Multi-concerns Composition for a Process Support Framework

Gabriel Pedraza, Idrissa Dieng, Jacky Estublier

LIG, 220 rue de la Chimie, BP53
38041 Grenoble Cedex 9, France
{Gabriel.Pedraza-Ferreira, Idrissa.Dieng, Jacky}@imag.fr

Abstract. A process is a set of activities performed to reach a goal. Software Engineering, in its search to provide more complex and scalable software systems, tries to use processes as an integration technique. An orchestration is a process model which expresses the component call order, that is, the overall application logic. It is not sufficient, for an orchestration to define and execute the control, it should also define how the different concerns that make up an application will be composed and managed. This paper presents how we used model composition techniques to develop a multi-concerns workflow and an orchestration system. The approach allows defining and managing how different concerns have to collaborate in order to really orchestrate a complex application and to improve the reuse of heterogeneous components and services. More generally, we show how extensible control formalism and model composition, together, provide a general way to build comprehensive support for a large range of process related areas.

Keywords: Process, Software Engineering, Software Environments, Separation of Concerns, Model Composition and Reuse.

1. Introduction

Software Engineering is permanently looking for higher level abstraction tools and techniques to specify and build systems. Among the recent approaches one can find Model Driven Engineering (MDE) and process support. MDE [11] proposes to use models (and their associated meta-models) as the first class artifacts to create (software) systems. Process modeling and support is an ideal use case for the MDE approach because models are the “natural” way to specify processes [7][10]. Processes are used in a wide range of applications including business process, software process, workflows, and service orchestration.

Software Engineering is based on few principles: separation of concerns, information hiding and reuse. Information hiding, or encapsulation, the oldest and most successful technique so far, advocates for a clear separation between the visible functionality provided by parts (originally called modules, now called classes, components or services), and their implementation, which must be totally ignored by the clients of the parts. Separation of concerns aims at dividing an application into different concerns (parts). To be reusable, parts must be defined as independently as

possible from their context of use, and a composition mechanism must been provided to form an application from those parts [16].

Separation of concerns principle, also called ***inversion of control***, is present in several software development paradigms. In most aspect-oriented programming (AOP) techniques, a "main concern" (i.e, a functional concern) is defined as objects, and other concerns as aspects. Objects ignore aspects and do not call them. Unfortunately, the concern composition mechanism is based on the principle that an aspect "knows" the implementation details of objects, which defeats the encapsulation principle.

The publish subscribe mechanism [6], also called ***implicit control***, is a low coupling composing technique in which parts send events and ignore who, if any, is interested in these events. However, the receiver knows and must declare what it is interested in.

Service-Oriented Computing [13] (SOC) is an approach that uses the *Service* concept as a basic building block. Services make visible their functionality through their interfaces (as do components), they do not depend on neither other services (as in Web Services [1][17]) nor on the service implementation to be used. A service can be dynamically selected at execution (as in OSGi [12]: ***abstract or dynamic control***). Therefore, services nicely satisfy the encapsulation and independence principles and can be combined to build higher level services or applications.

Service Orchestration, also called ***external control***, is used to build applications composed of (web) services [8][9]. An orchestration model usually expresses the control and data flow between services using processes. Current orchestration languages, like BPEL4WS [2], are robust, but rather rudimentary. They barely express the partial ordering of the web services calls. They also hypothesize that all composed parts are web services, that the data used by the different services are compatible, and that there is no need to execute code in the orchestration itself. Further, orchestration describes the control from a single point of view, the one of the "main concern". There is no way to make explicit other concerns, as a consequence, developers have to deal with different technologies (e.g., Workflow, Web Services, AOP, etc.), at different levels of abstraction, to design and implement the targeted application.

We believe that the issues mentioned previously are not specific to the orchestration area, but are central to most, if not all, Process Driven Applications. Our approach is an attempt to solve the different problems identified above. It is based on the MDE approach and satisfies the basic Software Engineering principles, i.e., Separation of Concerns, Encapsulation, and Reuse. It is a multi-concern approach in which the (process driven) application is built by the flexible composition (weaving) of a control model (the main concern) and by a number of other concerns defined by other models.

The paper is organized as follows. In section 2, workflow support is introduced as a multi-concern composition. Section 3 describes our orchestration system. For building processes in a specific domain, several kinds of process extensions are presented in section 4. Finally, section 5 concludes the paper.

