Synthesis of Underspecified Composite *e*-Services based on Automated Reasoning

Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
<lastname>@dis.uniroma1.it

Diego Calvanese
Libera Università di Bolzano/Bozen
Facoltà di Scienze e Tecnologie Informatiche
Piazza Domenicani 3, 39100 Bolzano, Italy
calvanese@inf.unibz.it

ABSTRACT

In this paper we study automatic composition synthesis of e-Services, based on automated reasoning. We represent the behavior of an e-Service in terms of a deterministic transition system (or a finite state machine), in which for each action the role of the e-Service, either as initiator or as servant, is highlighted. In this setting we present an algorithm based on satisfiability in a variant of Propositional Dynamic Logic that solves the automatic composition problem. Specifically, given (i) a possibly incomplete specification of the sequences of actions that a client would like to realize, and (ii) a set of available e-Services, our technique synthesizes a composite e-Service that (i) uses only the available e-Services and (ii) interacts with the client "in accordance" to the given specification. We also study the computational complexity of the proposed algorithm.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program synthesis*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*State diagrams*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; F.3.1 [**Logics and Meaning of Programs**]: Specifying and Verifying and Reasoning about Programs—*Logics of programs*

General Terms

Theory, Verification, Design

Keywords

Theoretical Framework for Service Representation and Compositions, Service Composition Models and Language, Intelligent Services

1. INTRODUCTION

The Service Oriented Computing (SOC [33]) paradigm offers strong potential for a revolutionary change in the way the network technology is exploited. It allows for realizing the so-called *virtual enterprises* and communities [18, 13], i.e., a pool of companies that are able to export services as semantically defined functionalities to a vast number of customers, and to cooperate by automatically composing and integrating services over a distributed network. Such services, usually referred to as e-Services, are self-contained, modular applications that can be described, published, located and dynamically invoked, in a programming language independent way.

Research on e-Services spans over many interesting issues, including description, discovery, composition, synchronization, coordination, and verification [25]. In [34], the Service Oriented Architecture (SOA) is proposed, which is the commonly accepted and minimal architecture for e-Services. SOA provides the basic operations necessary to describe, publish, find and invoke e-Services. One of the main issues in SOC is e-Service composition [33]. e-Service *composition* addresses the situation when a client request cannot be satisfied by any available e-Service, but by a *composition* of them. In other words, the client request can only be satisfied by suitably combining "parts of" available e-Services, also called *component* e-Services in this context. Composition involves two different issues [17]. The first, typically called *composition synthesis*, is concerned with synthesizing a composition of available e-Services that satisfies a client request. The synthesis process produces a specification of how to coordinate, or *orchestrate*, the component e-Services to fulfill the 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 e-Services produced by the composition synthesis, by suitably supervising and monitoring both the control flow and the data flow among the involved e-Services.

In this paper, we deal with composition synthesis, by following the approach of [6]. That approach has two notable features on which we build here:

- The composition is based on the ability of executing the available component e-Services concurrently, and of controlling in a suitable way how such services are interleaved to serve the client.

- The client request is not a specification of a desired execution, but a set of possibly non terminating executions organized in an execution tree, whose nodes correspond to sequences of transitions executed so far and whose successor nodes represent the choices available to the client to choose from what to do next. In other words the client specifies the so-called transition system of the activities she is interested in doing.

Observe that both of these features are quite distinctive of that approach, and set the stage for a quite advanced form of composition.

Here we follow that approach, but introduce two fundamental extensions.

1. The composition is again based on controlling the concurrent execution of the the available component e-Services, but in addition it allows for *synchronization and communication* between the component e-Services. These aspects are completely missing in [6]. Here instead, we introduce the notion of *initiator* and *servant* of an (inter)action, and we require 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 e-Services to interact and synchronize suitably before starting to serve the client, or while serving it/him. This provides us with a bridge towards the message-based model of e-Services [9], and towards the e-Service communication model that form the basis to standard languages, such as BPEL4WS [2].

2. The client request is again a specification of the transition system that the client is interested in being able to execute. However here we allow for several forms of underspecification of such a transition system

    - by introducing forms of don't care nondeterminism (angelic nondeterminism) on the next set of transitions available to the client; that is the client allows the composition synthesis to resolve nondeterministic choices taking advantage of what the available component e-Services can do at that point of their computation[1];

    - and by allowing the activities in which the client is involved (i.e., those described by its transition system) to be interleaved in specified points with activities that are performed by the component e-Services without the client intervention (but of which the client is in any case aware); this allows the client (i) to exploit the synchronization and communication abilities that the component e-Services have (cf. point 1 above), and (ii) to allow such e-Services to perform some preliminary/extra work before or while serving it/him.

[1]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.

We study the problem of e-Service composition in this enhanced framework. Our main result is a *composition synthesis technique*, which is *sound*, *complete* and *terminating*, i.e., if a composition of the available component e-Services realizing the client specification exists, then such a technique will actually produce one such a composition. The composition produced is finite state, and hence, as a collateral result of our synthesis technique, we show that if a composition exists then there exists one which is indeed finite state. Also, our technique give us an EXPTIME upper bound in worst-case computational complexity for the composition synthesis problem. While assessing that such bound is in fact tight is still open, we conjecture that the problem is indeed EXPTIME-hard. From a more practical point of view, it is easy to find cases in which the composition must be exponential in the size of the component e-Services and the client specification, hence exponentiality is inherent to the problem.[2]

The synthesis technique is based on reducing the problem of checking the existence of a composition into checking satisfiability of a formula expressed in variant of Propositional Dynamic Logic (PDL [23]), equipped with graded modalities [14, 16, 38]. Interestingly such a logic corresponds to a particular expressive Description Logic, namely $\mathcal{ALCQ}_{reg}$, which is well-studied from the computational point of view (see, e.g., [10] in [4]). This correspondence allows us, in principle, to exploit the highly optimized DL-based reasoning systems, currently available [24, 21, 32].

