Requirements for Automated Service Composition*

Harald Meyer and Dominik Kuropka

Hasso-Plattner-Institute for IT-Systems-Engineering
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
{harald.meyer|dominik.kuropka}@hpi.uni-potsdam.de

Abstract. Automated service composition is an important approach to create aggregate services out of existing services. Several different approaches towards automated service composition exist. They differ not only in the used algorithms but also in provided functionality. While some support the creation of compositions with alternative or parallel control flow, others are missing this functionality. This diversity yields from a missing consensus on the required functionality to automatically compose real-world services. Hence, with this paper we aim at providing the foundation for such a consensus. We derived the required functionality from multiple business scenarios set up in the Adaptive Services Grid (ASG) project.

1 Introduction

Cooperation between enterprises on global level is essential to conduct successful business. Service composition yields the possibility to aggregate services from inside and outside of enterprises to new composed services which raises the value of the whole service chain. Compositions are currently modeled manually. This leads to inflexible service compositions with in-optimal quality as manual modeling makes adjustments for individual service requests too expensive. Automating the creation of compositions increases flexibility and quality. Furthermore it reduces the probability of enactment failures caused by services used in the model which either disappeared or have been modified over the time. Automated service composition will include newly registered services automatically while de-registered services will no longer be used in compositions.

Several approaches for automated service composition differing in provided functionality exist. While they motivate automated service composition, a consensus on the required functionality to perform automated service composition is missing. Zeng et al. [1] present an automated composition approach based on a rule inference engine. They describe the supported functionality of their system DY_{flow} like the composition of parallel and alternative control flow or the annotation of services with Quality of Service (QOS) properties. Pistore et al. [2] use

* This paper presents results of the Adaptive Services Grid (ASG) project (contract number 004617, call identifier FP6-2003-IST-2) funded by the Sixth Framework Program of the European Commission.

their model-checking planner *MBP* to do automated service composition. It supports non-deterministic services and partial observability of service effects. They explain that these features are necessary to model real-world services. Sirin et al. [3] implement automated service composition through Hierarchical Task Network (HTN) planning. For them an important feature of service composition is the hierarchical decomposition of complex task to atomic—invokable—services. The approach by Berardi et al. [4] is based on the automatic synthesis of finite state machines. In [5] they extend this approach to non-deterministic finite state machines to support real-world, external services. This small selection underlines that no common understanding on the required features exists. While rationale is provided for the extended features of each approach, the features are mostly rooting from the origins of the used algorithms. As most algorithms come from the area of automated planning, features are aimed at supporting problems in typical planning domains[1]. Features like parallel control flow that are usually not of importance in such domains are rather sparsely supported.

In this paper we aim at creating a consensus on the required features for automated service composition. We will motivate them through a real world business scenario. Following this introduction a scenario is presented that forms the basis for our requirements analysis. In Section 3 we present the functional requirements on a service composition component. Finally, Section 4 contains a summary of the presented requirements and an outlook.

2 Use Case Scenario

The *attraction booking* use case scenario presented here serves as the motivating scenario in the Adaptive Services Grid (ASG)[2] project. The goal of the ASG project is to develop a reference architecture for adaptive service matchmaking, composition, binding and enactment. Furthermore a prototypical implementation for testing and evaluation purposes of this reference architecture is provided by the project. The scenario allows customers to retrieve information about attractions in the immediate surrounding of the customer by using a mobile device. Additionally customers may request directions and maps to the attraction or they can directly book tickets with their mobile device.

Based on the customer's current position and the specified query, the system compiles a list of attractions. This list is displayed as it is or as a map showing the position of the customer and the attractions found. The customer can then request additional information for specific attractions. This includes its address, opening hours or price. The customer can also request a route description to lead him to the attraction. If the attraction is bookable, the customer can book a digital ticket. The route description consists of a map and a textual description. While the map shows streets and buildings, indicating the path as a colored line,

[1] Domains at the biannual International Planning Competition (IPC) [`http://ipc.icaps-conference.org`] include for example controlling ground traffic at an airport or the flow of oil through a pipeline network.

