Best Service Synthesis in the Weighted Roman Model

Diego Calvanese and Ario Santoso

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy
{calvanese,santoso}@inf.unibz.it

Abstract. This paper presents an extension of a framework for synthesizing a composition of services, named Roman Model, such that it is able to model the best service composition synthesis problem. In such extension, which we call the Weighted Roman Model, the services are modeled as Weighted Transition Systems so that one can capture the cost of operations executed by a service. Within this setting, we can make a comparison among all possible compositions of the available services by considering the total cost of operation execution performed by each possible composition of services for each interaction between the service and the client. Besides defining the notion of best composition, we also propose an algorithm for synthesizing the best composition and show that it is sound and complete.

1 Introduction

Services are modular applications that can be described, published, located, invoked, and composed over a variety of networks (including the Internet): any piece of code and any application component deployed on a system can be wrapped and transformed into a network-available service, by using standard (XML-based) languages and protocols (e.g., WSDL, SOAP, etc.). One of the interesting aspects is that this wrapping allows each program to export a simplified description of itself, which abstracts from irrelevant programming details. The promise of Web services is to enable the composition of new distributed applications/solutions: when no available service can satisfy a client request, (parts of) available services can be composed and orchestrated in order to satisfy the request itself.

In reality, there could be several possible ways of composing the available services for satisfying the requested service. However, not all compositions of services can be considered as equally good. They might have different resource consumption (e.g., bandwidth, memory, etc). In this situation, one might be interested in finding the "best" composition among all possible ones.

In this paper, we consider the framework for service composition adopted in [2,5,13,10,3,4], sometimes referred to as the "Roman Model" [9]. In this work, we extend the Roman Model to the Weighted Roman Model in such a way that it is able to model the problem of synthesizing the best composition of services. Moreover, we also describe a sound and complete algorithm for synthesizing the best composition.

The rest of the paper is structured as follows. The next section explains the Roman Model and service composition in this setting. Section 3 presents the Weighted Roman

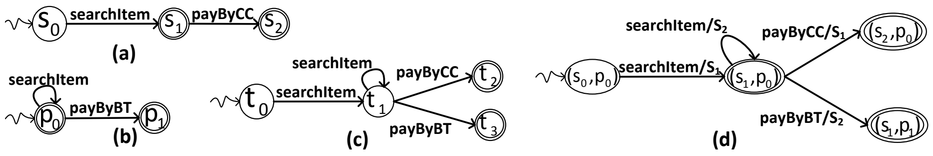

Fig. 1. (a) Service $\mathcal{S}_1$ (b) Service $\mathcal{S}_2$ (c) Target Service $\mathcal{S}_t$ (d) Possible composition of services $\mathcal{S}_1$ and $\mathcal{S}_2$ that simulates the target service in (c)

Model and defines the notion of best service composition. Section 4 presents the technique for synthesizing the best service composition within the Weighted Roman Model. Finally Section 5 concludes the paper.

2 Service Composition and The Roman Model

Services in the Roman Model (RM) represent software artifacts capable of performing operations. A service *interacts* with the client through the following steps: *(i)* it offers to its clients a choice of operations it can perform, *(ii)* based upon the service state; the client chooses one of the offered operations, and *(iii)* the service executes it, changing its state accordingly. Fig. 1 shows an example of services in the RM in the scenario of a simple online shopping system. Intuitively, in the service $\mathcal{S}_1$ in Fig. 1(a), the interaction can be started at the initial state s_0, where the service offers the "searchItem" operation (i.e., the "searchItem" operation is *executable* in this state). After executing this operation, the service's state changes to s_1. In s_1 the interaction can be terminated since it's a final state, or the client can continue requesting the operation "payByCC" (i.e., to pay by credit card). Similarly, in $\mathcal{S}_2$ after a finite sequence of item searches, the client can request a payment by bank transfer ("payByBT"). Formally, a *service* in RM is a transition system (TS) $\mathcal{S} = (S, \mathcal{O}, \delta, s_0, F)$, where: *(i)* S is the finite set of service's *states*; *(ii)* $\mathcal{O}$ is the set of possible *operations* that the service recognizes; *(iii)* $\delta \subseteq S \times \mathcal{O} \times S$ is the service's *transition relation*, which accounts for its state changes; *(iv)* $s_0 \in S$ is the *initial state*; and *(v)* $F \subseteq S$ is the set of *final states*, i.e., those states where the interaction with the service can be legally terminated by the client. When $\langle s, o, s' \rangle \in \delta$, we say that *transition* $s \xrightarrow{o} s'$ *is in* $\mathcal{S}$. Given a state $s \in S$, if there exists a transition $s \xrightarrow{o} s'$ in $\mathcal{S}$, then operation o is said to be *executable* in s. A transition $s \xrightarrow{o} s'$ in $\mathcal{S}$ denotes that s' is a possible successor state of s, when operation o is executed in s. In this work we consider only *deterministic* services, i.e., there are no two distinct transitions $s \xrightarrow{o} s'$ and $s \xrightarrow{o} s''$ with $s' \neq s''$. Such services are *fully controllable* by just selecting the operation to perform next.