The rest of this paper is organized as follows. In Section 2 we define our formal framework for e-Services. In Section 3 we present the notions of client specification and composition. In Section 4 we present our composition synthesis technique. In Section 5 we relate our work with other existing approaches to e-Service composition. Finally, in Section 6 we draw some conclusions and discuss future work.

2. *E*-SERVICE MODEL

An e-Service is a software artifact that interacts with its client and possibly other e-Services in order to perform a specified task. A client can be either a human or a software application. When executed, an e-Service performs a given task by executing certain actions in coordination with the client or other e-Services. Specifically, each action in the task has a (single) *initiator*, typically the client, which requests the execution of the action possibly passing along information, and one or more *servants*, which are e-Services that respond to the request, possibly exchanging with the initiator further information.

We build on the approach of [6, 7], and characterize the exported behavior of an e-Service by means of an *execution tree*. The nodes of such a tree represent the sequence of actions that have been performed so far by the e-Service, while the successor nodes represent the actions that can be performed next at each point of the computation. The root represents the initial state of the computation performed by the e-Service, when no action has been executed yet. We label the nodes that correspond to completed execution of the e-Service as "final", with the intended meaning that in these nodes the e-Service can (legally) terminate. In order to represent the role an e-Service has wrt a given action, we

[2]Obviously, this does not give us a tight lower bound result, since the problem could be, for example, PSPACE-hard.

annotate each action symbol as follows: if the *e*-Service is one of the servants of an action a, then the action appears as $^{\gg}a$, conversely if the *e*-Service is the initiator of a, then the action appears as $a^{\gg}$. Observe that in such an execution tree, for each node we can have at most one successor node for each annotated action. That is, we assume that the state of the *e*-Service is entirely determined by the sequence of (annotated) actions executed so far.

To represent the set of *e*-Services available to a client, we introduce the notion of *community* $\mathcal{C}$ of *e*-Services, which is a (finite) set of *e*-Services that share a common (finite) set of actions Σ, also called the *alphabet* of the community. Hence, to join a community, an *e*-Service needs to export its behavior in terms of the alphabet of the community. Also, a client interacts with *e*-Services in $\mathcal{C}$ using the alphabet Σ.

In this work, we concentrate on *e*-Services whose behavior can be represented using a *finite number of states.* We do not consider any specific representation formalism for representing such states (such as action languages, situation calculus, state-charts, etc.). Instead, we use directly deterministic finite state machines, (or equivalently, finite deterministic transition systems)[3].

Formally, each *e*-Service in the community is described by a finite state machine (FSM) $A_i = (\Sigma^+, S_i, s_i^0, \delta_i, F_i)$, where:

- $\Sigma^+ = \{^{\gg}a, a^{\gg} \mid a \in \Sigma\}$ is the alphabet of the FSM;
- S_i is the set of states, representing the finite set of states of the *e*-Service;
- s_i^0 is the initial state, representing the initial state of the *e*-Service;
- $\delta_E : S_i \times \Sigma^+ \rightarrow S_i$ is the (partial) transition function, which is a partial function that given a state s and an annotated action $^{\gg}a$ (or $a^{\gg}$) returns the state resulting from executing the action in s;
- $F_i \subseteq S_i$ is the set of final states, representing the set of states that are final for the *e*-Service, i.e., the states where the *e*-Service can terminate.

Given an *e*-Service A_i, the execution tree $T(A_i)$ *generated* by A_i is the execution tree containing one node for each sequence of actions obtained by following (in any possible way) the transitions of A_i, and annotating as final those nodes corresponding to the traversal of final states.

The different roles that an *e*-Service can play with respect to a given action (i.e., either as initiator or as servant) induces a classification between the *e*-Services of a community:

- *pure-servant* *e*-Service: it is an *e*-Service that acts only as servant in all possible sequences of interactions it can be involved in; its associated FSM presents only actions of type $^{\gg}a$; such an *e*-Service can be directly exploited by a client, as it is able to completely satisfy client requests and execute its tasks (this is the kind of *e*-Services studied in [6]);
- *pure-initiator* *e*-Service: it is an *e*-Service that acts only as initiator in all possible sequences of interactions it can be involved in; its associated FSM presents only actions of type $a^{\gg}$; this kind of *e*-Service is, for example, the one typically associated to the client, i.e., it represents what the client wants to actually perform (c.f. [6]).
- *mixed* *e*-Service: it is an *e*-Service that acts as initiator in at least one possible sequence of interactions it can be involved in; its associated FSM presents at least one action of type $a^{\gg}$; such an *e*-Service can be exploited in a composition only if a matching *e*-Service that acts as servant can be found in the community, to which it can delegate the execution of actions it initiates.

[3]Note that the finite state machines are deterministic since they are used to compactly represent the execution tree of the corresponding *e*-Service, which in turn has at most one successor node for each (annotated) action.

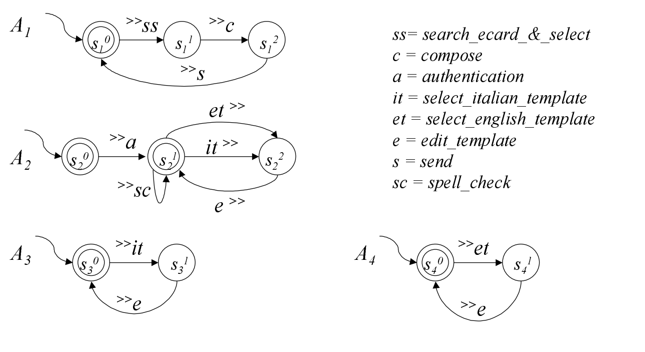

Figure 1: *e*-Services of the community

Observe that the fact that a client request is a specification of a pure-initiator *e*-Service correspond to the idea, almost universally accepted in the SOC literature, that the client cannot be exploited by the *e*-Services in the community for carrying on their tasks.

