Dynamic Adaptation of Workflow Based Service Compositions

Heiko Pfeffer, David Linner, and Stephan Steglich

Technische Universität Berlin
Franklinstr. 28/29, 10587 Berlin, Germany
{heiko.pfeffer,david.linner,stephan.steglich}@tu-berlin.de
http://www.tu-berlin.de

Abstract. Service composition mechanisms successfully enable 'programming in the large' within business oriented computing systems. Here, the composability of single software components provides dynamicity and flexibility in the design of large-scale applications. However, this dynamicity is restricted to the late binding of services to service interface descriptions; the workflow, i.e. the execution order of the single services, remains static.

Within this paper, we present the modification of abstract service composition plans, extending service compositions' dynamicity from late binding of service implementations to a dynamic reconfiguration of the service composition structure itself. As a proof of concept, we demonstrate the adaptability of the service composition plan by applying genetic operators on the workflow graph of the service composition. Finally, an evaluation mechanism is presented to estimate the degree of similarity between the resulting composition plan and present ones.

1 Introduction

Service composition, i.e. the loosely coupled combination of single services to a large-scale application, constitutes a key pillar of the success of Service Oriented Architectures (SOA). By enabling reusability of software components, flexibility in application configuration and collaboration among different parties to achieve higher-level goals, the principle of service composition is the main enabler to meet today's SOA requirements. Here, the dynamicity provided by service composition relies on the composition of interface descriptions instead of service implementations, rendering possible a late binding of the actual services. Semantic extensions of those service descriptions provide means to address services based on their functionality and non-functional properties, abstracting from the concrete service interface during service discovery.

However, today's approaches to increase dynamicity in service composition procedures focus on the dynamic aggregation of services only; the service composition plan itself,i.e. the specification of the execution order of and interaction among single single services remains static.

Within this paper, we built on a service composition model constituting of two control graphs specifying the service compositions workflow and data flow between single services, respectively [1]. We show that this model builds an appropriate basis for adapting service compositions in their structural representation. Therefore, we exemplarily outline the transformation of service composition plans through the application of genetic operators and discuss their subsequent evaluation.

2 State of the Art and Related Work

SOA and WebServices together with their various models for describing large-scale business processes are the success story of modern service composition approaches [2]. The most prominent language is the Business Process Modeling Language (BPEL), originating from the conflation of Microsoft's block-oriented XLANG [3] and IBM's graph-based WebService Flow Language (WSFL) [4]. BPEL has become a OASIS WebService standard in 200t, referred to as WS-BPEL version 2.0, formerly BPEL4WS [5,6]. However, there are various other composition languages, feraturing advantages and disadvantages within different use cases; van der Aalst et al. thus assent that there isn't yet an holistically accepted standard for WebService Composition [7]. For instance, ebXML containing the Business Process Specification Schema (BPSS) [8] represents an alternative composition language that is rarely used within Europe and the US, but is common in Asia.

Multiple semantic descriptions such as DAML-s [9] and OWL-S [10] have been proposed within the scope of Semantic Web activities, enabling the dynamic discovery of WebServices and automatic generation of compositions. Especially for mobile devices, more lightweight descriptions have been proposed in order to reduce the computation complexity entailed by semantic reasoning processes [11]. However, the automatic creation of appropriate service descriptions for automatically generated service compositions is still challenging. Within this paper, we thus aim at modifying existing service compositions in order to produce equivalent or similar service compositions with regard to their functionality, which may however differ in their non-functional properties. Those modifications of service compositions are exemplarily performed through the application of genetic operators. Canfora et al. [12] have used a similar approach for the QoS-aware creation of service compositions. However, they did not focus on the problem of related semantic service descriptions.

3 Modeling Service Compositions

Within this section, we introduce service composition model based on two control graphs connecting semantically annotated service blueprints. Therefore, we introduce a service model featuring semantically extended I/O descriptions in section 3.1 and discuss their composition in section 3.2. The formal definition of the according control graphs is finally outlined in section 3.3, summarizing the main definitions presented in [1].