A *community* $\mathcal{C} = \langle \mathcal{S}_1, \ldots, \mathcal{S}_n \rangle$ *of available services* consists of n available services that share the same operations $\mathcal{O}$. A *target service* is a desired service that also shares the operations in $\mathcal{O}$. The *goal of the composition in the RM* is to maintain with the client the same, possibly infinite, interaction that he would have with the (virtual) target service, by suitably orchestrating the (concrete) available services. An *orchestrator* is a system component that is able to activate, stop, and resume any of the available services, and to instruct them to execute an operation among those executable in their current state. Essentially, the orchestrator, at each step, will consider the operation chosen by the client (according to the target service) and delegate it to one of the services that can execute it, and so on, possibly at infinitum. The aim of the orchestrator is to maintain the interaction with the client, as if it was interacting with the target service, without ever failing to delegate an operation chosen by the client to one of the available services.

Formally, an orchestrator is a *function* from *(i)* the *history* of the whole system (which includes the state trajectories of all available services and the trace of the operations chosen by the client, and executed by the services), and *(ii)* the *operation* currently chosen by the client, to the index i of the service $\mathcal{S}_i$ to which the operation has to be delegated. Intuitively, the orchestrator *realizes* a target service if and only if, at every step, given the current history of the system, it is able to delegate every operation executable by the target to one of the available services. Hence, the orchestrator controls the evolutions of the services' states in the community s.t. together they "mimic" the target service.

The goal of service composition is to synthesize an orchestrator that realizes the target service by exploiting available services. In [12,3], the problem has been tackled using a simulation-based approach. The idea is essentially checking if the target service is *simulated by* the *Community-TS*, which is the *asynchronous product* of the services in $\mathcal{C}$. Intuitively, it checks if the Community-TS supports all possible interactions that are supported by the target service (i.e., it checks if there is always a way to realize any interaction that is possible between the client and the target service).

Going back to the running example in Fig. 1, suppose the community consists of services $\mathcal{S}_1$ and $\mathcal{S}_2$. Taking the asynchronous product we obtain the Community-TS and we can find a fragment of it that simulates the target service in Fig. 1(c). This fragment, which encodes the specification of the orchestrator, is shown in Fig. 1(d). It says how the requested operation can be delegated to the services in the community. For example the execution of operation "payByCC is delegated to service $\mathcal{S}_1$ (denoted by the label "payByCC/S_1" in the transition).

3 Best Service Composition and the Weighted Roman Model

An extension of the RM into the Weighted Roman Model (WRM) is partly inspired by the work on weighted automata [6]. The WRM framework aims at addressing the problem of best service composition synthesis. The target service in the WRM is represented using a transition system as in the RM, while the available services are represented using weighted transition systems (WTS). Intuitively, a WTS is a TS augmented with a semiring $\hat{S} = (\hat{C}, \hat{+}, \hat{\cdot}, \hat{0}, \hat{1})$[1][8]. We use the semiring elements in the set $\hat{C}$ to denote the cost of service's operation execution. Fig. 2 shows an example of services in the WRM. Intuitively, in service $\mathcal{S}_1$ (Fig. 2(a)), the cost of executing operation "searchItem" is 4. We use the semiring multiplication operator $(\hat{\cdot})$ for the aggregation of service's operations execution costs, and the semiring addition operator $(\hat{+})$ for comparing the costs. As a prominent example, consider the tropical semiring $\hat{T} = (\mathbb{Z} \cup \{\infty\}, min, +, \infty, 0)$, whose elements are the integers together with positive infinity, whose multiplication operator is addition over integers, and whose addition operator is the minimum operator.