2. Workflow model: Control, Data and Resources concerns

We define a workflow as the composition of three “independent” concerns:

- the control concern, which defines the ordering (partial or strict) of component execution.
- the data concern, which defines the entities the workflow is acting upon, and
- the resource concern, which defines the actors in the workflow.

Clearly, data and resources exist whether a control is formalized or not, and their definition is relevant for very many purposes and concerns in the company, therefore, it is normal that their definition be performed irrespective of a given process. Similarly, the control should be abstract and general enough to express high level processes: business processes, software processes, or automation processes, irrespective of the detail and format of data, and irrespective of the actual available resources.

For us a (basic) workflow model is the composition of three models, one for each of the three concerns: control (the main concern), data and resource. At runtime a workflow engine is the composition of an engine or interpreter for each one of these models [4].

2.1. Control Concern: the APEL formalism

A process is a set of activities performed to reach a goal (e.g., a business goal or a software release) [19]. A workflow model is the description of the order of activities and the circulation of associated objects, for process automation purpose. The (actual) workflow technology fits processes with rather a deterministic behavior, like in office automation. It can be used for supporting some simple parts of a process. APEL (for Abstract Process Engine Language) [3] is an activity-based process modeling language. It proposes a high level formalism to model processes and it has a flexible execution system which supports dynamic evolution of processes (their models and/or their instances).

The APEL formalism [3] contains the following concepts: Activity, Product, Port, Resource and Dataflow. An activity is a step in the workflow during which an action is performed. Products are objects (e.g., documents, data) produced, transformed or consumed by activities. Ports are the activity interfaces and they define and control the products that are expected and/or produced by activities. Ports are the only externally visible part of an activity (the encapsulation principle). Input ports perform an AND over their incoming data flows. That means that the port fires when at least one exemplar of each expected product is available in the port. Firing means that products are removed from the port and the activity is started with that product set as input. When an output port is full it fires (either automatically or manually), which means products are sent to all destination ports. Dataflows describe how products are exchanged among activities. Resources are responsible for activities execution. The APEL metamodel is presented in Fig. 1.

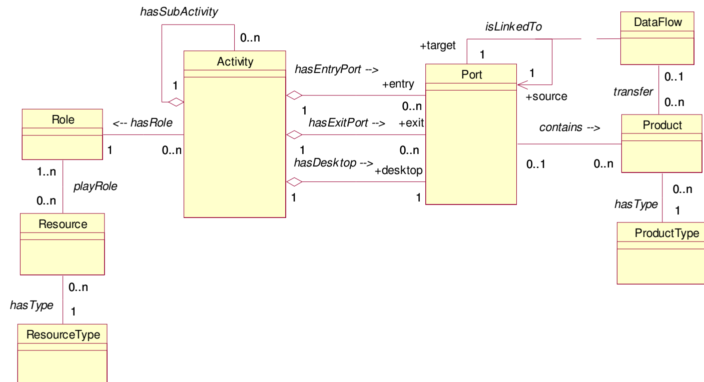

Fig. 1 APEL metamodel

A reseller process example using APEL is presented in Fig. 2. The process starts with the *Receiver Order* activity in which a customer orders a product, followed by two activities executed in parallel. In the first one, the products are shipped using the *Ship Products* activity; in the other one, an invoice is sent to customer with the *Send Invoice* activity, and a payment is awaited in *Receive Payment* activity.

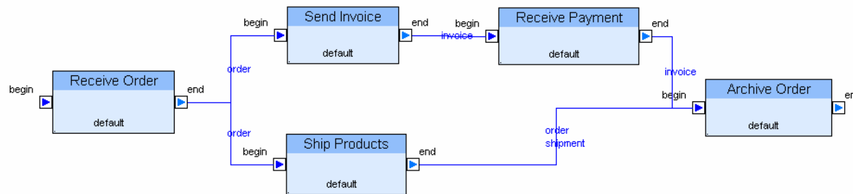

Fig. 2 An APEL control model

Finally the *Archive Order* activity archives ordering process documents and the process finishes. The real nature of the activities is not defined in the model, and they can be manual, automatic, or a combination of both. The nature, format and content of the data circulating between activities are not defined either, due to the fact that APEL products are placeholders for information circulating in the process. An APEL product can be physical, an electronic document (e.g., a file, a Software configuration), or structured data of any kind (e.g., a data base record, a Java object, an XML document).