EXAMPLE 2.1. *Figure 1 shows a community of* e-*Services constituted by* e-*Services* A_1, A_2, A_3, A_4. A_1 *repeatedly allows for (*i*) searching an* e-*card and selecting one among those returned (*`search_ecard_&_select`*), for (*ii*) writing the* e-*card (*`compose`*), and for (*iii*) sending it (*`send`*);* A_1 *is servant for all these actions.* A_2 *provides functionalities to compose an* e-*card: (*i*) it validates the (registered) user information (e.g., name, email address, ...) received in input, or it registers a new user along with his information (*`authentication`*); then it repeatedly allows for either (*ii*) writing a message and performing a spell check (*`spell_check`*), or for selecting a template, which provides help in writing the message: such template can give suggestions (*iii*) in English (*`select_english_template`*), or (*iv*) in Italian (*`select_italian_template`*); finally, after one of such templates has been selected, it can be edited (*`edit_template`*).* A_2 *is servant for the* `authentication` *and* `spell_check` *operation and it is initiator for all the remaining operations.* A_3 *and* A_4 *act as servants for the operations for which* A_2 *is initiator:* A_3 *repeatedly allows for (*i*) selecting an English template and then for (*ii*) editing it;* A_4 *repeatedly allows for (*i*) selecting an Italian template and then for (*ii*) editing it.*

Finally, note that A_1, A_3 *and* A_4 *are pure-servant* e-*Services, whereas* A_2 *is a mixed* e-*Service.*

3. *E*-SERVICE COMPOSITION

When a client requests a certain service from an *e*-Service community, there may be no *e*-Service in the community that can deliver it directly. However, it may be possible to suitably orchestrate (i.e., coordinate the execution of) the e-Services of the community so as to provide the service requested by the client. In other words, there may be an orchestration that coordinates both the initiator and the servants of each action, using the e-Services in the community, and that realizes what requested by the client.

Formally, let the community $\mathcal{C}$ be formed by n e-Services $A_1, \ldots, A_n$, and let the service requested by the client be denoted by A_0. A composition O of the e-Services in $\mathcal{C}$ can be formalized as a so-called *composition tree* $\mathcal{T}(O)$.

- The root ε of the tree represents the fact that no action has been executed yet.
- Each node x in the composition tree $\mathcal{T}(O)$ represents the history up to now, i.e., the sequence of actions and their initiator as orchestrated so far.
- For every action a belonging to the alphabet Σ of the community and $I \in [0..n]$ [4] (0 stands for the client and $1, \ldots, n$ stand for the e-Services $A_1, \ldots, A_n$, respectively), $\mathcal{T}(O)$ contains at most one successor node $x \cdot (a, I)$.
- Some nodes of the composition tree are annotated as *final*: when a node is final, and only then, the orchestration can be stopped.
- Let's call a pair $(x, x \cdot (a, I))$ an *edge* of the tree. Each edge $(x, x \cdot (a, I))$ of $\mathcal{T}(O)$ is labeled by a triple (I, a, S), where a is the orchestrated action, $I \in [0..n]$ denotes the initiator, and $S \subseteq [1..n]$ denotes the nonempty set of e-Services in $\mathcal{C}$ that act as servants. As an example, the label $(0, a, \{1, 3\})$ means that the action a is initiated by the client and served by the e-Services A_1 and A_3.

Given a composition tree $\mathcal{T}(O)$ and a path p (i.e., a sequence of edges) in $\mathcal{T}(O)$ starting from the root and arriving to a node x in $\mathcal{T}(O)$, we call the *projection* of p on an e-Service A_i the sequence of (annotated) actions obtained from p as follows:

1. we remove from p all edges whose label (I, a, S) is such that $i \notin \{I\} \cup S$
2. in the resulting sequence, we replace
   (a) by $a^{\gg}$, each edge labeled by (I, a, S) where $I = i$;
   (b) by ${}^{\gg}a$, each edge labeled by (I, a, S) where $i \in S$.

Intuitively, point 1 above throws away all edges where e-Service A_i is neither servant nor initiator for action a; points 2(a) and 2(b) deal with edges where A_i is, respectively, the initiator or a servant for action a.

We say that a composition O is *coherent* with a community $\mathcal{C}$ if its tree $\mathcal{T}(O)$ has the following properties:

- for each edge labeled with (I, a, S), the action a is in the alphabet of $\mathcal{C}$, and for each e-Service A_i in $I \cup S$, A_i is a member of the community $\mathcal{C}$;

[4] We use $[i..j]$ to denote the set $\{i, \ldots, j\}$.

- for each path p in $\mathcal{T}(O)$ from the root of $\mathcal{T}(O)$ to a node x, and for each e-Service A_i appearing in p, the projection of p on A_i is a (sequence of annotated actions represented by) a node y in the execution tree $\mathcal{T}(A_i)$ of A_i, and moreover, if x is final in $\mathcal{T}(O)$, then y is final in $\mathcal{T}(A_i)$.

Next we turn to modeling client requests. Following [6], we assume that the client request is a specification of the pure-initiator e-Service that the client wants to realize. However in this paper we allow for underspecification of such a pure-initiator client e-Service, that shows up in the form of *don't-care nondeterminism* in its specification. Moreover we allow in specified points of the pure-initiator client e-Service that the e-Service itself is interleaved with activities performed by other e-Service in the community in which the client is not involved at all (but of which the client is in any case aware).

More precisely, we define as *client specification* a specification of the composition tree that the client would like to have realized using the e-Services in the community. Of the composition tree, the client specifies (i) the actions for which he is the initiator, and (ii) the possibility of having activities in which the client himself is not involved. Notably, we allow for incomplete information on the tree specified by the client, in forms of don't-care nondeterminism. Formally, we define the client specification as a *nondeterministic* FSM $\mathcal{A}_0 = (\Sigma_0, S_0, s_0^0, \delta_0, F_0)$, where:

- $\Sigma_0 = \{a^{\gg} \mid a \in \Sigma\} \cup \{\tau\}$ where τ is a special action that represents a finite sequence of actions in which the client is not the initiator (nor a servant);
- S_0 is the set of states;
- s_0^0 is the initial state;
- $\delta_0 : S_0 \times \Sigma_0 \to 2^{S_0}$ is a partial function that given a state and an action returns the set of possible successor states;
- $F_0 \subseteq S_0$ is the set of final states.