[1] A *semiring* is a structure $\hat{S} = (\hat{C}, \hat{+}, \hat{\cdot}, \hat{0}, \hat{1})$, where $\hat{C}$ is a nonempty set closed under a binary, associative, and commutative *semiring addition*, $\hat{+}$, and a binary, associative *semiring multiplication* $\hat{\cdot}$, respectively with $\hat{0}$ and $\hat{1}$ as neutral elements, and where $\hat{\cdot}$ distributes over $\hat{+}$.

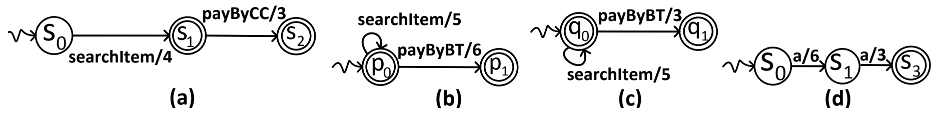

Fig. 2. Example of available services in the WRM: (a) Service $\mathcal{S}_1$ (b) Service $\mathcal{S}_2$ (c) Service $\mathcal{S}_3$ (d) Example of modeling cost dependencies inside a service in the WRM

In this setting, the total cost of service's operations execution is aggregated by the addition operator (i.e., just the sum over each cost of operations execution) and then we can compare costs of service's operations execution by using the minimum operator.

To make the comparison meaningful, we restrict the usage of semirings by adding the requirement that $c_1 \hat{+} c_2 = c_1$ or $c_1 \hat{+} c_2 = c_2$, where $c_1, c_2 \in \hat{C}$ (i.e., $\hat{+}$ is an operator for comparing two semiring elements). We call such semiring a *comparison semiring*. Given two semiring elements c_1 and c_2 in a comparison semiring, we say that c_1 *is better than* c_2 if $c_1 \hat{+} c_2 = c_1$, and vice-versa. Consider again the tropical semiring, where addition is the minimum operator. In this case c_1 *is better than* c_2 if $min(c_1, c_2) = c_1$ (i.e., if c_1 is smaller than c_2).

The usage of a semiring in our framework gives us flexibility in defining the notion of "best". For example, we can use the tropical semiring if we are interested in the minimum cost, while we can use the arctic semiring $\hat{A} = (\mathbb{Z} \cup \{-\infty\}, max, +, -\infty, 0)$ if we are interested in the maximum cost. Moreover, it gives us flexibility in defining the domain of the cost, for example whether it ranges over integers, reals, positive integers, etc. Another example is where one models the situation where the cost of operation execution represents the probability of a service being successfully executed, in this case the cost might range from 0 to 1. However, to make explanations more intuitive, from now on we focus on the tropical semiring only.

Formally, a service in the WRM is a WTS $\mathcal{WS} = (S, \mathcal{O}, \hat{T}, \nu, s_0, F)$, where S, $\mathcal{O}$, s_0, and F are as for a TS, $\hat{T} = (\mathbb{Z} \cup \{\infty\}, min, +, \infty, 0)$ is the tropical semiring, and $\nu : S \times \mathcal{O} \times S \times \hat{C}$ is the service's *transition relation*. When $\langle s, o, s', c\rangle \in \nu$, we say that *transition* $s \xrightarrow{o,c} s'$ *is in* $\mathcal{S}$. Intuitively, the semiring element c in the transition $s \xrightarrow{o,c} s'$, represents the cost of performing operation o in state s. In this case, the fact that available services are *deterministic* means that there are no two distinct transitions $s \xrightarrow{o,c} s'$ and $s \xrightarrow{o,c'} s''$ in $\mathcal{S}$ such that $s' \neq s''$ or $c \neq c'$. The notion of community of available services in the WRM is similar to the one in the RM except that the services are represented by WTSs. As for simulation checking in the RM, we can construct a *Community-WTS* by taking the asynchronous product of all available services WTSs.

