Survey on Services Composition Synthesis Model

Ibrahima Kalil Toure[1,*], Yang Yang[2] and Shariq Hussain[3]

**[1,2,3] School of Computer and Communication Engineering, University of Science and Technology Beijing
Beijing, 100083, China**

Abstract

Current web services development tools are more sophisticated though ease of use, which leverage the creation of more web services thereof. This is the fact that, web services are being created and updated frequently, this multiplication of web services cannot be easily controlled by human being because it is almost impossible to analyze them and generate the composition plan. Composition of web services is the issue of synthesizing a new composite web service, obtained by combining a set of available (component) services, when a client request cannot be satisfied by available web services. To address this issue, three main models have been proposed as a solution. The OWL-S model, the Conversational model and the Roman model which is investigated here. In this paper, we propose a survey on the so-called Roman model and present the framework and all its extension. We also underline its drawback, shortcomings and some advantages, and then try to provide some research direction.

Keywords: *Web Service, Composition, Synthesis, Behavior*

1. Introduction

The rapid development of the information technology has facilitated the construction of application and their publication over the internet. Currently we are witnessing presence of large number of services, which make it difficult; to choose the right services to satisfy the user request, to coordinate available service for building more complicated and more flexible applications. Research on web services considers, as fundamental service composition i.e. how to compose and coordinate different services, to be assembled together in order to support more complex services and goals. Interestingly, many contributions on this issue come from the Artificial Intelligence (AI) community [1–2, 4, 8]. Despite the work done so far, service composition is still largely unexplored and to the best of our knowledge an overall agreed upon comprehension of what service and service composition are, in an abstraction and general fashion is still lacking.

Research on services composition encompasses many challenges, such as description, discovery, composition, synchronization, coordination, and verification [38]. In [39], the Service Oriented Architecture (SOA) is developed, which is seen as the basis architecture for services. SOA provides the basic operations necessary to describe, publish, find and invoke services. One of the main issues in Service Oriented Computing (SOC) is service composition [40]. The composition is required in the situation where any single available services cannot satisfy the client request, but a combination of them. In other words, the client request can only be satisfied by suitably combining (parts of) available services, also called component services in this context. Composition mainly enclosed two different issues [37]. The first, typically called composition synthesis, is concerned with synthesizing a composition of available services that satisfies a client request. The synthesis process produces a specification of how to coordinate, or orchestrate, the component services to fulfill the client request. Such a specification can be produced either automatically, i.e. using a tool that implements a composition algorithm, or manually by a human. The second issue, often referred to as orchestration, is concerned with how to actually execute the composition of the services produced by the composition synthesis, by suitably supervising and monitoring both the control flow and the data flow among the involved services.

In this paper, we are going to follow the footstep of [3] which proposed a brief survey on the Roman model, to provide a deep survey on this area with more detail and also we shall provide some research direction. The remainder this paper is organized as follows: Section 2 presents the Roman model and provides a description of its framework. Section 3 describes different extensions and variants of the Roman model including the techniques used. In Section 4 we conclude the paper and try to provide future research direction.

2. Roman Model

In the Roman model, the services are represented as finite transition system with respect to their conversational behavior. In [3] the Roman model is a framework for composing conversational services, where:

(i) Each service is formally specified as a transition system that captures the possible conversation with a generic client;
(ii) The desired specification is a target service, that described itself as transition system;

* Corresponding author

(iii) The aim is to synthesize an orchestrator which realizes the target service by exploiting execution fragments of available services.

The Roman model well exemplifies what can be achieved by composing conversational services and, also uncovers relationships with automated synthesis of reactive processes in verification and planning AI.

2.1 General Framework

In this section we provide a description of the Roman model, by following [5, 22–23, 28]. The service is defined as a software artifact (delivered over the internet) that interacts with its client in order to perform a specified task [5]. This framework can be built from an abstract and conceptual point of view, based on the following two facets:

(i) The service scheme specifying functional requirements (a service scheme may also specify non-functional requirements, such as quality and performance), i.e. what a service does;
(ii) The service instance occurred as a result of service being effectively run and constantly interacting with a client.

A client can be a human or another service. A service is characterized in terms of sequence of actions that is able to execute, meaning its behavior. Typically, an atomic interaction results from the following steps:

(i) At current state, client can request different operations depending on the availability of service;
(ii) The client selects one of the offered operations;
(iii) The available service executes client's selection, moves to a new state, according to its behavioral specification, and iterates to the next step (iterates the process).

Originally, in Roman model [23–24], available services are deterministic, which makes them fully controllable and the result of executing an operation in a given state is a certain successor state. In a clear expression, one can fully control available services transition by assigning operation execution.