[2] `http://asg-project.org`

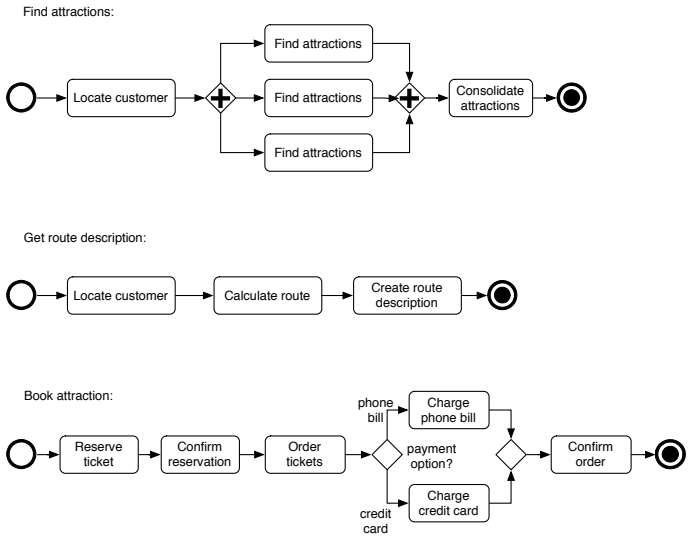

Fig. 1. Example compositions for attraction booking use cases

the textual description includes street names and directions (e.g.: where to turn left or right). To book a specific attraction or event, the customer selects the number of people to attend the event. The system then offers the customer a number of tickets for a certain price category. If the customer decides to accept the offer, he is charged the specified amount of money. In return he receives the given number of digital tickets for the event. The attraction booking booking scenario encompasses the three use cases find attractions, get route description, book attraction.

All three are fulfilled by enactment of proper service compositions. Figure 1 shows some example compositions for each use case. The activities inside the compositions represent service invocations. The shown compositions are tailored exactly to the specific user request. For example, if the customer's location is already specified in the service request, the invocation of a localization service is not necessary. A similar flexibility applies to the booking of an attraction. No payment option is specified in the shown example. Instead the user selects a payment option just before payment. But the customer may have already specified his preferred payment option in a profile. The client application on the mobile device then transfers the information about the user preference and the customer does not have to select a payment option. If service providers register new attraction information services or remove old services, future queries for attraction information will incorporate these changes of the service landscape. In this scenario automated service composition allows changes in the set of available services and in the client application (new use cases, additional information) without requiring manual adjustment of the service compositions.

3 Functional Requirements of Service Composition

In the previous section a use case scenario from the ASG project was introduced. In the ASG project, the industry partners developed and evaluate five different scenarios for their business areas. The attraction booking scenario together with another scenario proved to scenarios with the biggest market potential. Our approach was to derive the requirements from these two scenarios. We will use the attraction booking scenario in this section to motivate the requirements on a service composition component and a composition language. The composition language defines how service compositions look like. The requirements are ordered according to the categories elements of composition, control flow, data flow, data model, and quality of service.

This categorization is based on the process representation perspectives by Curtis, Kellner, and Over [6]. While other approaches [7,8,9] propose additional aspects or categories, our first four categories represent the core of all these approaches. We see quality of service and the optimization for it as one of the main advantages of automated service composition. If we did not introduce an own category for it QOS requirements would be scattered among the elements of composition (QOS properties of services) and the control flow (QOS fulfillment of service compositions).

3.1 Requirements on the Elements of Composition

The elements of composition are the building blocks from which service compositions are created.

Req. 1: Elements of composition are services interactions The elements of a composition are activities that perform a task. All these activities must be service interactions. This does not limit generality as we can encapsulate all other activities—even manual—in services. For example: In the above scenario the activity to locate a customer is realized by a service. Each service performs exactly one task and is stateless. It is therefore not necessary to support more than one operation per service. Besides invoking services, a composition can itself be invoked as a service.

Req. 2: Elements of composition are service compositions Service interactions are the atomic elements of compositions. To improve reusability existing service compositions are used as elements of compositions as well. Service compositions can be either manually modeled or they can be results from previous composition requests. Using manually modeled compositions allows to express process parts that cannot be created by the composer automatically (e.g. loops are impossible with most composition approaches). Reusing previous automated service compositions is useful in most scenarios especially if equal functionalities are often requested.