In the WRM, we can also model some scenarios in which the cost of executing an operation depends on the execution of another operation. Fig. 2(d) shows an example where the second execution of operation a is modeled as having a smaller cost than the first execution of a. In general, we could represent the fact that an operation a executed at some state where a has already been executed has smaller cost. However, there are many scenarios that cannot be captured easily, for example when the cost of executing an operation in a service depends on an operation execution by another service.

Now we introduce the notion of best composition in the WRM through our running example. Suppose the community of available services consists of the service $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ in Fig. 2. The target service $\mathcal{S}_t$ is still the one in Fig. 1(c). Two possible fragments

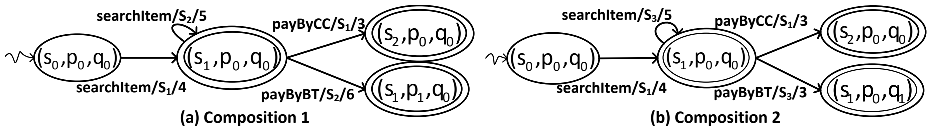

Fig. 3. Two possible composition of services $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$ for the target service $\mathcal{S}_t$ in Fig. 1(c)

of the Community-WTS that simulate $\mathcal{S}_t$ (i.e., serve as a composition for $\mathcal{S}_t$) are shown in Fig. 3. Intuitively, the first composition uses services $\mathcal{S}_1$ and $\mathcal{S}_2$ to "mimic" the target service $\mathcal{S}_t$ and the other one uses $\mathcal{S}_1$ and $\mathcal{S}_3$. Considering the target service in Fig. 1(c), suppose the client requests to execute operation "searchItem" twice, followed by operation "payByBT". Intuitively she searches for the item in the shop twice and then purchases it by using a bank transfer. Formally, this is represented by the path

$$\tau = t_0 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{payByBT} t_3$$

in target service $\mathcal{S}_t$. Notice that τ starts at the "initial state" and ends at a "final state". We call this an *accepting path.* To realize the request, an orchestrator must be able to delegate the execution of the requested operations to the available services in the community. An orchestrator D_1 based on Composition 1 in Fig. 3(a) delegates the first "searchItem" request to $\mathcal{S}_1$, the second one to $\mathcal{S}_2$, and the "payByBT" request to $\mathcal{S}_2$. Formally, this delegation corresponds to the following path in the Community-WTS:

$$\tau' = (s_0, p_0, q_0) \xrightarrow{searchItem/S_1/4} (s_1, p_0, q_0) \xrightarrow{searchItem/S_2/5}$$
$$(s_1, p_0, q_0) \xrightarrow{payByBT/S_2/6} (s_1, p_1, q_0)$$

We call this a *realization path.* Since we use the tropical semiring, the *weight* of this realization path is just a summation of the weights of all operations along the realization path. In this case the weight of this realization path is 15. However, there might be more than one possible way to realize a certain sequence of operations request. In our examples, we might also delegate to $\mathcal{S}_3$ the execution of the second "searchItem" and of the "payByBT" requests. Hence, there might be more than one corresponding realization in the Community-WTS. Each of them has its own weight. To compare them and find the best weight, since here we use the tropical semiring, we take the minimum among all of them. Hence, in this case we get the best weight as the minimum weight among all possible realizations. In our example one possible best weight is 12 (possibly obtained by doing the delegation based on Composition 2).

The goal of the best composition synthesis is to synthesize the *best orchestrator D*, which informally means that for all possible sequences of operations requested by the client (which correspond to accepting paths in the target service), we have that for all possible delegations of those operations execution to the available services done by *D*, the total cost of this execution is the best among any other possible delegation done by all possible orchestrators. Considering again the tropical semiring, intuitively we are interested in finding the best orchestrator that minimizes the total cost of the realization of all possible interactions between the target service and the client that started from the initial state and end at a final state. It is not immediate to gain the decidability of this problem, since once we have a loop in the target service, the client can make an infinite number of different sequences of operations request. Hence we can't just enumerate all possible sequences of operations executions and check if a certain orchestrator can realize them all in the "best" way.