3.1 Semantically Enhanced Service Model

Services themselves are considered as atomically executable parts of application logic, whereby the execution of the service is independent from outer computations and data structures. Instead of relying on IOPE descriptions characterizing a service functionality by its inputs, outputs, preconditions and effects as discussed by Jaeger et al. [13], the presented service model relies on extended I/O descriptions. The I/O descriptions are initially restricted to primitive data types to ease the transformation of service compositions. Inputs and outputs of a service are distinguishable by a unique identifier referred to as a *port*.

As an extension of the classic I/O description patterns, services may generate a special type of semantically described outputs, referred to as *gains*, defining what a user/requester 'gains' by executing the service. Those gains are assumed to possess an importance beyond simple I/O passage within the service composition and thereby considerably constitute the services functionality; they are globally managed within a *gain space*, where they are accessible for all services within a service composition.

For instance, a Restaurant Finder service may get a String as input, specifying the current location of the user by an address within a city. Assume the output of the service is a Boolean, indicating whether a restaurant was found in, e.g., a radius of 2 kilometers. In case a restaurant was found, a map is pushed to the requesting user indicating the location of the restaurant. Within the presented service model, such a map is specified as a semantically described gain. Notably, this gain is a key feature of the service, specifying its functionality more profoundly than the simple I/O pattern.

Figure 1 shows an exemplary service designed with regard to the introduced service model.

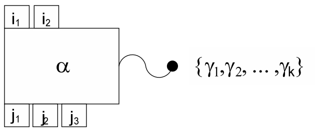

Fig. 1. Service Model

It requires two inputs i_1, i_2 for executing the atomic action α, entailing the generation of three outputs j_1, j_2, j_3 and s gains $\gamma_1, \gamma_2, ..., \gamma_k$ with $s \leq k$.

Within the subsequent section, we describe how this service model can serve as grounding for the description of complex service compositions.

3.2 Service Composition Model

In order to represent service compositions properly as well as to control their execution, two main concepts are introduced in the following. First, we describe the control of the service compositions by a bipartite graph concept; afterwards, we discuss the global handling of gains within a service composition. For representing the execution order of services, interleaving as well as their possible parallel execution, a workflow graph is introduced controlling the execution of the single components within a service composition. Each transition of the graph corresponds to a service, which is modeled as introduced in section 3.1, i.e. contains I/O parameters and a list of gains it can generate during execution.

When passing a transition from one vertice of the workflow graph to another one, an action representing a service functionality is performed, which consumes the service's input and generates a finite set of gains and a finite set of outputs. Edges between vertices are optionally annotated with guards, which restrict the transition of the edge by requiring the presence of specific gains or previously generated outputs. Since outputs of a service are not necessarily required as inputs for a service reached by the next edge but may become relevant after multiple other services have been executed, a second graph is defined, specifying the dataflow within a service composition. This dataflow graph represents the flow of outputs from one service component to the input of another with its edges. The passage of a transition can thus be restricted by a guard operating on a set of gains and on the availability of all inputs that have to be created as outputs of other services before.

As motivated, gains generated during the execution of services are semantically described and supposed to represent the functionality encapsulated by the services. For instance, a service receiving a string as input and representing a map on a screen showing the location of the city specified within the input string may have a Boolean as output, indicating whether the service execution was successful or not. The coevally generated gain is an abstraction of the fact that the map is pushed to the user in case the service execution was successful. Notably, (as outlined in section 3.1) only a subset of the annotated gains has to be generated during the service execution. Given other inputs, the same service may perform a direct call setup between the requester and the restaurant nearby. This call setup would also be annotated to the service as a gain. Which of the two gains is really generated then depends on the inputs and cannot be derived from the service description itself.

Within the approach presented in this work, gains are stored globally within a gain space. The gain space thus enables a global view of gains generated during the execution of a service composition. Access restrictions for the gain space emerging from security related issues have to be formulated by special guards; however, this topic is out of scope for this paper and part of currently ongoing research. Beside the semantic representation of a gain itself, the service that has generated it is kept. By handling gains globally during the execution of a service composition, transition guards within the workflow graph can operate on all gains already generated. Thus, a guard may check whether a specific gain has been created or whether it has been created by the appropriate service component.