Observe that the nondeterministic FSM $\mathcal{A}_0$ specifies a *set* $\mathcal{T}(\mathcal{A}_0)$ of composition trees, and the client requires the orchestrator to realize one (any one) among such trees. Specifically, each composition tree in $\mathcal{T}(\mathcal{A}_0)$ is obtained by

- unfolding the FSM and while doing so, resolving the nondeterminism by choosing a single successor state for each transition (including τ transitions); this generates a (deterministic), possibly infinite tree, whose edges are labeled by Σ_0 and whose nodes, corresponding to final states of $\mathcal{A}_0$, are annotated as final;
- replacing each edge labeled by $a^{\gg}$ with an edge labeled by $(0, a, \cdot)$; this means that in the composition the client is the initiator of a;
- replacing each edge labeled by τ with a finite sequence of edges, each one labeled by $(j, a, \cdot)$, where a is some action, and $j \in [1..n]$; this means that for a τ action the composition can contain any finite sequence of interactions initiated by whatever e-Service except by the client;
- choosing for each edge a set of servants, and adding it to the label of the edge.

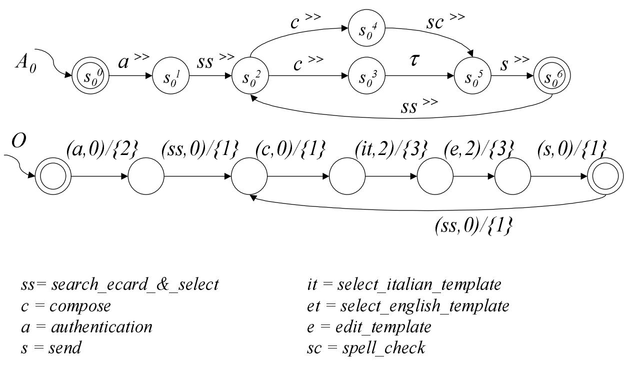

Figure 2: Client specification and composition.

We say that a composition O *realizes* a client specification $\mathcal{A}_0$ if $\mathcal{T}(O) \in \mathcal{T}(\mathcal{A}_0)$.

Given a community $\mathcal{C}$ of e-Services (consisting of both pure-servant, pure-initiator and mixed e-Services), and a client specification $\mathcal{A}_0$, the problem of *composition existence* is the problem of checking whether there exists a composition that is coherent with $\mathcal{C}$ and that realizes $\mathcal{A}_0$. The problem of *composition synthesis* is the problem of synthesizing a composition that is coherent with $\mathcal{C}$ and that realizes $\mathcal{A}_0$.

Since we are considering e-Services that have a finite number of states, we would like also to have a composition that can be represented with a finite number of states, i.e., as a Mealy FSM (MFSM) of the form $O = (\Sigma\times[0..n], 2^{[1..n]}, S_c, s_c^0, \delta_c, \omega_c, F_c)$, where:

- $\Sigma \times [0..n]$ is the alphabet of the MSFM, which denotes actions and their initiator;
- S_c, s_c^0, δ_c, F_c are the set of states, the initial state, the transition function, and the final set of states of the MSFM, in analogy with the e-Service FSMs;
- $2^{[1..n]}$ is the output alphabet of the MFSM, which is used to denote which are the servants of each action;
- $\omega_c : S_c \times \Sigma \times [0..n] \rightarrow 2^{[1..n]}$ is the output function of the MFSM, which, given a state, an action a, and an initiator for a, returns the set of servant e-Services for a; we assume that the output function ω_c is defined exactly when δ_c is so.

EXAMPLE 3.1. *Figure 2 shows a client specification $\mathcal{A}_0$, which specifies that the client would like to act as initiator of a* `authentication` *action, followed by a* `search_ecard_&_select` *action. At this point the client specifies a nondeterministic choice between two possible paths: along one path he first* `composes` *a message and then asks for a* `spell_check`*; along the other path he* `composes` *a message and then he allows the orchestration to act in a way not requiring interaction with the client himself (τ action). The client "doesn't care" which path is followed and sets the composition synthesis free to specify which sequence of action to execute. Next, the client would like to* `send` *a message. Finally, he wants to choose whether to stop or send another* e-card *by performing the action* `search_ecard_&_select`*.*

O is the MSFM[5] *that represents a composition coherent with the* e*-Services of Example 2.1 and realizing the client specification $\mathcal{A}_0$. The composition specifies the client as initiator of the action* `authentication` *with A_2 as servant, then specifies the client as initiator of the action* `search_ecard_&_select` *with A_1 as servant; at this point, the composition executes the "lower path" of the client specification and again specifies the client as initiator of the* `compose` *action, for which A_1 is the servant. Then, the composition specifies A_2 as initiator of the actions* `select_italian_template` *and* `edit_template`*, which are served by A_3 (note that, correctly, the client is not involved, as specified by the τ action in $\mathcal{A}_0$). Next, the composition specifies the client as initiator of the action* `send` *with A_1 as servant. Finally, the client chooses whether to stop the service execution or to send another* e*-card: in the latter case, the composition specifies that the client is initiator for the action* `search_ecard_&_select` *for which A_1 is servant.*

Note that O is not the only composition which is coherent wrt the client specification. In effect, several ones exists. For example, a first one is identical to O, but the τ action is realized by the sequence `select_english_template` *and* `edit_template`*. Another one is obtained when the nondeterminism is resolved in the composition by choosing the "upper path" of the client specification: it specifies the client as initiator of (*i*) the* `compose` *action, for which A_1 is the servant, and of (*ii*) the* `spell_check` *action for which A_2 is the servant. No τ action exists along this path. Next, the computation continues as before, wrt the* `send` *and* `search_ecard_&_select` *actions.*

4. SYNTHESIS TECHNIQUE

We address the problem of composition existence and synthesis in the FSM-based framework introduced above. The basic tool we use is reducing the problem of composition existence to satisfiability of a formula written in PDL_{gm}, a variant of PDL [23] equipped with graded modalities [14, 16, 38].