2.2. Data Concern

Considering that data managed in APEL are simple placeholders, APEL models use variables with a symbolic name (a string), whose type is also a symbolic name. This

is sufficient to understand the workflow purpose, but not to execute it. The real data structure and content must be defined elsewhere. In our approach it is expressed in another model conform to the data metamodel, in this metamodel are present concepts like *Data Type* and *Attribute*.

2.3. Resources Concern

Resources are, in APEL, responsible for activity execution (e.g., humans, teams, tools, code, services), but nothing is specified about the resources nature and organization in an APEL model, only a resource name and a set of roles is given. A meta-model for resource definition is specified. This meta-model defines, among other things, the *Role* and *Human* concepts.

2.4. Multi-concerns Composition

The three models (concerns) presented above are defined in an independent way to improve their flexibility and their reutilization capabilities. Executing the control model alone is only a simulation, not really a workflow execution. A workflow execution makes sense when control is acting upon data and is performed by resources. Composition is performed by relating the control model with the data and resource models. Control has been selected as the main concern because the control model has entities pertaining to the other models. However, they must be consistently composed to have a usable workflow system.

To compose control and data models we define a connection between a *Product Type* in a control model, and a *Data Type* in a data model. In the same way, for the control and resource composition, a relationship between the *Role* concept in a control model and the *Role* concept in a resource model is established.

The Codele tool is used to support these compositions at design and execution time. In Fig. 3 we show a composition example between control and data models. *Product Types* used in the example (the reseller process) are on the left side, and they are being composed with *Data Types* defined in an independent data model which are in right side of the window.

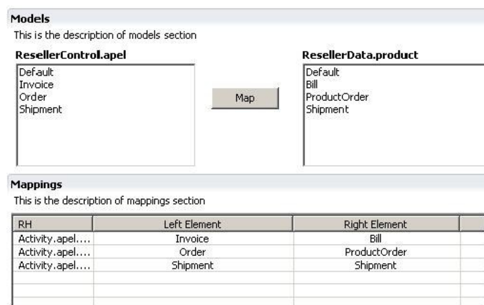

Fig. 3 Control / Data composition

2.5. An executable workflow

With the composition of control, data and resource models, a composite workflow model can be created. An execution environment permits executing these composite models in an abstract and transparent way. So far, our workflow model is made of three abstract models. Executing the workflow at that level is a dry run, since neither real data nor resources have been selected, and only their abstract definitions are available.

A complete workflow support system needs to relate the above model definitions with real data (for example a database, a CVS or subversion base, a file system), real resources (in a LDAP repository for example), and real users working on a computer using a work-list tool.

In our approach, the abstract level can be bound to executable tools to support different technological needs. The abstract layer is still rather independent since the tools can be changed without changing the abstract models. The objective of this separation is to increase the abstraction level (encapsulation) and the reutilization property. An adaptation layer is always necessary to bind models in abstract layer with tools in implementation layer. The architecture is presented in Fig. 4.

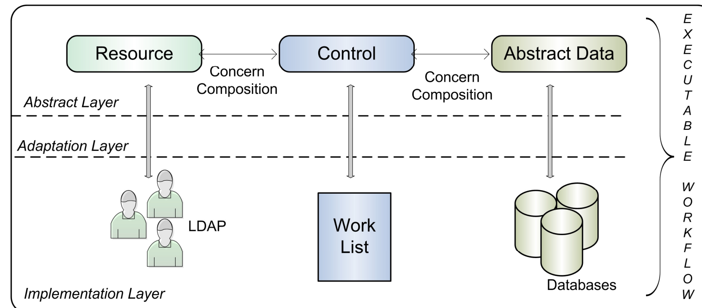

Fig. 4 Concerns Composition and Tools Binding

A complete and executable APEL support system has been developed and used for years, which supports concern definition, composition and integration with different tools.

3. Service Orchestration

Process Driven technology, under the name workflow, originally targeted office automation. More recently, process technology is applied to applications integration and (web) service orchestration. In this section, we show how we have modified the workflow support environment presented above to develop a service orchestration environment, using again, the control as the central concern.