In the following section, the bipartite graph representation of a service composition is formally introduced, specifying its guard based transition behavior and its gain consideration mechanisms.

3.3 Bipartite Graph Concept

The representation of the workflow graph bases on timed automata and has already been formally introduced in our previous work [1]. In the following, we will shortly outline the most important parts of the graph representation of service compositions.

In general, a timed automaton [14,15] is finite automaton extended by a set of real-time valued clocks; for the remainder of this paper, we follow the definition provided by Clarke et al. [16]. In short, a timed automaton is a digraph with nodes that are connected with transitions. A transition itself is labeled with a guard and an action. In case the guard is satisfied, the passage of a transition leads to the execution of the respective action. For timed automata, guards can also operate on a special type of variables, so-called *clocks*, which are real-valued variables that progress simultaneously. Thus, in case one clock is increased by a value δ, all other clocks are augmented by the same δ. Moreover, a *reset* operation can be applied to a finite set of clocks, resetting their value to zero when passing an according transition.

Since actions are supposed to model the behaviour of a service, the execution of an action α is mapped to the consumption of inputs, the creation of outputs and the generations of gains. Inputs and outputs are regarded as typed variables bounded to specific ports, enabling the definition of I/O passage by means of a dataflow graph. The domains of variables thus constitute their primitive data type.

We model Workflow Graphs Θ as timed automata, extended by the ability of actions to operate on typed variables and gains. Typed variables are defined as 2-tuples $(x, \Gamma(x))$, where x is a variable and $\Gamma(x)$ its respective domain. For now, we only consider variables x with $x \in R \subset \mathbb{R}$. Let $\mathcal{V}$ be a finite set of typed variables. Regarding the gain space $\mathcal{G}$, we define $\mathcal{G}^*$ as a final set of variables $\gamma^* \in \{0, 1\}$ indicating whether a gain γ has been created (γ^* set to 1) or not (γ^* set to 0). We assume that $\mathcal{V}$ contains at least all variables of the gain space, thus, $\mathcal{G}^* \subseteq \mathcal{V}$.

Guards are logical formulas evaluating to a Boolean, blocking the passage of a transition within the workflow graph in case it is evaluated to *false*. Out model operates on *guard constraints* $\mathcal{C}(X \circ R \backslash \mathcal{V})$ that contain *clock constraints* $\mathcal{C}(X)$ and *real-value constraints* $\mathcal{C}(R \backslash \mathcal{V})$ over $\mathcal{V}$. A clock constraint is a conjunctive formula of atomic constraints of the form $x * n$ or $x - y * n$, where x and y are considered as real-valued clocks, $* \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. A guard constraint is a propositional logic formula $x * n$, where $x \in \mathcal{V}$ is a typed variable, $n \in \mathbb{R}$ and $* \in \{<, \leq, \geq, >, ==\}$. A guard constraint is either a clock constraint, a real-value-constraint, or a conjunctive form of both.

We now define the workflow graph as a timed automaton whose actions and guards can also operate on typed variables [1].

Definition 1 (Workflow Graph). *A workflow graph Θ is a timed automaton, where*

1. *Σ is a finite alphabet. The alphabet represents the actions which are identified by the single services within the service composition.*

2. *S is a finite set of locations defining the service compositions's current state,*
3. *$s_0 \in S$ are the initial locations defining the initial state of the service composition,*
4. *X is a set of clocks,*
5. *$I : S \rightarrow \mathcal{C}(X)$ assigns invariants to locations, restricting the system in the amount of time it is allowed to remain in the current state, and*
6. *$T \subseteq S\times\ \mathcal{C}(X \circ R\backslash\mathcal{V})\ \times\Sigma \times 2^X \times S$ is the set of transitions, denoting the execution of a service (represented by an action α). The passage of a transition (and thus the execution of a service) thereby depends on whether the according guard is met.*

Abbreviatory, we will write $s \overset{g,\alpha,\lambda}{\rightarrow} s'$ for $\langle s, g, \alpha, \lambda, s'\rangle$, i.e. the transition leading from location s to s'. The transition is restricted by the constraint g (often called guard); $\lambda \subseteq X$ denotes the set of clocks that is reset during the transition passage.