Formally, a service behavior is a transition system $S = \{O, S, s^0, S^f, g\}$ where:

(i) O is the set of possible *operations* that the service recognizes, also called alphabets of operation;
(ii) S is the finite set of service's *states*;
(iii) $s^0 \in S$ is the initial state;
(iv) $S^f \subseteq S$ is the set of *final states*, i.e. those states where the interaction with the service can be legally terminated by the client (though she does not need to);
(v) $g \subseteq S \times O \times S$ is the service's *transition relation*, which accounts for its state changes.

When $\langle s, \circ, s \rangle \in \rho$, we say that transition $s \xrightarrow{\circ} s'$ is in S. Given a state $s \in S$, if there exists a transition $s \xrightarrow{\circ} s'$ in S, then operation o is said to be executable in s. A transition $s \xrightarrow{\circ} s'$ in S denotes that s' is a possible successor state of s, when operation o is executed in s.

We see that when executing a given operation in a given state, there may be two different transitions systems possible as results, which are describe as follows:

- A service S is *deterministic* if there are no two distinct transitions $s \xrightarrow{\circ} s'$ and $s \xrightarrow{\circ} s''$ such that $s' \neq s''$. Notice that given a deterministic service's state and an executable operation in that state, unique next service's state is always known. That is, deterministic services are indeed fully controllable by just selecting the operation to perform next. TS_1 is deterministic and models the case in which after operation a one can perform both b and c.
- TS_2 A service S is non-deterministic, when executing a given operation in a given state several transition can take place. So, when choosing the operation to execute next, the client of the service cannot be certain of which choices will be available later on, this depending on which transition actually takes place. In other words, non-deterministic services are only partially controllable. TS_2 is non-deterministic and models the case in which after operation a, one is allowed to perform either b or c, depending on the actual transition that takes place after executing a.

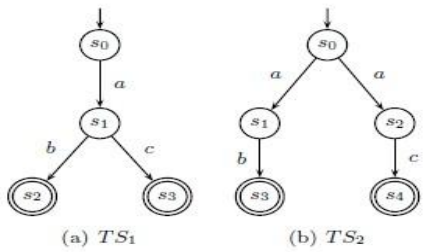

Fig. 1 Two different transition systems

As it turns out, finite state machine (and language theory) non-determinism is *angelic* and becomes just a compact way to represent the set of accepted operation sequences. On the other hand, Transition System in non-determinism is devilish, meaning that the client can ask for operation execution but the actual transition is chosen (in a devilish manner) by the transition system. As anticipated, we follow the original proposal of the Roman model and focus on deterministic transition systems only.

Available Services: the software artifact which is directly available to the client is called available services. They are defined once for all and develop gradually according to their behavior. The only thing one can do with them is to control their gradual development by instructing them to execute legal operation sequences. Most of the time, there are many $s_i (i = 1,...,n)$ and each of them has a transition system $s_i = \langle O_i, S_i, s_{io}, \delta_i, S_i^f \rangle$.

Service Community: is formally characterized by:
(i) A finite common set of actions, called the alphabet of the community;
(ii) A set of services specified in terms of the common set of actions.

Therefore a service needs to export its common set of actions to service community. A service community can delegate the execution of some or all its actions to other service instances in the community that is called the added value of the community of services or service composite.

Target service: is generated by the community. Its execution is a complete delegated action to other members of the community. Its generation is made by suitably composing parts of services instances in the community. The target service is coherent with the virtual service and it is also deterministic. The target service is defined as a transition system as follows $TS_t = (S_t, s_{t0}, G_t, \delta_t, F_t)$, and we have to notice that it does not exist in the service community and it has to be built by suitably combining parts of available services.

Orchestrator: In [3] the orchestrator is formally a function from (a) 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 (b) the operation currently chosen by the client, to the index *i* of the service 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 is able to delegate every operation executable by the target to one of the available services. This certainly means that an orchestrator is a system component that could activate, stop, and resume any of the available services, and to order them to perform an operation among those which are executable in their state. The orchestrator is the engine of the composition mechanism, it has full observability on available services states, at any step, will consider the operation chosen by the client (according to the target service) and delegate it to one of the services for which the operation is executable, and so on. It keeps tracking (at runtime) the availability of the current state during their interaction with the client to avoid any failure.

2.2 Composition Techniques

The aim of the service composition, in the Roman model, is to synthesize an orchestrator that can build the target service from the available service community. The specific composition problem has been addressed using different techniques.

Firstly, Berardi et al. proposed an automatic composition synthesis technique, in which the fundamental idea is to put the client request and some domain independent conditions into code by means of a specific description logics, and to reduce service composition problem to satisfiability by using Propositional Dynamic Logics (PDLs) [23–24, 27–28]. Notably, Logics of Programs are tightly related to Description Logics (DLs), for which highly optimized satisfiability checkers exist (e.g., RacerPro, Pellet, FACT, etc.). Berardi et al. [25] succeeded in building a single orchestrator by relying on the technique cited above to deal with non-deterministic finite state services. It is advocated by Fabio et al. [5] that this technique can only build finite state orchestrators, and it is actually made effective by a crucial result, showing that if an orchestrator exists then there exists one which is finite [25]. The conceptual schema of PDL-based approach to service composition is described by the following steps:
(i) The Roman model is used to describe the problem instance where the services are modeled as a finite state machine and then as transition system.
(ii) In the generated abstract PDL formula, each finite state corresponds to a finite state orchestrator as a solution to the original problem, vice versa; each composition problem's finite state solution has a corresponding model of the PDL formula.
(iii) In this phase the generated abstract PDL formula is encoded into DL knowledge based.
(iv) DL Reasoner uses this encoding for suitably generating a model of knowledge base, provided it is consistent.