3.1. Orchestration: composing abstract process with services

Service orchestration is an approach to build applications by composition of existing services [8][9]. Generally this composition is defined in a control model, and performed by an engine which interprets the model to determine the services execution order (control flow) and their data exchange (data flow). We use our APEL formalism for the control model, which includes the abstract data model used by the services. Resource concern is implicit because, usually, orchestration hypothesizes that Web Services are the only ones performing the task, not humans.

In a typical workflow, the data model is linked to a data base, the resource model to an LDAP directory, and control is linked to a work-list or agenda tool. In contrast, in orchestration, resources are (Web) Services, the control model is linked with services, and the data is used to define the service parameters. The mechanism which consists in establishing the relationships between activities (in the control model) and services is called *Grounding*.

3.2. Abstract Services and Flexible Binding

An abstract service is a functionality definition. In our system, an abstract service is represented as a Java interface, a set of properties (a set of attribute/value pairs), and a semantic description [14] (e.g., the WSML language [15]). An orchestration is defined based on abstract services. The concept of abstract service increases reusability and flexibility since a number of different real services, of different kinds, can implement a required abstract service.

Grounding is the mechanism by which an activity is associated with an abstract service, which permits expressing the orchestration logic in an abstract way. Given that abstract services are not executable, consequently, an abstract orchestration is not directly executable either.

We call *Binding* the mechanism which assigns a service implementation (a real functionality) to an abstract service (a functionality definition). The *Binding* step introduces flexibility because it offers the possibility of selecting service implementations, of changing them, and of adding new ones, at any time, including at execution time if required.

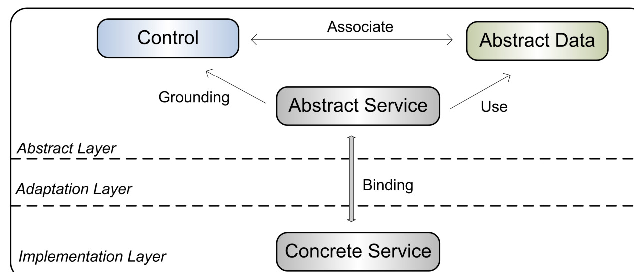

Fig. 5 Orchestration general architecture

However, if a service implementation has been defined independently from the abstract service, it is likely that its interface or technology does not directly fit the abstract service. To benefit from the full reuse potential of service implementations, it is possible to introduce mediators in the adaptation layer, allowing a service implementation to become a valid implementation of an abstract service, even in the presence of syntactic incompatibilities. The Binding tool actually supports services implemented in some of these technologies: Web Services, OSGi, EJB and Java.

3.3. FOCAS: an framework for service orchestration

We have developed a framework called FOCAS (Framework for Orchestration, Composition and Aggregation of Services), which assists orchestration designers and developers in defining, composing and executing service based applications. FOCAS is provided as an Eclipse feature, and is composed of a number of specialized graphical editors: one for each type of concern (Control, Data and Service), one for each composition relationship (Associate, Grounding and Binding) and one for the support of formalism extensibility (presented in the next section).

From a technical point of view, a complete service based application may be made up of a large number of files (e.g., models, xml, code, proxies), located in a number of IDE projects, directories, and libraries. The definition, creation and management of these technical artifacts are fairly complex and error prone tasks, if undertaken by hand. In FOCAS, the user interacts only with the above editors, and is unaware of the many associated files and directories, because FOCAS fully generates and manages these artifacts allowing non software experts to easily define and execute orchestration models. FOCAS hides the complexity associated with implementation details and coding corresponding to the different models composition.

In FOCAS when a new orchestration model is created, existing control, data and service models can be selected from their corresponding repositories and associated to the new orchestration project. The selected models and their associated files and directories are copied in the right place and transformed into the corresponding Eclipse artifacts. In practice, an orchestration model is mapped to an Eclipse project, and each individual model is contained in a number of files in a directory. A control/data mapping file is transparently generated with a default association (assuming name and type compatibility), designers only have to complete some mapping in the Codele tool. A default grounding empty file is generated too, it is after modified using the grounding tool.

A change to any of the models (i.e. control, data and services) triggers the rebuilding of the whole application. From the orchestration designer point of view, a unique orchestration compiler generates the application executable code.