4.1 *PDLgm*

In PDL_{gm}, starting from a set of *atomic propositions* and *atomic actions*, one can build complex formulas and complex programs by applying the *constructs* shown in Figure 3. We also use the usual abbreviation $\phi_1 \rightarrow \phi_2$ for $\neg(\neg\phi_1 \vee \phi_2)$.

In PDL and its variants, the semantics is specified through the notion of interpretation. An *interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *interpretation domain* and $\cdot^{\mathcal{I}}$ is an *interpretation function* that assigns to each formula ϕ a subset $\phi^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each program R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, respecting the conditions specified in Figure 3.

An interpretation $\mathcal{I}$ is a *model* of a formula ϕ if $\phi^{\mathcal{I}} \neq \emptyset$. A formula ϕ is *satisfiable* if it admits a model.

PDL_{gm} formally corresponds to the well-known description logic $\mathcal{ALCQ}_{reg}$ [10]. Exploiting such a correspondence we can state that PDL_{gm} enjoys two properties that are of particular interest for our aims. The first is the *tree model property*, which says that every model of a PDL_{gm} formula can be unwound to a (possibly infinite) tree-shaped model (considering domain elements as nodes and atomic actions

[5]An edge (s_1, s_2) labeled $(a, I)/S$ indicates a transition $\delta(s_1, (a, I)) = s_2$ with output S, where I is the initiator of a and S is the set of servants.

Formulas ϕ	**Syntax**	**Semantics**
atomic propositions	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
negation	$\neg\phi$	$\Delta^{\mathcal{I}} \setminus \phi^{\mathcal{I}}$
conjunction	$\phi_1 \wedge \phi_2$	$\phi_1^{\mathcal{I}} \cap \phi_2^{\mathcal{I}}$
disjunction	$\phi_1 \vee \phi_2$	$\phi_1^{\mathcal{I}} \cup \phi_2^{\mathcal{I}}$
universal modalities	$[R]\phi$	$\{o \mid \forall o'.(o,o') \in R^{\mathcal{I}} \to o' \in \phi^{\mathcal{I}}\}$
existential modalities	$\langle R\rangle\phi$	$\{o \mid \exists o'.(o,o') \in R^{\mathcal{I}} \wedge o' \in \phi^{\mathcal{I}}\}$
graded modalities	$(\leq n\ \langle P\rangle\phi)$	$\{o \mid \sharp\{(o,o') \in P^{\mathcal{I}} \mid o' \in \phi^{\mathcal{I}}\} \leq n\}$
Programs R	**Syntax**	**Semantics**
atomic action	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
union	$R_1 \cup R_2$	$R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$
concatenation	$R_1; R_2$	$R_1^{\mathcal{I}}; R_2^{\mathcal{I}}$
refl. trans. clos.	R^*	$(R^{\mathcal{I}})^*$
test	$(\phi)?$	$\{(o,o) \mid o \in \phi^{\mathcal{I}}\}$

Figure 3: Syntax and semantics of PDL_{gm}**.**

as edges). The second is the *small model property*, which says that every PDL_{gm} that is satisfiable, admits a finite model whose size (in particular the number of domain elements) is at most exponential in the size of the formula itself.

4.2 PDL_{gm} Encoding

Given the specification of a client e-Service in terms of a nondeterministic FSM $\mathcal{A}_0$ and a community of n e-Services $A_1, \ldots, A_n$, we build an PDL_{gm} formula Φ. As set of atomic propositions in Φ we have (i) one atomic proposition s for each state s of A_j, for $j \in [0..n]$, which intuitively denotes that A_j is in state s;[6] (ii) atomic propositions F_j, for $j \in [0..n]$, denoting whether A_j is in a final state; (iii) atomic propositions served_j, for $j \in [1..n]$, denoting whether (component) FSM A_j is a servant of a transition; (iv) atomic propositions $\mathsf{initiated}_j$, for $j \in [0..n]$, denoting whether FSM A_j is a servant of a transition; (v) an atomic proposition Init representing the initial state of the required service; (vi) one atomic proposition a for each action $a \in \Sigma$. We have a single atomic action *trans* in Φ, such a role will be used to denote state transitions caused by actions. The formula Φ is formed as follows.

- For the client specification $\mathcal{A}_0 = (\Sigma, S_0, s_0^0, \delta_0, F_0)$ we form the formula $[trans^*]\Phi_0$ where Φ_0 is the conjunction of:
  - $s \to \neg s'$, for all pairs of states $s, s' \in S_0$; these say that atomic concepts representing different states are disjoint.
  - $s \to \bigvee_{s' \in \delta_0(s, a^{\gg})} \langle trans\rangle(\mathsf{initiated}_0 \wedge a \wedge s')$, for each $a \in \Sigma$ and s with $\delta_0(s, a^{\gg}) \neq \emptyset$; these encode the transitions different from τ.
  - $s \to \bigvee_{s' \in \delta_0(s,\tau)} \langle R_\tau^*\rangle s'$, for each s with $\delta_0(s,\tau) \neq \emptyset$, where R_τ stands for

    $$((\leq 1\ \langle trans\rangle \neg\mathsf{initiated}_0))?; trans; (\neg\mathsf{initiated}_0)?$$