A formal specification of the transition relation of the proposed workflow graph can be found in our previous work [1]; we conclude the key principle of the theory in the following. The state transition relation has to enable two basic operations: the resetting of clocks to zero and the execution of an action. Therefore, two transition relations are introduced. The *delayed transition* enables a timed automaton the remain in a state and elapse a specific amount of time as long as the state invariant is not violated. The *action transition* describes the passage of a transition from a state s to a state s', preconditioning that the transition guard is fulfilled and the state invariant of s' is not violated. During the passage of the transition, a finite amount of time can be elapsed. However, this passage of time is optionally and restricted by the state invariant of s'.

Figure 2 shows an exemplary part of a workflow graph and its respective transition annotations.

In order to decide, whether an output of a service component is required as an input for another one, a dataflow graph is introduced in the following [1]. Dataflow graphs keep the same locations as the workflow graph introduced in Definition 1 and use labeled transitions to express inter-component data passing.

Definition 2 (Dataflow Graph Ω). *A dataflow graph Ω is a labeled directed digraph $\Omega = \{N, P, E\}$, where,*

1. *N is a final set of labeled nodes, which is equivalent to the set of locations S held in the according workflow graph Θ,*
2. *P is a set of port mappings represented by 2-tuples $p = (p_1, p_2)$ indicating the passage of the output from port p_1 to the input port p_2, and*
3. *$E \subseteq N \times P \times N$ is a final set of labeled directed transitions.*

Each location represents the data generated by the execution of action α_i; we therefore label a node with $d(\alpha_i)$ indicating the output data from action α_i. If a transition is passed within a workflow graph Θ entailing the execution of action α_i, all outgoing transitions from location $d(\alpha_i)$ within the according dataflow graph Ω are passed. A transition passage $d(\alpha_i) \overset{(p_m,p_n)}{\rightarrow} d(\alpha_j)$ within Ω effectuates

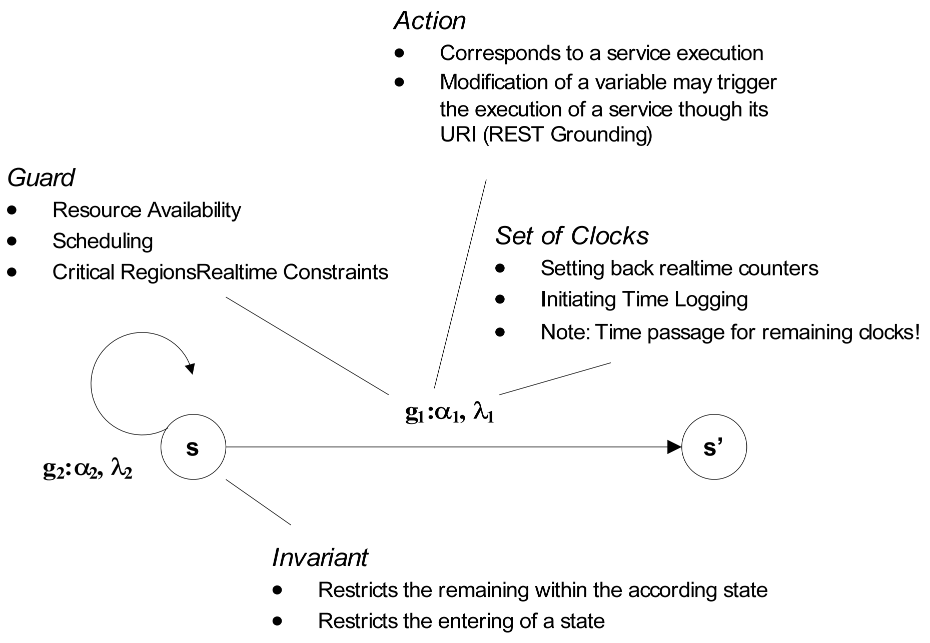
Action
Corresponds to a service execution
Modification of a variable may trigger the execution of a service though its URI (REST Grounding)
Guard
Resource Availability
Scheduling
Critical RegionsRealtime Constraints
Set of Clocks
Setting back realtime counters
Initiating Time Logging
Note: Time passage for remaining clocks!
g1 :α1, λ1
s
s'
g2:α2, λ2
Invariant
Restricts the remaining within the according state
Restricts the entering of a state

