation of the technical report ULCS-02-015)
12. Huhns, M.N. (2002) ‘Agents as Web Services’, IEEE Internet Computing, July/August 2002, pp. 93-95.
13. IBM (2003) BPEL4WS: Business Process Execution Language for Web Services, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
14. IBM Business Process Execution Language for Web Services Java Run Time. http://www.alphaworks.ibm.com/tech/bpws4j
15. Jennings N.R., P. Faratin, T. J. Norman, P. O'Brien and B. Odgers (2000) "Autonomous Agents for Business Process Management" Int. Journal of Applied Artificial Intelligence 14 (2) 145-189.
16. Jennings N.R., K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems, 1(1):7--38, 1998.
17. Model Driven Architecture homepage. The Object Management Group (OMG). http://www.omg.org/mda/

18. Müller, J.P., Bauer, B. (2002) ‘Agent-oriented Software Technologies: Flaws and Remedies’, *Proceedings of Workshop on Agent Oriented Software Engineering (AOSE 2002)*, Bologna, pp. 210-227.
19. Müller, J.P. The Design of Autonomous Agents – A Layered Approach, volume 1177 of Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.
20. Radjou, N., Orlov, L. M., und Nakashima, T.: Adaptive Agents Boost Supply Network Flexibility. 2002. March 2002 Tech Strategy Brief. Forrester Research.
21. Thöne, S., Depke R., Engels G. (2002) ‘Process-Oriented, Flexible Composition of Web Services with UML’, Proc. of Joint Workshop on Conceptual Modeling Approaches for e-Business (eCOMO 2002); Tampere; (to appear)
22. UML Homepage. The Object Management Group. http://www.omg.org/uml/
23. W3C Web Services glossary. http://www.w3.org/TR/ws-gloss/
24. Wooldridge M., Jennings J.R., and Kinny D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multiagent Systems, volume 3, number 3, 2000, pp. 285—312.
25. Kleppe M., Warmer J., Bast W. MDA Explained – The Model Driven Architecture: Practice and Promise, Addison Wesley, 2003