The generated model corresponds to a model of the original PDL formula, which also correspond to a composition problem's solution. A tool was developed to

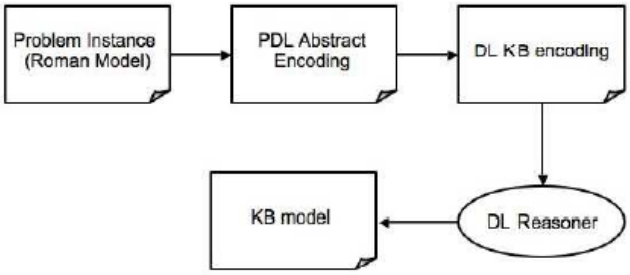

Fig. 2 Conceptual Schema of PDL-based approach to service composition

support this conceptual model in [25].

- More recently [23], the problem has drawn favorable attention and was approach by the techniques of Linear Time Logic (LTL) synthesis [35], based on model checking of game structures for the so called safety games (see also ATL [41–42]).
- Another approach recently proposed is based on directly computing compositions by exploiting (variants of) the formal notion of simulation relation between transition systems [5, 22, 43].

The two latter approaches promise both a high level of scalability, since in practice they can be based on symbolic model checking technologies. In [5] they do not use pure finite state machine to model service, instead they proposed a generic transition system which are suitable for such simulation model, capable of dealing with non-deterministic communities. Basically simulation based solutions are finite structures that represent all possible and even infinite state orchestrators that realize a target service called composition generators. The observation shows that the composition problem is proven EXPTIME-complete. A conceptual schema of such an approach is depicted in [5] as synthesis engine are available, they proposed a translation module that implements a procedure for automatic reduction of a service composition instances into a game structure.

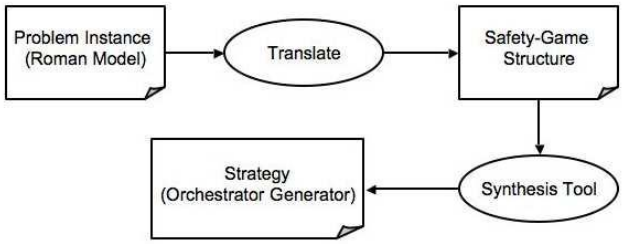

Fig. 3 Conceptual Schema of the Game-based approach to service Composition.

3. Extensions and Variants of the Roman Model

The success of the service composition technique in [25] base on the reduction to satisfiability in PDL; is the fact that PDL satisfiability shares the same basic algorithm behind the success of the description logics based reasoning systems (Fact, Racer, Pellet) used for OWL, and since the applicability of these reasoning system in the context composition it appears to be quite promising [13]. Thus, many extension and variants has been proposed to improve the original technique of composition, such as the following:

3.1 Forms of Target Service's Loose Specifications: (or Non-deterministic (angelic) target specification) [31]

The author in [31] proposed a method of automatic composition synthesis of service by representing the service behavior as finite state machine, based on PDL, under the assumption of a possibly incomplete specification of the sequences of actions and a set of available services. The authors followed the approach in [6], upon which they build their approach by introducing two fundamental extensions:

(i) The composition is not only based on controlling the concurrent execution of the available component services, but also it allows the synchronization and communication between the component services. They introduce the notion of initiator and servant, and work under the assumption that each action involves one initiator and one or more servants that suitably synchronize and exchange information in order to complete the action. The composition can control who is interacting at each step and allows two component services to interact and synchronize suitably before starting to serve the client, or while serving it.

(ii) The client request is a specification of transition system that the client is interested in being able to execute. They present several form of under-specification of such a transition system:
   - By introducing forms which do not care either there is non-determinism (angelic non-determinism) on the next set of transitions available to the client or there isn't; it mean that the client lets the composition synthesizes to resolve non-deterministic choices by taking advantage of what the available component services can do at that point of their computation;
   - This has to be contrasted with the fact that at the same time the composition synthesis must generate a composition that allows the client to make all choices specified in its transition system.

- And by letting the activities in which the client is involved to be interleaved in specified point with activities that are performed by the component service without the client intervention (but of which the client is in any case aware); allows the client (a) to exploit the synchronization and communication abilities that the component services have, and (b) to allow such service to perform some preliminary/extra work before or while serving it.