Fig. 2. Abstract Fragment of a Workflow Graph

that the output at port p_m from action α_i is redirected to input port p_n of action α_j in case action α_i is executed within Θ.

A service composition is defined by its workflow and dataflow graph, i.e. is defined as a 2-tuple $\langle \Theta, \Omega \rangle$.

4 Applying Genetic Operators to Service Compositions

As a proof of concept for the dynamicity of the introduced service composition model, we present techniques for the generation of new service compositions through the application of generic operators. Therefore, existing service compositions are transformed and subsequently evaluated with regard to their functional equivalence to other present service compositions. Within section 4.1, a definition for stable service compositions is introduced, specifying the class of proper service composition transformations. Section 4.2 then discusses the genetic operators that can be applied to service compositions in order to modify their structure. Section 5 later deals with the evaluation of stable service compositions with regard to their functionality.

4.1 Stable Service Composition Transformations

A *Service Composition Transformation* is defined as the swapping of a non-empty set of transitions within the workflow graph of a service composition.

Thus, the set of transitions $T = \{t_0, ..., t_i, ...t_{i+m}, ..., t_k\}$ is replaced by $T' = \{t_0, ..., t'_i, ...t'_{i+n}, ..., t_k\}$. The swapping is not restricted to sets of transitions of the same cardinality. Thereby, inputs of some services may not be available any more, since they have been produced by actions modeled as transitions that have been removed by the transformation procedure. The workflow and dataflow graph both have to be restructured in order to integrate the new transitions and the related I/O flow. In case every input of the new dataflow graph is generated -enabling its execution with regard to holistic input availability- , the service composition is regarded as *stable* and can be evaluated by means of its functionality as discussed in section 5. Otherwise the composition is considered to be morbid and is therefore discarded. Within the next section, two genetic operators are introduced, defining the transformation of the workflow and dataflow graph by exchanging transitions as well as their final test of stability.

4.2 Genetic Operators

Service Composition aims at creating added-value by reusing existing functional elements. In addition to service discovery, service selection, and late-binding, decentralized and dynamic computing environments require to consider varying availability of services. Service compositions are not bindable if for at least one of the constituting services there is no instance available.

For this reason, we investigate mechanisms for the availability-aware modification of service compositions on a structural level while preserving the semantics of the resulting services. In [17] we presented techniques for the generation of service composition variations through the appliance of generic algorithms. Existing service compositions are transformed and subsequently evaluated with regard to their functional equivalence to other service compositions. Therefore, the straightforward applicability of genetic operators was one of the major requirements for the definition of the graph presented in this work.

For the beginning, we realized a cross-over operator and a mutation operator. Both operators are basically built on inclusion and removal of graph elements. Usually, a transformation on the workflow graph also requires the dataflow graph to be recovered, while a transformation on the dataflow graph has no impact on the consistency of the workflow graph. Since both operators, mutation and cross-over are not restricted in the way they modify a workflow graph, there is a marginal probability that the corresponding dataflow graph will not be recoverable afterwards. In this case, the generated offspring is rejected as candidate before evaluation.