4 Best Composition Synthesis

Recall that given a Community-WTS $\mathcal{WC}$ and the target service $\mathcal{S}_t$, it can be shown that an orchestrator D for the given $\mathcal{WC}$ and $\mathcal{S}_t$ corresponds to a certain fragment of the $\mathcal{WC}$ that simulates $\mathcal{S}_t$. We call such fragment a *target service realization*. Intuitively, a target service realization encodes the specification for an orchestrator. Similarly, it can also be shown that a best orchestrator corresponds to a certain fragment of $\mathcal{WC}$, which we call a *best target service realization*. More formally, a best target service realization is a fragment $\mathcal{SR}$ of $\mathcal{WC}$ s.t. for all accepting paths τ in $\mathcal{S}_t$, the weight of each possible realization path of τ in $\mathcal{SR}$ is the best. Intuitively, a best target service realization encodes the specification of a best orchestrator. Knowing this fact, we can reduce the problem of checking the existence of a best composition to the problem of checking the existence of the best target service realization. Moreover, a best orchestrator can be synthesized from a best target service realization, if it is exits.

Before presenting the algorithm for checking the existence and synthesizing a best target service realization, we introduce some preliminary notions: *(i)* A *simple cycle path* is a cycle path that has no state repetition in it, except for the first and the last states, which coincide. In Fig. 1(c), the path $t_1 \xrightarrow{searchItem} t_1$ is a simple cycle while $t_1 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1$ is not. *(ii)* A *k-bounded accepting path* is an accepting path where the length of each cycle path in it is less than or equal to k. In our example, for $k = 2$, the path $\pi' = t_0 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{payByCC} t_2$ is a k-bounded accepting path while the path $\pi'' = t_0 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{searchItem} t_1 \xrightarrow{payByCC} t_2$ is not.

The algorithm for checking the existence and synthesizing the target service best realization takes the target service and the community as input. We sketch it here briefly *(1)* For each fragment $\mathcal{SR}$ of $\mathcal{WC}$, repeat the following steps. *(2)* Verify if $\mathcal{SR}$ simulates the target service. *(3)* Verify if for each simple cycle path in the target service, we have that the weight of all its possible realization paths in $\mathcal{SR}$ is the best among all its possible realization paths in $\mathcal{WC}$. *(4)* Verify if for each possible k-bounded accepting path in the target service, the weight of all its possible realization paths in $\mathcal{SR}$ is the best among all its possible realization paths in $\mathcal{WC}$, where k is equal to the size of the community transition system. *(5)* If $\mathcal{SR}$ fulfills the verification in Steps 2, 3, and 4, the algorithm returns "yes" and $\mathcal{SR}$ is one possible target service best realization. Otherwise, go back to Step 1 to and continue the check with the next fragment. If none of the fragments fulfills the verification in Steps 2, 3,and 4 then the algorithm returns "no". It can be shown that the algorithm above is sound, complete, and always terminates, and that its complexity is double exponential in the combined size of the target service and the community of available services.

In our running example, suppose the target service is as in Fig. 1(c), and that for simplicity of explanation there are only two possibles fragments of Community-WTS as in Fig. 3 (Note: In this example we might have more than these two fragments). In the second step, it is easy to see that both fragments simulate the target service. For the third step, there is only one simple cycle, namely $t_1 \xrightarrow{searchItem} t_1$ and either in Fig. 3(a) or Fig. 3(b) there is only one possible realization path and both of them have the same weight. In our example, the one which has a realization path with different weight is only $t_0 \xrightarrow{searchItem} t_1 \xrightarrow{payByBT} t_3$. The corresponding realization path in Fig. 3(a) is $(s_0, p_0, q_0) \xrightarrow{searchItem/S_1/4} (s_1, p_0, q_0) \xrightarrow{payByBT/S_2/6} (s_1, p_1, q_0)$ with weight equal to 10, and the one in Fig. 3(b) is $(s_0, p_0, q_0) \xrightarrow{searchItem/S_1/4} (s_1, p_0, q_0) \xrightarrow{payByBT/S_3/3} (s_1, p_0, q_1)$ with weight equal to 7. Due to space limitation, we can't give the full illustration, but one can check that in this case for all of the possible k-bounded accepting paths in the target service, we have that the weight of each possible corresponding realization path in the fragment in Fig. 3(b) is the best, while this does not hold for the fragment in Fig. 3(a). Since the fragment in Fig. 3(b) satisfies the checks in Step 2, 3, and 4, the algorithm return "yes", and this fragment is one possible target service best realization.