The author's main result is a composition synthesis technique, which supposes that a composition of the available component service realizing the client specification exists, and then such a technique will actually produce one such composition. Since the result produced is FSM, based on the collateral result of their synthesis technique, they demonstrate that if composition exists then the existence is in the finite state. They solve the problem as EXPTIME-Hard. The synthesis technique is based on reducing the problem of checking the existence of a composition into checking satisfability of a formula expressed in variant of PDL [23], equipped with graded modalities [14, 16, 20, 44]. Interestingly such logic corresponds to a particular expressive DL, namely ALCQreg, which is well-studied from the computational point of view (see, e.g., [7] in [10]). This correspondence allows them in principle, to exploit the highly optimized DL-based reasoning systems, currently available [10, 17–19].

3.2 Look-ahead:

To address automated composition problem, the look-ahead technique was firstly adopted in [11] to extend the Roman model where only regular activities are considered because the activities are modeled by finite state automata. To this purpose, the authors introduce the notion of delegator that can settle the assignment according to entire sequence of activities, check the existence of the mediator in EXPTIME complexity, and when any of the available delegator can simulate the target service the k look-ahead delegator solution technique is proposed for building the delegator which can do the right delegations, since the delegator informs about the client's immediate choice and its future choice in next move. They also show the existence of a strict hierarchy of k look-ahead delegation problem.

Instead of considering regular activities under which activity models are finite automata [11], author proposed a framework in which more complex and non-regular activity sequence are possible. In [26] the automata theoretic techniques use are different from the techniques used in [11]. Reference [26] firstly approach composability issue, in [11], it was shown that composability is decidable for a system $(A; A_1, ..., A_r)$ of deterministic finite automata (DFA). [26] Generalizes this result to the case when A is an NPCM (non-deterministic pushdown automaton with reversal-bounded counters) and the A_i's are DFAs. In contrast, [26] shows that it is undecidable to determine, given DFAs A and A_1, and a deterministic reversal-bounded counter-machine (DCM) A_2 with only one 1-reversal counter (i.e. once the counter decrements it can no longer increment), whether is composable. Secondly, follows the approach in [11] for providing the k look-ahead delegator for infinite state automata that can check the existence of deterministic delegator within some resource bound. The delegator does not need to look back to its delegation history to decide where the current activity shall be delegated. For a positive integer k, a k delegator for $(A; A_1, ..., A_r)$ is a deterministic reversal-bounded counter-machine D which, knowing (a) the current state of $A; A_1, ..., A_r$ and the signs of their counter (zero or non zero), and (b) the k-look-ahead symbols (the k future activities) to the right of the current input symbol being processed, can deterministically determine the A_i to assign the current symbol. In addition, every string w delegated accepted by A is also accepted by D, which imply that the subsequence of string w delegated by D to each A_i is accepted by A_i. In other words, if a system $(A; A_1, ..., A_r)$ has a k-delegator for some k, then it must be composable.

3.3 Security and Trust-aware Services Composition [13]:

The authors tackle the automatic composition problem in the presence of component services that have access control and authorization constraints, and impose further reputation constraints on other component services. We are in trust community, where different component services may either have trust or not for others. To enhance this model in secure manner, the authors provide an access control model based on credentials which restrict the set of the client and subjects that can invoke service's operation. Credentials are signed assertions describing properties of a subject that are used to establish trust between two unknown communicating parties before allowing access to information or services.

The behavior of the available services is considered to be non-deterministic and not fully controllable by the orchestrator. In addition, the security constraint is imposed to control the access, authorization and reputation. The model used is based on reduction to satisfiability in PDL [23] with a limited use of the reflexive-transitive-closure operator. Now, PDL satisfability shares the same basic algorithms, which are also behind the success of the description logics-based reasoning systems used for OWL2, such as FaCT3, Racer4, Pellet5, and hence its applicability in the context of composition synthesis appears to be quite promising.

The framework is formally define as in [24, 31, 9], but also added novel notion such as reputation matrix *Rep* which has rows available services and columns available services and possibly third parties. The cell *Rep* (i, j) represents the reputation level (set of all possible levels are finite) that the available service S_i has on the available services S_j or on the third party P_{j-n} . In addition, a set of credentials is defined to let the client has various part of an available service to execute.

Credential: is the trust relation between client and service provider. Formally let $C = \{c_1, ..., c_m\}$ be the set of credentials that are associated to clients. Each c_h is a pair of variable $(Attr, Issuer)$ where $Attr$ is the attribute variable of the credential, whose value characterizes the client and $Issuer$ is the issuer variable that contains the name of the entity that issued the value for the attribute variable. Δ is the finite domain. $I = \{1, ..., n, n+1, ..., n+l\}$, where $1, ..., n$ are identifiers of available services and $n+1, ..., n+l$ are identifiers of third parties $P_1, ...P_l$.