4. Toward a Domain Specific Process Support Framework

Even though most process domains have common core concepts (i.e., control, data and resource concerns), each specific process domain has its own concepts and needs

[5][18]. Ideally, the process formalism should propose the concepts relevant in a given process domain. For that purpose, we have identified two kinds of extensions. One extends the basic process environment adding new concerns (other than data and resource), and the other one is concerned with the explicit definition and (re) use of recurring patterns to extend the basic control formalism.

4.1. Extensions through Multi-Concerns Composition

Our approach is based on the separation of concerns principle, in which each concern is defined through a model. Model composition is used as a powerful composition technique which weaves different concerns without modifying the original models i.e., using a non-invasive method for ensuring the reutilization property for models and implementations or support tools.

Our approach permits defining process support environments in a large spectrum of domains, through the addition and removal of concerns, maintaining control as the central concern. The nature of the concerns which can be added varies according to the process specific domain needs. Functional and non-functional concerns can be appended. For example, software processes automation requires introducing concerns related to the software artifacts, that is, a functional concern. On the other hand, service orchestration may require introducing non-functional concerns, like transactional behavior or security support, since these characteristics are of critical importance in a real life orchestration application.

For example, we have added the security concern to the core service orchestration environment. The security concern is defined as a model that annotates the control model with security characteristics, like integrity and confidentiality that are applied to the data that is being exchanged between services. The composition mechanism between control and security also uses a non-invasive composition technique, which does not modify the original models. In this composition, the security model is used to generate configurations and code that will be executed by a security middleware. Graphically, the extension adding security support in service orchestration is presented in Fig. 6.

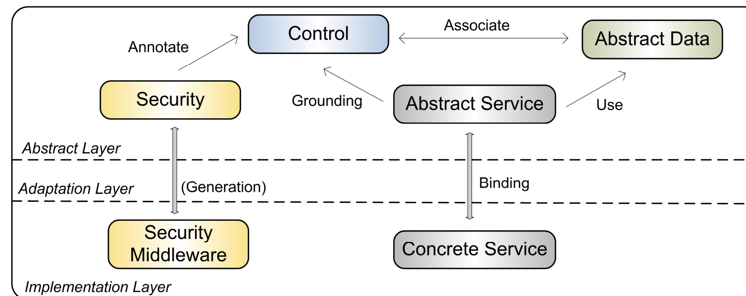

Fig. 6 Extending orchestration with the security concern.

4.2. Control Formalism Extension

4.2.1. Operators: Generic Extensions

O*perators* add new concepts in the APEL formalism. Operator syntax and semantics are statically defined and are adapted when used in a process model. This kind of extension can be reused in several application domains.

An operator is defined as a special kind of activity associated with a code template. Being generic, the actual data present in operator ports is undefined, thus one has the possibility of adding products to it. The associated code template generates code adapted to the actual data and defines the behavior. In some cases, this code must be manually completed.

Basic control operators like *If*, *While*, *RepeatUntil*, *ForEach* can be added to imitate BPEL4WS control activities. However, higher level operators can also be added, like the *SendMail* operator which allows sending an electronic mail within a process. It is only when a *SendMail* operator is added into a model, that the parameter types are known. Specific code to that operator in that model is generated, and true data types are used by developers to complete functionality (give sender, subject and message).

4.2.2. Behavioral Extension

Sometimes, one may wish to specialize the usual behavior of an activity, or to code it directly in a particular application. The extended behavior of an activity takes the form of Java code which is inserted before/after the activity execution. In some cases, this code can directly perform the associated service computation, or call any program available in the orchestration machine.

An activity that performs data transformation or shows a message when it is executed, is a behavioral extension. The developer only writes the associated code to show the message, and links this code to the activity in the process model. The FOCAS environment shows the class in the Java editor in which the *doActive* method must be completed by the user to carry out the extension.

4.2.3. Type of Activity: Domain Specific Extensions

Domain specific extensions are recurrent activities in a specific domain. These activities can be defined once and declared as a type of activity, which can then be reused in any subsequent model. These types of activity adapt the formalism to an application specific domain (abstract data and services), and can be used only in the application domain in which they have been defined.

A type of activity can be an atomic or composite activity, with or without extended behavior. The code associated to the activity type defines its behavior, and this code cannot be modified. For example, finding a flight in the flight reservation domain could be a predefined activity type. This type of activity makes it possible to carry out the search for flights fitting customer criteria, and select one flight among the search result. It can be reused in several processes in the flight reservation domain.