The mutation operator is randomly applied to workflow graph or dataflow graph. A transformation on the workflow is implemented to either remove transitions or include new ones by random. The inclusion of transitions depends on the types of services available in the service environment. The removal of transitions requires the subsequent recovery of the dataflow graph. If the mutation operator is only applied to the dataflow graph, it randomly changes the passages of data between input ports and output ports, removes constants, or includes new ones, all by preserving the correct typing.

The cross-over operator is applied to the workflow graph only. The operator also removes and includes transitions, except for the difference that the removed transitions are exchanged with the workflow graph of another service composition, while the transitions obtained in return are included. Afterwards, the dataflow graphs of both compositions are recovered with respect to the newly introduced transitions.

First practical tests for service composition transformation by applying mutation and cross-over operator showed the appropriateness of the graph representation presented in this work for flexible modification in dynamic environments.

5 Evaluation of Functional Equivalence of Service Compositions

By the transformation of service compositions as introduced in section 4, executable service compositions are created whose functionality is unknown. Research effort has been spent on identifying the functionality of a service and to create an appropriate service description accordingly [18]. However, the dynamic creation of semantic service description is still an open problem when desisting from approaches basing on constrictive assumptions as within [19]. Currently, our work aims at transferring semantic descriptions instead of creating new ones. Therefore, the new description is geared to the description of the service is was derived from by transformation. The new service is thus evaluated with regard to its similarity in its functional behavior to other available service compositions.

Since the generation of gains during the execution of a service may depend on the actual service inputs, services cannot be evaluated offline with regard to their functionality (which is considerably dependent from the gains it produces), but have to be estimated during runtime. Therefore, newly created service compositions are compared with existing ones in a *Silent Execution Mode*. Here, two compositions are executed within a Sandbox [8] like environment, where their execution does not directly affect the current state of the computing environment. During silent execution, initial inputs are randomly generated and a selected service composition as well as the transformed one are executed with those input parameters. The outputs and generated gains of both service compositions can then be compared by a distance function δ allowing to make a statement on the proximity of effect and outputs the service compositions have been provided. A possible distance function δ can be given as follows.

Definition 3 (Distance Function δ). *Let m be the number of outputs, n the number of gains created by at least one of two service compositions. During a silent execution run, the distance $\delta \in (0, 1)$ between the compositions is given as*

$$\delta = \frac{\sum_{i=0}^{m-1} outAccur_i + \sum_{j=0}^{n-1} gainAccur_j}{m+n},$$

where $outAccur_i$ and $gainAccur_j \in (0,1)$ denote the level of similarity of output i or gain j of both compositions, respectively. $gainAccur_j$ is 1 in case gain j has been created by both service composition executions, else 0. The value of $outAccur_i$ depends on the primitive type of the output variable. For `String` *and* `Char`*, a test of equivalence is performed, i.e. $outAccur_i = 1$ if the* `Char` *or* `String` *created by the compositions match exactly, otherwise 0. For numeric variables such as* `Int`*,* `Float` *and* `Double`*, $outAccur_i$ is given by $\frac{out_{i1}}{out_{i2}}$ where out_{i1} is the smaller of the two outputs. In case an output was only created by one composition, $outAccur_i$ is 0. If multiple runs are performed for calculating the distance of two service compositions, δ is given as the arithmetic average of the single distance values.*

A higher similarity between both compositions is thus expressed through a higher value of δ. We identify the deviance in terms of output and gain accuracy between two service compositions with the deviance in service composition functionality. That is, two service compositions generating exactly the same outputs and gains are considered as functionally equivalent. The distance δ moreover remains a variation parameter for service composition evaluation. For instance, a composition may be considered as equivalent if the distance to another service composition is bigger than 0.9, because the deviance may be caused by simulation inaccuracy, different service contexts or randomness.

The exactness of such an evaluation technique apparently also depends on the number of simulation runs used to calculate the distance between the two service compositions' outputs. The number of simulation runs therefore constitutes the second parameter (beside the acceptance border for δ) enabling the assignment of a certain level of trust to a simulated distance between two service compositions. For instance, two service compositions may be regarded as functionally equivalent, if they have a distance greater than 0.8 which was derived from at least 50 simulation runs. In case a new service composition c' derived by the appliance of a genetic operator to a composition c is evaluated as functionally equivalent to c, the semantic description of c is overtaken for c'.