Available Services: are programs which provide client with a choice of available actions; the client selects one of them, the action is executed; and so on. Available services use credentials in order to decide which actions at each point of their execution are actually available to the client executing it (i.e. the client is authorized to execute the action).

3.4 Distributed Orchestrator

In [21] the available behaviors are partially controllable, and a controller is design to coordinate available behavior for realizing target behavior. The authors claimed that often a centralized orchestration is unrealistic: e.g. services deployed on mobile devices are;

- Too tight coordination
- Too much communication
- Orchestrator cannot be embodied anywhere

The authors drop centralized orchestrator in favor of independent controllers on single available services (exchanging messages). Under suitable conditions, a distributed orchestrator exists if only if a centralized one does. And then demonstrate that the EXPTIME-complete is still usable.

3.5 Shared Environments or Other Infrastructure for Communication among Services

The techniques for solving composition problem presented in [15] is not only applied to more realistic scenario, but also show how a workflow done by a team of cooperating agents, is realized as result of coordination, or more precisely orchestration of several behaviors which provide high-level descriptions of agents' capabilities. The main technical results in this paper demonstrate that there is an existence of a sound, complete and terminating procedure for computing a distributed orchestrator $X = (O_1, ..., O_n)$ that realizes a workflow W over a $WfSK$ κ relative to service $S_1, ..., S_n$ over κ and blackboard state γ_o . Moreover each local orchestrator O_i returned by such a procedure is finite state and require a finite number of messages (more precisely message types).

In [12] the composition technique proposed a model that allows dynamic and finite-state data structure representation in certain cases; they first modeled the problem in an abstract framework based on the formal definition. Secondly, they develop new method for performing automatic synthesis of the fully controllable module. The setting used in this framework is made by the following part:

- **A shared environment** structured by a finite set of shared actions, a finite set of possible environment states, an initial state of the environment and the transition relation among states. It is also non-deterministic.
- **A behavior** according to the shared environment defined on top, is non-deterministic and characterized by a finite set of behavior states; an initial state of the behavior; a set of guards; the behavior transition relation; and the set of final states of the behavior.
- **Runs and traces** where run is a possibly alternating sequence of behavior over shared environment; and trace is a sequence of pair actions guided by the behavior.

- **The system** is formed by the observable environment and the available behaviors.
- **The problem** raised a solution technique for building an orchestrator that realizes the target behavior if it realizes all its traces.

After all, by mean of an example, the authors used the technique based on reduction to satisfability in PDL [28], with a limited use of the reflexive-transitive-closure operator, to show that the solution technique developed is sound and terminating optimal with respect to computational complexity.

3.6 Data-aware Services

The service should give us the property that has the ability to manage data: in reality, services deal with data. The service describe in [28], is characterize by four components:

(i) *Real world state*, which is database instance over a relational database schema.
(ii) *Atomic process* is the functionalities or the operations that services are capable of doing, such as access and modification of the database, and also conditional effects. The services community is composed of web services, clients and any other participant of this community have to share the same ontology.
(iii) *Message passing behavior* is composed of send and receive message by the web service from the community. It is much more about the message types (classes) than message contents.
(iv) *The behavior of the web service* is composed of multiple atomic processes and message passing activities. Guarded automata are used in this framework. Guarded automaton is a finite state machine, such that from one transition to another can be clearly defined. A transition moves to the next stage only if its condition is evaluated to be true.

There are four kinds of web services defined in this framework, called "Colombo", they all belong to the same community and modeled by using guarded automata:

(i) *Non-Client Web Service* are well described services published in UDDI registry, capable of performing functionalities and operations.
(ii) *Client Web Services* is a behavior, which represents the interaction (send and receive) between client and the web services invocated by the client. Note that the client behavior is non-deterministic in term of actions made and choices selected by the client. Then guarded automata will only conceive two states, which are "ReadyToTransmit" and "ReadyToReceive". The client choice will be switching between the two states until it ends.
(iii) *Goal Service* is the desired behavior to realize. It is also specified as a guarded automaton in terms of alphabet of atomic processes *O*.
(iv) *Mediator Service* in Colombo framework used the topological approach for composition. This approach has a virtual service also called Mediator which is responsible of controlling data flow and control flow among participant services. Its behavior simulates the behavior of the goal service. The mediator service represents the composition synthesis specification which should be orchestrated to fulfill client request, it represents the expected output from the Colombo automatic composition algorithm.

In this framework each non-client and mediator web services instance possess includes the followings:

(i) A Local Store (LStore) is a database table that is used to store parameters values of incoming messages and output messages, and to populate parameters of outgoing messages and input parameters to atomic processes. The conditional branching of web services behavior at any time is based on the values stored in its LStore at this time.
(ii) A port for each incoming and outgoing message to let web services communicate among themselves.
(iii) A Queue Store (QStore) for each incoming message.

The work in [28] has proposed a new solution for automatic service composition algorithm in the presence of data.