Being part of the abstract orchestration, the *FindFlight* activity can be bound and adapted to a large range of actual web services that perform, in one way or another, a

flight search. It increases the reusability of this kind of predefined domain specific extension.

5. Conclusion

There is a very wide potential for process technology use, including business process, software process, office automation, service orchestration, and process driven applications in general. Despite this wide range of application domains, Process technology only found its customers in narrow niches. One reason is that each domain requires a specific language and its associated engine, with specific non functional properties and constraints, and excellent technical implementations. Therefore, a quality Process Support Environment (PSE) is necessarily expensive to design and build; it is no surprise to observe that such PSE exist only in specific domains (e.g., office automation with workflows, web service applications with orchestration), and that their formalisms, engines and properties are incompatible.

It is compelling to observe that, on one hand, all these environments share very much the same principles and even the same technology, but the solutions are incompatible because different characteristics are expected, the expertise and know-how developed in one niche is not reused in the others. We believe that it is possible to address a large range of the potential of process technology, reusing common pieces, and adapting each PSE to its specific domain through formalism extension, and through a flexible composition mechanism, selecting the required functionality and characteristics, and developing only the specific ones.

We have developed extension and composition techniques, and the environment and tools supporting the approach (e.g., formalisms, editors, engines, composers, tools), which allow the easy production of quality process driven applications. Using our approach and technology, developing an environment for a new target domain requires (1) defining the control formalism fitting the target domain concepts and needs (2) identifying the required concerns (functional or not), (3) reusing those concerns already available in libraries, (4) developing those not available. Then the environment is built pretty easily, extending the core control formalism, and using our generic tools to compose the selected concerns.

We hope this new approach will open the doors for process technology to be used in a wide range of applications, and notably in process driven applications.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, H.: Web Services - Concepts, Architectures and Applications. Springer Verlag (2003)
2. Cubera, F. et al.: Web Services Business Process Execution Language. Specification available at http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf (2007)
3. Estublier, J., Dami, S., Amiour, M.: APEL: A graphical yet executable formalism for process modeling. Automated Software Engineering: An International Journal **5**(1) (1998) 61–96
4. Estublier, J., Ionita, A.D., Vega, G.: A Domain Composition Approach, Las Vegas, International Workshop on Applications of UML/MDA to Software Systems, CSREA (2005) 1–7

5. Estublier, J., Villalobos, J., Tuyet LE, A., Sanlaville, S., Vega, G.: An Approach and Framework for Extensible Process Support System. Lecture Notes in Computer Science **2786**(2003) 46–61
6. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.: The many faces of publish/subscribe. ACM Computer Surveys **35**(2) (2003) 114–131
7. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Tool support for Model-Based Engineering of Web Services Composition. Volume 1., Orlando, 2005 IEEE International Conference on Web Services (ICWS 2005), IEEE Computer Society (2005) 95–102
8. Khalaf, A., Mukhi, N.K., Weerawarana, S.: Service-Oriented Composition in BPEL4WS. Proceedings of the 12th International World Wide Web Conference, Budapest (2003) 1–10
9. Khalaf, R., Keller, A., Leymann, F.: Business processes for Web Services: Principles and applications. IBM Systems Journal **45**(2) (2006) 425–446
10. Koehler, J., Hauser, R., Sendall, S., Wahler, M.: Declarative techniques for model driven business process integration. IBM Systems Journal **44**(1) (2005) 47–65
11. OMG: Model Driven Architecture. Available at http://www.omg.org/mda/
12. OSGi Alliance: OSGi 4.0 release. Specification available at http://www.osgi.org/ (2005)
13. Papazoglou, M., van den Heuvel, W.: Service oriented architectures: approaches, technologies and research issues. The VLDB Journal **16**(3) (2007) 389–415
14. The OWL Services Coalition: OWL-S 1.1 release. Specification available at http://www.daml.org/services/owl-s/1.1/ (2004)
15. Roman, D., Keller, U., Lausen, H., Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology **1**(1) (2005) 77–106
16. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Professional. 2nd Edition, England (2002)
17. W3C: Web Services Architecture Specification. Specification available at http://www.w3.org/TR/ws-arch/ (2004)
18. Wile, D.S.: Supporting the DSL spectrum. Journal of Computing and Information Technology **9**(4) (2001) 263–287
19. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer Verlag (2007)