5 Related Work and Conclusions

In this work we have proposed a weighted extension of the Roman Model, named Weighted Roman Model. It enhances the Roman Model with the capability to model the cost of service's operation execution and allows one to address the problem of best composition synthesis. We have shown that the problem of checking the existence and synthesizing the best composition can be addressed by checking the existence and synthesizing the so called best target service realization (encoding the specification of the best orchestrator). Relying on this result, we proposed a sound and complete algorithm for checking the existence and synthesizing the best target service realization.

We provide here a brief overview of related work in the literature. In the SM4ALL project [4], the Roman Model is used as the underlying framework for establishing a collaboration of services, involving composition. The framework is applied to the real world scenario studied in the project, namely that of private homes for users with different abilities and needs. The Roman Model is adopted also in [7], which addresses an optimization problem in the area of service composition. However, it considers finding the best composition for an ad-hoc interaction, i.e., for a given sequence of requested operations. Instead, we consider here all possible sequences of requested operations, hence in general we do not know which might be the next operation requested by the client. Also, the use of a semiring gives more flexibility in defining the meaning of optimum cost. We mention also works where the quantitative aspect comes into play for measuring similarity between transition system-like structures. [14] presents a similarity measure on control flow graphs, which are formalized as labeled transition systems, that is based on a weighted variant of simulation. The work in [11] proposes a technique for matching statecharts that is motivated by model management in software engineering. However, in both works, the transition system-like structures do not contain quantitative information, as in our case.

One interesting further direction of our work is to model the situation where we consider the initial and final weights of a service. Intuitively, the initial weight can model the cost of initializing the service and the final weight the cost of terminating it. Another interesting direction is to analyze the intrinsic complexity of the best composition synthesis problem, and check whether our upper bounds can be improved. Fully taking into account data for verification and synthesis in the context of the Roman Model, or other service-based frameworks, is a very challenging task that has been tackled only recently, see, e.g., [1]. We are not aware of any work that addresses synthesis, fully taking into account data, even in an unweighted setting. It is a very interesting research direction, to tackle this problem, both for an unweighted and for a weighted setting.

References

1. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of relational artifacts verification. In *Proc. of the 9th Int. Conference on Business Process Management (BPM 2011)*, volume 6896 of *LNCS*, pages 379–395. Springer, 2011.
2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *Int. J. of Cooperative Information Systems*, 14(4):333–376, 2005.
3. D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi. Automatic service composition and synthesis: the Roman Model. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 31(3):18–22, 2008.
4. T. Catarci, F. Cincotti, M. Leoni, M. Mecella, and G. Santucci. Smart homes for all: Collaborating services in a for-all architecture for domotics. In *Collaborative Computing: Networking, Applications and Worksharing*. Springer Berlin Heidelberg, 2009.
5. G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 2007.
6. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009.
7. C. E. Gerede, O. H. Ibarra, B. Ravikumar, and J. Su. Minimum-cost delegation in service composition. *Theoretical Computer Science*, 409(3):417–431, 2008.
8. J. Golan. *Semirings and Their Applications*. Kluwer Academic Publishers, 1999.
9. R. Hull. Web services composition: A story of models, automata, and logics. In *Proc. of the 3rd IEEE Int. Conf. on Web Services (ICWS 2005)*, 2005.
10. A. Muscholl and I. Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.
11. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proc. of the 29th Int. Conf. on Software Engineering (ICSE 2007)*, pages 54–64, 2007.
12. F. Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA Università di Roma, Dipartimento di Informatica e Sistemistica, 2009.
13. S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, 2008.
14. O. Sokolsky, S. Kannan, and I. Lee. Simulation-based graph similarity. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCS*, pages 426–440. Springer, 2006.