3.7 Artificial Intelligence Planning

In [29], a novel framework is built for automated composition of web services based on planning method in asynchronous domains. In this frame work, BPEL4WS concrete process is automatically generated from a given set of BPEL4WS abstract specification of published web service and given a composition requirement; the generated BPEL4WS process can interact asynchronously with the published services. The deployment and the execution of the generated BPEL4WS are characterized by the following steps:

(i) The BPEL4WS abstract process is defined by transition system which is capable of communicating by asynchronous input/output actions (published protocol) or by means of internal actions (internal behavior not visible to external parties).
(ii) Within asynchronous conversation, the input queue mechanism is modeled in such a way that a process can immediately receive a message or after an internal action, which prevent the message being lost.
(iii) Under this modeling supposition a novel method planning is developed in asynchronous domain for

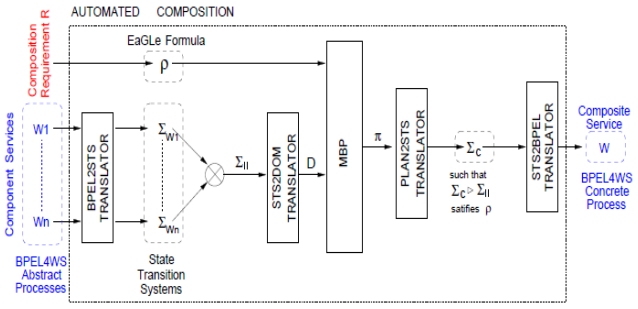

Fig. 4 Approach

generating executable and deployable BPEL4WS code that is depicted in Figure 4.

4. Conclusions and Future work

Generally there are differences between the approach in which the interaction between services and their clients is modeled through actions, and the approach that can be found in standard languages such as WSDL [23] where the focus is on exchanged messages. For example, in WSDL, an interaction between the service and the client is modeled by an operation, say search by author with a message that the client sends to the service for requesting a search say search by author request, and a message that the service sends back to the client (and, in his turn, the client receives), containing the results of the computation, say search by author response. Hence, each WSDL operation roughly corresponds to an action in our framework.

Formally, the advantages brought by the Roman model approach are quite important for the following reason: (a) the developed framework, abstracts enough the conversation human-machine, so that it can be considered as conceptual model for several classes of scenarios, which make this theoretical technique applicable to much more context such as web services composition, multi-agent system, etc. (b) It consider stateful services which impose some constraints on the possible sequences of operations (a.k.a., conversations) that a client can engage with the service. Composing stateful services poses additional challenges, as the composite service should be corrected with respect to the possible conversations allowed by the component ones. We have to say that services are just the high-level descriptions of software artifacts, especially when we deal with a behavioral model. In fact, services are characterized by states and state transition triggered by inputs, which represent requested operations. From the interpretation, it is shown that service-runs are regarded as computation fragments, which can generate more complex services through combining.

In [5] it is advocated that service composer developed in [25] based on the original approach, can synthesize an orchestrator that realizes the target services, but brought three major shortcoming:(a) only finite-state orchestrators are returned; (b) the obtained solution is not *flexible*, that is if a solution has been built which relies on an available service and such a service becomes unavailable at runtime, then the solution is no longer valid and the best one can do using this approach is to re-compute a new solution; (c) on the practical side, due to implemented DL reasoner limitations, ESC is actually able to synthesize a model only for some particular inputs, though it is complete with respect to checking for the existence of a model. [30] Point out that one of this approach's problems is that it doesn't scale well (needs EXPTIME).

To overcome the problems cited above, recently novel techniques have been developed that are more flexible and more scalable, based on the formal notion of simulation [6, 23, 29] and the Linear Time Logic (LTL) synthesis [26], based on model checking of game structures for the so called safety games (see also ATL [2–3]). Both these two technique are based on symbolic model checking technologies, which an explication to their high level of scalability.

Despite all the efforts which have been made in this field, it still requires much more attention for solving the raised problems which have not yet been completely fixed, and can constitute an interesting research area. In [31], here are presented the kind of angelic non-deterministic of the target specification of the client, meaning that the client specifies (a) the actions for which he is the initiator, and (b) the possibility of having activities in which the client himself is not involved, also called silent actions, in this case the orchestrator could be unable to satisfy the client request. Figure 5(a) represents the target service and the Figure 5(b) the community service. It is supposed that both services start in their initial state. If the service S_a execute a and move from S_1 to S_2, while S_b also will execute a and move from S_3 to S_4. From S_4 the service S_b cannot execute b or c. From this, it is clearer that the community services S_b

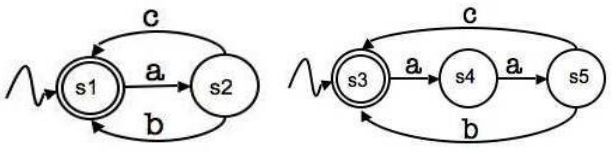

Fig. 5 (a) Target Service S_a ; (b) Community Service S_b .

cannot simulate the target service S_a . But one can slightly plan its evolution for S_b being able to perfectly simulate S_a . One can use the planning technique for reachability, where the orchestrator aim at executing a plan so to lead the community, from current state, to a desired state which simulates current target's one. When the goal is reached, then one can, through the plan of the orchestrator, compute the target service current state by simulation. This will improve the community capability, by increasing the set of target services actually realizable.