Req. 3: Services have input and output parameters Services perform a specified functionality and this functionality usually depend on data provided in the service request as input parameters. For example the localization of a customer needs the customer's telephone number as input data. Services usually also return a result (e.g.: the customer's location). To allow the input and output of data, parameters are needed. A service has zero or more input and output parameters. In reality, mapping between the input and output parameters of different services is a complicated task [10]. Here we assume that input and output parameters correspond to concepts from one ontology.

Req. 4: Service functionality is described semantically Service functionality is described semantically to allow automated service composition. Besides input and output parameters, specifications of services include preconditions and effects. The precondition of a service specifies the assumptions that must hold in order to invoke the service. The effect defines the changes to the current state that result from invoking the service on the world beyond the output parameter. Preconditions and effects describe the state of the world and the state of available information. To do so, logical relations between input parameters, between output parameters and between input and output parameters can be defined. Additional variables that are not parameters of a service might be necessary to describe functionality for instance by referring to objects of the world which are not directly inputs or outputs of a service. In order to express preconditions and effects a logical language is necessary. In the semantic community well known and often used languages are Description Logics [11] and Frame Logics [12].

Req. 5: Services can have more than one precondition or effect It is common that a service has more than one precondition or effect. This can be implemented by most logical languages though conjunction of several logical expressions.

Req. 6: Services can be invokable in different situations Often a service is not just invokable in more than one situation. Confirming the order in the *book attraction* composition from Figure 1 can be invoked if the tickets have been paid by credit card or through the telephone bill. This can be achieved by having two distinct order confirmation services—one for credit card payment and one for telephone bill payment—or by supporting disjunction in the precondition. All possible situations in which the service is invokable are linked via disjunction and if just one of them is true, the service can be invoked.

Req. 7: Services can have uncertain effects Services can have more than one possible outcome and it might be impossible to determine the concrete output in advance. Uncertain effects can be expressed through disjunction possible effect. Disjunctive effects increase the complexity of automated service composition [13,14] but they are usually unavoidable when real world services have to be modeled.

3.2 Control Flow Requirements

The control flow of a process or service compositions defines the order in which the elements of composition are enacted. This includes simple sequential ordering but also complex parallel or alternative control flows. Figure 1 shows sequential, parallel, and alternative control flows.

Requirements regarding control flow can be separated into two types of requirements: Requirements regarding composition language and requirements regarding automated composition functionality. The first one describe what can kind of control flow can be modeled and how it can be expressed. The second one describe which subset of these composition language features can be automatically composed. With workflow patterns [15] a categorization for different control flow constructs exists for the workflow management domain. Requirements analysis regarding control flow will be performed according to these patterns.

Req. 8: Composition of sequential control flow In a *Sequence* of activities the activities are enacted one after another in a well-defined order. Figure 1 shows a sequence in the *get route description* composition: First the customer is located, then a route is calculated, and finally a description for the route is generated.

Req. 9: Composition of parallel control flow Parallel control flow allows the parallel invocation of activities. Figure 1 in the *find attractions* composition shows an example for parallel control flow. The different attraction information services are invoked in parallel. In general every usage of parallel control flows can be sequentialized. But sequentialization can dramatically reduce process performance. For the application to real world scenarios it is therefore mandatory. Parallel control flow is implemented by two different patterns: *Parallel Split* and *Synchronization*. A parallel split splits a single thread of control into multiple threads. A synchronization merges them later.

Req. 10: Composition of alternative control flow Alternative control flows are parts in a process where—depending on some condition—one out of many possible control flows is selected. One example is displayed in Figure 1 in the *book attraction* composition. It shows the realization of an alternative control flow using the workflow patterns *Exclusive Choice* and a *Synchronizing Merge*.

Req. 11: Composition Language supports workflow patterns So far all control flow requirements were requirements regarding the functionality of the composition component. However the composition output, an instance of the composition language, has also to be considered. The basic requirement on the composition language regarding control flow is modeling support for the above-mentioned required workflow pattern. This can be achieved through a graph-structured or a block-structured approach. In a graph-structured approach activities are vertices that are connected through edges symbolizing ordering constraints. In a block-structured approach structured activities exist that contain other activities and determine their enactment order. While a graph-structured approach is more

generic, a block-structured approach is easier to visualize and to reason about. In general it is best to support both approaches like WS-BPEL [16] does. The composition language should also support the patterns that cannot be composed automatically.