    These encode the τ transitions of $\mathcal{A}_0$; a τ transition is realized through a *single sequence* of actions in which $\mathcal{A}_0$ does not participate; the qualified number restriction is used to ensure that there is a single sequence.
  - $s \to [trans](a \to \neg\mathsf{initiated}_0)$, for each a such that $\delta(s, a^{\gg})$ is not defined; these say that $a^{\gg}$ is not a possible transition.
  - $s \to [trans]\mathsf{initiated}_0$, if $\delta(s,\tau)$ is not defined; these say when a τ transition is not possible.
  - $F_0 \equiv \bigvee_{s \in F_0} s$; this highlights final states of $\mathcal{A}_0$.
- For each component FSM $A_i = (\Sigma, S_i, s_i^0, \delta_i, F_i)$, we form the formula $[trans^*]\Phi_i$, where Φ_i is the conjunction of:
  - $s \to \neg s'$, for all distinct pairs of states $s, s' \in S_i$.
  - $s \to [trans](a \wedge \mathsf{served}_i \to s')$, for each s and a such that $s' = \delta_i(s, {}^{\gg}a)$; these encode the transitions of A_i, conditioned to the fact that A_i is required to be a servant of a in the composition.
  - $s \to [trans](a \wedge \mathsf{initiated}_i \to s')$, for each s and a such that $s' = \delta_i(s, a^{\gg})$; these encode the transitions of A_i, conditioned to the fact that A_i is required to be the initiator of a in the composition.
  - $s \to [trans](a \to \neg\mathsf{served}_i)$, for each s and a such $\delta_i(s, {}^{\gg}a)$ is not defined.
  - $s \to [trans](a \to \neg\mathsf{initiated}_i)$, for each s and a such $\delta_i(s, a^{\gg})$ is not defined.
  - $s \to [trans](\mathsf{served}_i \vee \mathsf{initiated}_i \vee s)$, for each $s \in S_i$; this encodes that when A_i does not participate to an action, it does not change state.
  - $F_i \equiv \bigvee_{s \in F_i} s$; this highlights final states of A_i.
- to encode the general structure of models, we form the formula $[trans^*]\Psi$, where Ψ is the conjunction of:
  - $(\leq 1\ \langle trans\rangle(a \wedge \mathsf{initiated}_0))$, for each action $a \in \Sigma$: this represents that the realized composition is deterministic wrt to the action annotated by the initiator.

[6]In this paper we are not concerned with compact representations of the states of the FMS. However, we observe that if states are succinctly represented (e.g., in binary format) then, in general, we can exploit such a representation in Φ to get a corresponding compact PDL_{gm} formula Φ as well.

- $[trans](\bigvee_{a\in\Sigma} a)$; to represent that each transition is caused by an action.
- $[trans](\bigvee_{i\in[1..n]} \mathsf{served}_i)$; to represent that each transition must have some e-Service as servant.
- $[trans](\bigvee_{i\in[0..n]} \mathsf{initiated}_i)$; to represent that each transition must have an initiator, either an e-Service in the community or the client.
- $[trans](\neg\mathsf{initiated}_i \vee \neg\mathsf{initiated}_j)$, for each $i, j \in [0..n]$ with $i \neq j$; to represent that transitions have a single initiator (but possibly several servants).
- $F_0 \rightarrow \bigwedge_{i\in[1..n]} F_i$; this says that when the client specification is in a final state also all component e-Services must be in a final state.
- $\mathsf{Init} \rightarrow s_0^0 \wedge \bigwedge_{i\in[1..n]} (s_i^0)$; to represent that initially all e-Services are in their initial state.
- $\mathsf{Init} \rightarrow \bigwedge_{a\in\Sigma} (\neg a)$
  $\mathsf{Init} \rightarrow \bigwedge_{i\in[0..n]} (\neg\mathsf{initiated}_i)$
  $\mathsf{Init} \rightarrow \bigwedge_{i\in[1..n]} (\neg\mathsf{served}_i)$;
  to represent that initially no action has been executed yet.

Finally, we define Φ as $\mathsf{Init} \wedge [trans^*]\Phi_0 \wedge \bigwedge_{i=1,\ldots,n} [trans^*]\Phi_i \wedge [trans^*]\Psi$.

4.3 Results

THEOREM 4.1. *The PDL_{gm} formula*

$$\Phi = Init \wedge [trans^*]\Phi_0 \wedge \bigwedge_{i=1,\ldots,n} [trans^*]\Phi_i \wedge [trans^*]\Psi$$

is satisfiable if and only if there exists a composition that is coherent with $A_1, \ldots, A_n$ and that realizes the client specification $\mathcal{A}_0$.

Proof (sketch). "$\Leftarrow$" From the composition tree $T(O)$ of an composition O that is coherent with $A_1, \ldots, A_n$ and that realizes $\mathcal{A}_0$, we can construct a tree-like model of Φ such that Init is satisfied in the root. Such a model is essentially obtained from $T(O)$ by annotating the nodes of $T(O)$ with the states of the services, and the client specification, and with the initiator and servant(s) of each action.

"$\Rightarrow$" If Φ is satisfiable, then there exists a tree-like model of Φ where Init is satisfied in the root. From such a model, one can derive a composition tree $T(O)$ that is coherent with $A_1, \ldots, A_n$ and realizes $\mathcal{A}_0$, essentially by extracting the information on initiator and servants from the interpretation of the propositions $\mathsf{initiated}_i$ and served_i. □

Observe that, the size of Φ is polynomially related to the size of $\mathcal{A}_0$, $A_1, \ldots, A_n$. By the small model property of PDL_{gm}, if Φ is satisfiable, then it is satisfiable in a model that is at most exponential in the size of Φ. From such a finite model one can extract a representation of the composition that has the form of a MFSM. Specifically, given a finite model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, we define such an MFSM $O = (\Sigma \times [0..n], 2^{[1..n]}, S_c, s_c^0, \delta_c, \omega_c, F_c)$ as follows:

- $S_c = \Delta^{\mathcal{I}}$;
- $s_c^0 = \mathsf{Init}^{\mathcal{I}}$;
- $s' = \delta_c(s, (a, I))$ iff $(s, s') \in trans^{\mathcal{I}}$, $s' \in a^{\mathcal{I}}$, and $s' \in \mathsf{initiated}_I^{\mathcal{I}}$;
- $\{j_1, \ldots, j_\ell\} = \omega_c(s, (a, I))$ iff $(s, s') \in trans^{\mathcal{I}}$, $s' \in a^{\mathcal{I}}$, $s' \in \mathsf{initiated}_I^{\mathcal{I}}$, and $s' \in \mathsf{served}_j^{\mathcal{I}}$, for exactly those j in $\{j_1, \ldots, j_\ell\}$;
- $F_c = F_0^{\mathcal{I}}$.