In the literature, some earlier work have point out the necessity to enables the data-management ability for services. In fact, services are more concern about sending and receiving data from one to another to activate or accomplish their task, according to their state. Interest of transaction-based data management systems is highlighted when web services are developed to access and filter data [32]. A model based on Mealy machine is proposed in which conversation is guarded (guided) according to a predefined set of channels [33–34]. Methodology is presented that show to synthesize web services as Mealy machines whose conversations (across a given set of channels) are compliant with a given specification. In [34] an extension of the framework is developed where services are specified as guarded automata, having local XML variables in order to deal with data semantics. A transition system method for modeling web services communicating through messaging and model checking techniques is used to compose the services in the presence of some limited support of data [12]. The used of the technique in [12] for finitely handling data ranging from infinite domain to their framework, in order to provide an extension to it. The difficulty comes from the presence of data which will ultimately derive to infinite state system verification and synthesis. It becomes a hard task for non-trivial properties and also undecidable for general ones. Adding the possibility of dealing with data in the services composition framework will be a great improvement.

We observe that [28] tackles the composition problem by relying on PDL-based approach. However, under the same model one can recast the problem in terms of (data-aware) simulation, which is defining a relation between two data-aware services that interact with a common underlying data structure, whose data content may come from an infinite domain. This way, one would get the advantages brought by a simulation-based approach, though the actual resolution would be more complex due to state space infiniteness, which calls for some abstraction procedure. Finite state systems are capable of producing a strong effect on behavioral model for service, which allow them to enhance composition problems, at the same time giving prove that there are complete solution approaches available.

Acknowledgments

The work reported in this paper was supported by Grant No: 61070182 and No: 61272508
The inclusion of images and examples from external sources is only for non-commercial educational purposes, and their use is hereby acknowledged.

References

[1] M. Aiello, M. P. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serani, and P. Traverso, "A request language for web-services based on planning and constraint satisfaction", in Proc. of the Third Intl. Workshop on Technologies for E-Services (TES '02), 2002, pp. 76–85.

[2] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara, "DAML-S: Web service description for the semantic web", in Proc. of the First Intl. Semantic Web Conf. on The Semantic Web (ISWC '02), 2002, pp. 348–363.

[3] D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi, "Automatic Service Composition and Synthesis: the Roman Model", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 31, No. 3, 2008, pp. 18–22.

[4] S. McIlraith, and T. Son, "Adapting Golog for composition of semantic web services", in Proc. of the 8th Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR 2002), 2002, pp. 482–493.

[5] F. Patrizi, "An Introduction to simulation-based Techniques for Automated Service Composition", in Proceeding of Fourth European Young Researchers Workshop on Service Oriented Computing (YR-SOC 2009), 2009, pp.37–49.

[6] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of e-Services that Export their Behavior", in Proc. of 1st Intl. Conf. on Service Oriented Computing (ICSOC), 2003, pp. 43–58.

[7] D. Calvanese, and G. De Giacomo, "Expressive description logics", in Baader et al., chapter 5, pages 178–229.

[8] J. Yang, and M. Papazoglou, "Web components: A substrate for web service reuse and composition", in Proc. of the 14th Intl. Conf. on Advanced Information Systems Engineering (CAiSE 2002), 2002, pp. 21–36.

[9] D. Berardi, D. Calvanese, G. De Giacomo, and M. Mecella. Automatic Composition of Web Services with Nondeterministic Behavior. Technical Report TR-05-2006, Univ. Roma LA SAPIENZA, Dipartimento di Informatica e Sistemistica, 2006. Extended abstracts/short papers in Proc. ICSOC 2005 and in Proc. ICWS 2006.

[10] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation and Applications, New York: Cambridge University Press, 2003.

[11] C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su, "Automated composition of service: Lookaheads", in Proc. of the 2nd Intl. Conf. on Service Oriented Computing (ICSOC '04), 2004, pp. 252–262.

[12] P. Traverso, and M. Pistore, "Automated Composition of Semantic Web Services into Executable Processes", in Proc. of Intl. Semantic Web Conference (ISWC 2004), 2004, pp. 380–394.

[13] F. Cheikh, G. De Giacomo, and M. Mecella, "Automatic web services composition in trust-aware communities", in Proc. of the 3rd ACM workshop on Secure Web Services (SWS), 2006, 43–52.