Req. 12: Composition is block-structured Supporting structured activities in the composition language does not mean that the composer outputs a block-structured result. Actually, service composition generates a partially-ordered set of services that can be represented as a graph. As already mentioned block-structuring has its advantages and should be preferred when possible. To have block-structured compositions, either the composition algorithm could support them directly or post-processing could be performed. The first approach is a new research area, so it is unclear whether it can be successful. Hierarchical-Task-Network planners [17,18,19] generate block-structured plans. But they do not actually generate block structures, but rather reuse them. Post-processing rises the problem of complexity. As shown in [20] reordering compositions subsequently can be as complex as composition itself.

3.3 Data Flow Requirements

The data flow of composition defines how data is exchanged between services. Services have input and output parameters. The output of one service can be the input for another service. Data flow requirements include for example the ability to exchange data and the usage of process input data. The following four requirements regarding data flow are all defined over activities instead of services. The process enactment engine stores the outputs of invoked services and sends them to services that require them. Hence, data is exchanged between the activities of the composition.

Req. 13: Activities exchange data The fundamental data flow requirement is the ability to exchange data. Exchanging data between activities and therefore data flow is supported in nearly all process meta-models [6,21]. Activities have formal parameters that are replaced by actual data when invoked. This data can either be process input data or output from other activities.

In workflow management, two different approaches to model data flow are in use. With the first approach all data is stored on the process level. Input parameters of activities are read from this central storage. Output parameters are stored in this central storage or the so called *blackboard*. With the second approach, data actually flows between activities. Explicit data flow connectors connect the outputs of one activity with the input of another one. So the main difference is that in the first approach all data exchange must be done through the central storage. If the output of one activity is used by two other activities, it is still written only once to the storage and read twice. In the second approach two distinct data connectors exist. WS-BPEL uses the blackboard approach through process variables [16]. In contrast, Leymann and Roller [21] propose a meta-model, used in IBM MQSeries Workflow, that facilitates explicit data

connectors. The flexibility gained by the blackboard approach, stands in contrast to the fact that it is harder to follow the implicit data flow. This requirement and service composition in general are agnostic to the selected approach of data flow representation.

Req. 14: Activities use process input data Certain data elements in the attraction booking scenario like *PhoneNumber* and *Attraction*, are not produced by any activity. These data elements are part of the process data and are inputs for the process. Processes must have such data, and activities must be able to use them.

Req. 15: Data exchange implies control flow Control flow embeds an ordering constraint between two activities if one activity depends on another one. Dependencies are for example causal links (e.g.: an activities creates the precondition of another one) or the protection of causal links. Causal links do not only exist on the level of semantic service descriptions, but also for input and output parameters. If one activity uses the output of another activity as an input a causal link between the two activities exist. Therefore an ordering constraint between the two activities must be included.

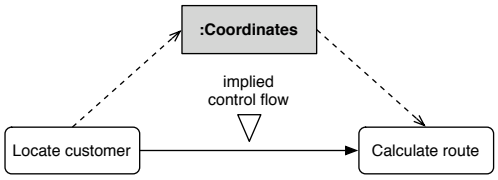

Fig. 2. Control flow implied by data flow

Refer to Figure 2 for an example: Even if the customer localization service and the route calculation service are not linked through preconditions and effects, it is necessary to add an ordering constraint between them as the route calculation service uses the coordinates created by the localization service.

Req. 16: Activities create new variables While this requirement sounds trivial and self-evident, it actually is not for automated planning / composition. Activities create new variables, means that activities do not just write data into already defined variables, but they create variables on the fly. With automated planning this is normally not possible. All the variables that are usable during composition must be defined in advance. This includes also intermediate variables that are neither used in the input nor in the output. When retrieving a route description in the Attraction Booking scenario, a coordinates variable must be defined. This coordinates variable is never used in the input or the output of the request. It is also not obvious why such a variable could be necessary. Hence, by adding this variable we are encoding assumptions about automatically created service

composition into the service request. This is bad as it hampers flexibility. Other service compositions are possible that do not need this variable.