From the above construction, we have the following lemma.

LEMMA 4.2. *Any finite model of the PDL_{gm} formula Φ built as above denotes a MFSM composition that is coherent with $A_1, \ldots, A_n$ and realizes the client specification $\mathcal{A}_0$.*

By Theorem 4.1 and Lemma 4.2, we get the following result.

THEOREM 4.3. *If there exists a composition that is coherent with $A_1, \ldots, A_n$ and that realizes a client specification $\mathcal{A}_0$, then there exists one that is a MFSM of size at most exponential in the size of $\mathcal{A}_0, A_1, \ldots, A_n$.*

Proof (sketch). By Theorem 4.1, if there exists a composition tree, then the PDL_{gm} formula Φ constructed as above is satisfiable. In turn, if Φ is satisfiable, for the small-model property of PDL_{gm}, there exists a model $\mathcal{I}$ of size at most exponential in Φ, and hence in $\mathcal{A}_0$ and $A_1, \ldots, A_n$. From $\mathcal{I}$ we can construct a MFSM A_c as above. Notice that the composition tree generated by A_c essentially corresponds the tree-like model obtained by unwinding $\mathcal{I}$. □

By the theorems above and the EXPTIME-completeness of satisfiability in PDL_{gm} (cf. [10]), we get the following complexity upper bound.

THEOREM 4.4. *Checking the existence of a (MFSM) composition that is coherent with $A_1, \ldots, A_n$ and that realizes a client specification $\mathcal{A}_0$ can be done in EXPTIME.*

We do not have a tight lower bounds for the complexity of the composition existence problem, however we conjecture that the above bounds are in fact tight (i.e., that the problem is EXPTIME-hard). Notice that, even if we do not have a tight lower bound yet, it is easy to find cases in which the composition must be exponential in the size of the component e-Services and the client specification. Hence exponentiality (though not necessarily EXPTIME-hardness) is inherent to the problem.

Exploiting reasoning methods for PDLs (or corresponding Description Logics) based on model construction, such as tableaux algorithms [8, 12, 5], one can actually construct such a MFSM composition. Notice that such algorithms need to be able to deal with full reflexive transitive closure, introduced in Φ due to τ transitions in the client specification. If such τ transitions are not present (while the client specification may still be underspecified), one can resort to state-of-the-art implemented Description Logics systems, such as FaCT [24] and Racer [21, 32], to check the existence of a composition, while these systems are not yet usable for the actual construction of the MFSM composition, since they do not return the constructed model. A prototype implemented in Java that actually generates a composition using PDL tableaux techniques (not fully supporting transitive closure yet) is available[7].

[7]cf. the PARIDE (Process-based frAmewoRk for composItion and orchestration of Dynamic E-services) Open Source

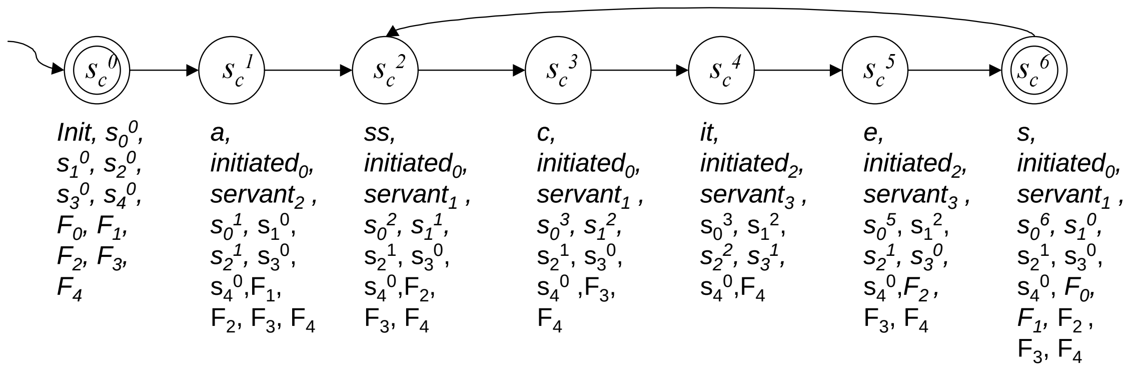
s_c^0
s_c^1
s_c^2
s_c^3
s_c^4
s_c^5
s_c^6
Init, s_0^0, s_1^0, s_2^0, s_3^0, s_4^0, F_0, F_1, F_2, F_3, F_4
a, initiated$_0$, servant$_2$, s_0^1, s_1^0, s_2^1, s_3^0, s_4^0, F_1, F_2, F_3, F_4
ss, initiated$_0$, servant$_1$, s_0^2, s_1^1, s_2^1, s_3^0, s_4^0, F_2, F_3, F_4
c, initiated$_0$, servant$_1$, s_0^3, s_1^2, s_2^1, s_3^0, s_4^0, F_3, F_4
it, initiated$_2$, servant$_3$, s_0^3, s_1^2, s_2^2, s_3^1, s_4^0, F_4
e, initiated$_2$, servant$_3$, s_0^5, s_1^2, s_2^1, s_3^0, s_4^0, F_2, F_3, F_4
s, initiated$_0$, servant$_1$, s_0^6, s_1^0, s_2^1, s_3^0, s_4^0, F_0, F_1, F_2, F_3, F_4

Figure 4: Model of the PDL_{gm} formula encoding our running example.