6 Conclusion and Future Prospect

Within this paper, we presented a workflow model for the representation of service compositions featuring rapid modifiability through gain-oriented service descriptions. The transformation of service composition plans has been exemplarily demonstrated by the application of algorithms derived from genetic programming. Finally, a method for the evaluation of resulting service compositions has been outlined, enabling the comparison of service composition plans with regard to the outputs and gains they generate.

The automatic creation of workflow oriented service composition plans based on a requested set of gains is part of ongoing research.

References

1. Pfeffer, H., Linner, D., Steglich, S.: Modeling and Controling Dynamic Service Compositions. In: Proceedings of The Third International Multi-Conference on Computing in the Global Information Technology (ICCGI 2008), Athens, Greece (2008)
2. Bucchiarone, A., Gnesi, S.: A Survey on Services Composition Languages and Models. In: Bertolino, A., Polini, A. (eds.) Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006), Palermo, Sicily, Italy, pp. 51–63 (2006)
3. Thatte, S.: XLANG - Web Services for Business Process Design (2001)
4. Leymann, F.: Web Service Flow Language (WSFL 1.0). In: IBM (May 2001)
5. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1 (May 2003), (Accessed, November 2007), `http://www.ibm.com/developerworks/library/specification/ws-bpel/`
6. Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., Weerawarana, S.: Business Process Execution Language for Web Services (Version 1.0) (July 2002)
7. van der Aalst, W.: Don't go with the flow: Web services composition standards exposed (2003)
8. UNCEFACT and OASIS: ebXML Business Process Specification Schema Version 1.0.1 (2001)
9. Ankolekar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K.: DAML-S: Web Service Description for the Semantic Web. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342. Springer, Heidelberg (2002)
10. The OWL Services Coalition: OWL-S: Semantic Markup for Web Services (November 2004) (Accessed February 26, 2007), `http://www.daml.org/services/owls/1.1/`
11. Pfeffer, H., Linner, D., Jacob, C., Steglich, S.: Towards Light-weight Semantic Descriptions for Decentralized Service-oriented Systems. In: Proceedings of the 1st IEEE International Conference on Semantic Computing (ICSC 2007). Volume CD-ROM, Irvine, California, USA (2007)
12. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. In: GECCO 2005: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 1069–1075. ACM, New York (2005)
13. Jaeger, M., Engel, L., Geihs, K.: A Methodology for Developing OWL-S Descriptions. In: First International Conference on Interoperability of Enterprise Software and Applications Workshop on Web Services and Interoperability (INTEROP-ESA 2005), Springer, Heidelberg (2005)
14. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of of the 5th Annual Symposium on Logic in Computer Science, pp. 414–425. IEEE Computer Society Press, Los Alamitos (1990)
15. Dill, D.: Timing assumptions and verification of finite-state concurrent systems. In: Sifakis, J. (ed.) Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems. LNCS, vol. 407, pp. 197–212. Springer, Heidelberg (1989)

16. Clarke, E.M.J., Grumberg, O., Peled, D.A.: Model Checking. The MIT Press, Cambridge (1999)
17. Linner, D., Pfeffer, H., Steglich, S.: A genetic algorithm for the adaptation of service compositions. In: Proceedings of the 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (2007)
18. Babik, M., Hluchy, L., Kitowski, J., Kryza, B.: Generating Semantic Descriptions of Web and Grid Services. In: Kacsuk, P., Fahringer, T., Nemeth, Z. (eds.) Distributed and Parallel Systems - From Cluster to Grid Computing (Proceedings of the 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS)), Innsbruck, Austria. International Series in Engineering and Computer Science (ISECS), pp. 83–93 (2006)
19. Caprotti, O., Davenport, J.H., Dewar, M., Padget, J.: Mathematics on the (Semantic) NET. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 213–224. Springer, Heidelberg (2004)