Defining all necessary variables requires a lot of information about the available services and at least a rough idea on how the composition could look like. Therefore it is required here that activities can create new variables and that the service composer takes them into account.

3.4 Data Model Requirements

The data model defines how data elements are described. The data model is of importance for the service composition component, as it has to use data elements to replace the formal parameters with actual parameters. We assume that all services use one data model (ontology). Without this assumption complex problems in structural and semantic heterogeneities [22,10] arise that often cannot be solved automatically.

Req. 17: Data elements are typed Data elements are exchanged between services as parameters. To ensure that only valid data elements are passed as parameters it makes sense to type them. Besides predefined types, user-defined types are necessary as well. Type-safety not only prevents service enactment from invoking services with wrong parameters, it also eases planning. With typed data elements and parameters the search space of the service composer is reduced. So the composer can already eliminate many of the potential compositions.

Req. 18: Data element types are defined in an ontology Service specifications are annotated semantically to allow automated service composition. Service specifications therefore include preconditions and effects. Both are modeled as logical expressions. To use input parameters, output parameters and variables in these logical expressions, the types of data elements are described in an ontology.

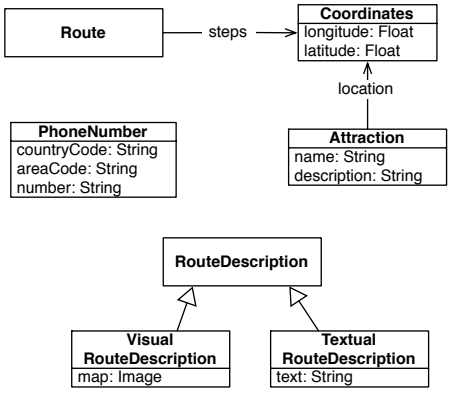

Fig. 3. Extract from the Attraction Booking ontology

An ontology is a model of linguistic means of expression on which several actors have agreed on and which are (or can be) used by those actors. [23] In this context a model is a representation of an (not necessary tangible) object system written by the mean of a formal (but not necessary textual) modeling language. So an ontology defines concepts and their relations which are important in a specific domain. Figure 3 shows the ontology of the Attraction Booking scenario modeled using the Unified Modeling Language.

Req. 19: Composer is aware of data element structure Besides having data elements with ontology-based types it is also necessary that the composer is aware of the internal structure of data elements described using an ontology. To do so the composer has to understand a logic language that can represent such structured information. Frame Logic [12] is an enhancement of first-order logic as it adds object-oriented concepts like object identity, inheritance and complex objects.

Req. 20: Data elements can be used to evaluate control flow conditions Requirement 9 stated that the composer can compose alternative control flows. An alternative control flow can be the result of an *Exclusive Choice* or a *Multiple Choice*. Both have one incoming and multiple outgoing control flows. To decide which control flows are actually executed, conditions are assigned to the individual flows. In our example, booked tickets for an attraction can be paid by via telephone bill or credit card. Based on the user provided payment option, the correct service is invoked. Conditions are expressed over data elements. So the composer must be able to create such conditions to enable a proper execution of the composition at run-time.

3.5 Quality Of Service Requirements

The quality of service (QOS) requirements here are part of the functional requirements. They describe functionality that is needed to ensure quality of service properties for service compositions defined in a service request. The requirements on the representation of quality of service properties for service requests and service specifications is out of scope of this requirements analysis. Together with a solution strategy for calculating QoS of compositions a detailed analysis can be found in [24] and [25].

Req. 21: Composer uses QOS properties In order to fulfill QOS requirements stated in a service request the composer must use the QOS properties specified by services. Based on the values of the properties for different services the composer is able to select the best service.

Req. 22: Compositions fulfill QOS requirements from service request To use the quality of service properties to select the best service compositions, it is important to describe the relevant quality of service properties and their desired

values in the service request. A service request in the Attraction Booking scenario could state that finding attractions costs at most 0.10 €. Accordingly, all services invoked can only cost 0.10 €. Then the service composer selects and composes service compositions according to these quality of service parameters.