[14] M. Fattorosi-Barnaba, and F. De Caro, "Graded modalities. I", Studia Logica, Vol. 44, No. 2, 1985, pp. 197–221.

[15] G. De Giacomo, M. de Leoni, M. Mecella, and F. Patrizi, "Automatic workflows composition of mobile services", in Proc. of IEEE Intl. Conf. on Web Services (ICWS 2007), 2007, pp.823–830.

[16] K. Fine, In so many possible worlds, "Notre Dame Journal of Formal Logic", 13(4):516–520, 3072

[17] V. Haarslev and R. Moller, "RACER system description", in Proc of (IJCAR 2001), volume 2083 of LNAI, pages 701–705. Springer-Verlag, 2001.

[18] I. Horrocks, "The FaCT system", in Proc. Of (TABLEAUX'98), volume 1397 of LNAI, pages 307–312. Springer-Verlag, 3098.

[19] R. Moller and V. Haarslev, "Description logic systems", In Baader et al. [6], chapter 8, pages 282–305.

[20] W. Van der Hoek, "On the semantics of graded modalities" Journal of Applied Non-Classical Logics, 2(1):81–323, 3092.

[21] S. Sardina, F. Patrizi, and G. De Giacomo, "Automatic synthesis of a global behavior from multiple distributed behaviors", in Proc. of AAAI 2007.

[22] D. Berardi, F Cheikh, D. De Giacomo, and F. Patrizi, "Automatic service composition via simulation", Intl. Journal of Foundations of Computer Science 30, 2 (2008), 429–451.

[23] D. Harel, D. Kozen, and J. Tiuryn, "Dynamic Logic" The MIT Press, 2000.

[24] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic service composition based on behavioural descriptions", Intl. Journal of Cooperative Information Systems 14, 4 (2005), 333–376.

[25] D. Berardi (2005): "Automatic Service Composition: Models, Techniques and Tools". Ph.D. thesis, SAPIENZA Universita degli Studi di Roma.

[26] Z. Dang, O. H. Ibarra, and J. Su, "On composition and look-ahead delegation of service modeled by automata" Theor. Comput. Sci., 341(1–3):344–363, 2005.

[27] G. De Giacomo and S. Sardina. "Automatic synthesis of new behaviors from a library of available behaviors" in Proc. of IJCAI 2007.

[28] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. "Automatic composition of transition based semantic web services with messaging". in Proc. of VLDB 2005.

[29] M. Pistore, P. Traverso, and P. Bertoli. "Automated composition of web services by planning in asynchronous domains" in Proc. of ICAPS 2005.

[30] U. KÄuster, M. Stern, and B. KÄonig-Ries, "A classi¯cation of issures and approaches in service composition", In Proceedings of the First International Workshop on Engineering Service Compositions (WESC05), Amsterdam, Netherlands, December 2005.

[31] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Synthesis of underspecified composite service based on automated reasoning. in Proc. of ICSOC 2004.

[32] P. Helland, "Data on the outside versus data on the inside" In CIDR, pages 144–83, 2005.

[33] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition" in Proc. of WWW 2003.

[34] X. Fu, T. Bultan, and J. Su. "Analysis of Interacting BPEL Web Services" in Proc. of WWW 2004.

[35] N. Piterman, A. Pnueli, and Y. Sa'ar "Synthesis of reactive designs" in Proc. of VMCAI 2006.

[36] R. Hull, "Web Services Composition: A Story of Models, Automata, and Logics", In: 2005 IEEE International Conference on Services (SCC 2005).

[37] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, "Web Services: Concepts, Architectures and Applications" Springer, 2004

[38] R. Hull, M. Benedikt, V. Christophides, and J. Su, "E-Services: A Look behind the Curtain", in Proc. Of the PODS 2003 Conf..

[39] T. Pilioura and A. Tsalgatidou. "E-Services: Current Technologies and Open Issues", in Proc. of VLDB-TES 2001.

[40] M. Papazoglou and D. Georgakopoulos, "Service Oriented Computing (Special Issue)" Communications of the ACM, 46(10), October 2003.

[41] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic", Journal of the ACM, 49(5):672–713, 2002.

[42] R. Alur, T. A. Henzinger, F. Y. C. Mang, S. Qadeer, S. K. Rajamani, and S. Tasiran, "MOCHA: Modularity in model checking", in Proc. of CAV 1998.

[43] S. Sardina, F. Patrizi, and G. De Giacomo. "Behavior composition in the presence of failure" in Proc. Of KR 2008.

[44] S. Ghandeharizadeh, C. A. Knoblock, C. Papadopoulos, C. Shahabi, E. Alwagait, J. L. Ambite, M. Cai, C. Chen, P. Pol, R. R. Schmidt, S. Song, S. Thakkar, and R. Zhou, "Proteus: A System for Dynamically Composing and Intelligently Executing Web Services", in Proc. of ICWS 2003.