4 Conclusion

In this paper we presented the core requirements towards automated service composition. We identified 22 requirements regarding the elements of composition, the control flow, the data flow, the data model, and the usage of quality of service properties. They described expected functionality of a component for automated service composition and requirements regarding the languages to specify services and service compositions. These requirements can be used as a starting point to develop a new component for the automated service composition, to evaluate existing ones, and to justify their extension.

Our future work in the ASG project is to realize a component for the automated service composition that fulfills all identified requirements. We will also verify these requirements in another use case prototype that is currently implemented.

References

1. Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D., Chang, H.: Flexible Composition of Enterprise Web Services. Electronic Markets – Web Services **13** (2003) 141–152
2. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Workshop on Planning and Scheduling for Web and Grid Services (held in conjunction with The 14th International Conference on Automated Planning and Scheduling. (2004) 70 – 71
3. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: Htn planning for web service composition using shop2. Journal of Web Semantics **1** (2004) 377–396
4. Berardi, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: Proceedings of the First International Conference on Service-Oriented Computing. Volume 2910 of Lecture Notes In Computer Science., Heidelberg (2003) 43–58
5. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of services with nondeterministic observable behaviour. In: Proceedings of the Third International Conference on Service-Oriented Computing. Volume 3826 of Lecture Notes In Computer Science., Heidelberg (2005) 520–526
6. Curtis, B., Keller, M.I., Over, J.: Process modeling. Communications of the ACM **35** (1992) 75 – 90
7. Jablonski, S., Böhm, M., Schulze, W., eds.: Workflow Management – Entwicklung von Anwendungen und Systemen. dpunkt Verlag (1997)
8. Weske, M., Vossen, G.: Workflow Languages. International Handbooks on Information Systems. In: Handbook on Architectures of Information Systems. Springer (1998) 359 – 379

9. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services – Concepts, Architectures and Applications. Data-Centric Systems and Applications. Springer (2004)
10. Nagarajan, M., Verma, K., Sheth, A.P., Miller, J.A., Lathem, J.: Semantic interoperability of web services – challenges abd experiences. In: Proceedings of the 4th IEEE Intl. Conference on Web Services. (2006) (to appear).
11. Baader, F., Calvanese, D., McGuiness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, Cambridge, UK (2003)
12. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the Association for Computing Machinery **42** (1995) 741 – 843
13. Erol, K., Nau, D.S., Subrahamnian, V.: Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. Technical Report CS-TR-2797, UMIACS-TR-91-154, SRC-TR-91-96, University Of Maryland (1991)
14. Ghallab, M., Lau, D., Traverso, P.: Automated Planning - theory and practice. Morgan Kaufmann (2004)
15. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14** (2003) 5 – 51
16. Organization for the Advancement of Structured Information Standards (OASIS): Web Services Business Process Execution Language (WS-BPEL). (2004) `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.
17. Sacerdoti, E.: The nonlinear structure of plans. In: Proceedings of the International Joint Conference on Artificial Intelligence. (1975) 206 – 214
18. Erol, K., Handler, J., Nau, D.S.: Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, UMIACS-TR-94-31, ISR-TR-95-9, University Of Maryland (1994)
19. Nau, D., Au, T.C., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: Shop2: An htn planning system. Journal on Artificial Intelligence Research **20** (2003) 379 – 404
20. Bäckström, C.: Computational aspects of reordering plans. Journal Of Artificial Intelligence **9** (1998) 99 – 137
21. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall (2000)
22. Sheth, A.P.: Changing focus on interoperability in information systems: From system, syntax, structure to semantics. In: Interoperating Geographic Information Systems, Kluwer Academic Publishers (1998) 5–30
23. Kuropka, D.: Modelle zur Repräsentation natürlichsprachlicher Dokumente – Information-Filtering und -Retrieval mit relationalen Datenbanken. Logos Verlag, Berlin (2004)
24. Cardoso, J., Sheth, A.P., Miller, J.: Workflow quality of service. In: Proceedings of the International Conference on Enterprise Integration and Modeling Technology, Deventer, The Netherlands, The Netherlands, Kluwer, B.V. (2002) 303–311
25. Cardoso, J., Sheth, A.: Semantic e-workflow composition. J. Intell. Inf. Syst. **21** (2003) 191–225