We conclude the section by showing the encoding of our running example in a PDL_{gm} formula Φ and then show and discuss its model. For lack of space we report only part of the encoding of the client specification $\mathcal{A}_0$.

The transitions different from τ are encoded in Φ as follows:

$$\begin{array}{l}
s_0^0 \rightarrow \langle trans \rangle(\mathsf{initiated}_0 \wedge a \wedge s_0^1)\\
s_0^1 \rightarrow \langle trans \rangle(\mathsf{initiated}_0 \wedge ss \wedge s_0^2)\\
s_0^2 \rightarrow \langle trans \rangle(\mathsf{initiated}_0 \wedge c \wedge s_0^3) \vee \langle trans \rangle(\mathsf{initiated}_0 \wedge c \wedge s_0^4)\\
\dots
\end{array}$$

Note that the or in the third assertion is due to the fact that two c-transitions are defined from state s_0^2, leading to states s_0^3 and s_0^4, respectively.

The transition involving τ is captured by:
$s_0^3 \rightarrow \langle R_\tau^* \rangle s_0^5$ where R_τ stands for

$$((\leq 1\ \langle trans \rangle \neg \mathsf{initiated}_0))?; trans; (\neg \mathsf{initiated}_0)?$$

The following set of assertions encode that transitions different from τ are not defined from a state:

$$\begin{array}{ll}
s_0^0 \rightarrow [trans](ss \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](ss \rightarrow \neg\mathsf{initiated}_0)\\
s_0^0 \rightarrow [trans](c \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](a \rightarrow \neg\mathsf{initiated}_0)\\
\dots & s_0^3 \rightarrow [trans](c \rightarrow \neg\mathsf{initiated}_0)\\
s_0^0 \rightarrow [trans](it \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](s \rightarrow \neg\mathsf{initiated}_0)\\
s_0^0 \rightarrow [trans](et \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](it \rightarrow \neg\mathsf{initiated}_0)\\
\dots & s_0^3 \rightarrow [trans](et \rightarrow \neg\mathsf{initiated}_0)\\
s_0^1 \rightarrow [trans](a \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](e \rightarrow \neg\mathsf{initiated}_0)\\
s_0^1 \rightarrow [trans](c \rightarrow \neg\mathsf{initiated}_0) & s_0^3 \rightarrow [trans](sc \rightarrow \neg\mathsf{initiated}_0)\\
\dots & \dots
\end{array}$$

In $\mathcal{A}_0$ from s_0^3 a single τ transition is defined. While no τ transitions are defined for states different from s_0^3:

$$\begin{array}{ll}
s_0^0 \rightarrow [trans]\mathsf{initiated}_0 & s_0^4 \rightarrow [trans]\mathsf{initiated}_0\\
s_0^1 \rightarrow [trans]\mathsf{initiated}_0 & s_0^5 \rightarrow [trans]\mathsf{initiated}_0\\
s_0^2 \rightarrow [trans]\mathsf{initiated}_0 & s_0^6 \rightarrow [trans]\mathsf{initiated}_0
\end{array}$$

Observe that, in general, our framework does not prevent to define two τ transitions starting from a state: in this case, the client leaves the composition synthesis free to choose one transition and replace it with whatever sequence of actions. Also, our framework allows for both a τ transition and a non-τ transition, say labeled by action a, originating from a same state. Observe that even if the composition synthesis chooses to realize the τ transition by the a action, the composition remains deterministic, since the two a actions have different initiators (in one case it is the client, in the other it is an e-Service in the community).

Figure 4 shows a model for the PDL_{gm} formula Φ: for sake of clarity, for each state, we report only the atomic propositions that are true in that state and we highlight in italics the propositions that change their value from one state to another; also, we do not show the role *trans*, which labels all transitions of the model.

It is easy to verify that the interpretation in Figure 4 is indeed a model of Φ, since for each state (domain element), all constraints expressed in Φ are satisfied. For example, in the initial state s_c^0, (i) Init holds, (ii) all the FSM associated to the component e-Services and to the client specification start from their initial state, (iii) no action is executed, therefore no e-Service act as initiator nor as a servant, (iv) since the composition is in a final state, also all the component e-Services are in a final state. State s_c^1 is reached after performing action a, whose initiator is the client and whose servant is A_2: therefore, both the client specification and A_2 moved, respectively, to states s_0^1 and s_2^1, while the other e-Services do not change state. Note that in s_c^1 despite the fact that all e-Services are in a final state, the client specification is not. Finally, observe that in this model s_0^3 holds in s_c^4. In fact the formula in Φ talking about τ transitions does not require any specific state to be assigned to the intermediate steps that realize the τ transition, namely s_c^4 in our model.

It is easy to see that the (MFSM) composition reported in Figure 2 can be obtained from the model in Figure 4 by following the construction shown at the beginning of this section.

5. RELATED WORK

Generally speaking, research on e-Service description and modeling, and e-Service composition (especially synthesis, less orchestration) is relevant for our work [17]. A nice survey on such issues is [25]. Most of the work on these issues is based on research in workflows, which model business processes as sequences of (possibly partially) automated activities, focusing on both data and control flow among them

Project: `http://sourceforge.net/projects/paride/` that is the general framework in which we intend to release the various prototypes produced by our research.

(e.g., [36, 37]). Specific representations of e-Services are often based on finite state formalisms, e.g., in [31] e-Services are represented as statecharts, and in [9] e-Services are modeled as Mealy machines. In our paper, we are not concerned with the representation of e-Services. We are only interested in singling out (in the abstract) the computations that such e-Services can execute. Hence we simply model e-Services as finite state machines (i.e., deterministic finite transition systems).

Following [6] two specific features form the base of our proposal. The first one is that *the composition involves the concurrent executions of several* e-*Services*. Only few proposals in the literature follow a similar idea. In particular, the most related ones are [9, 29, 35]: they have in common with our proposal the fact that the e-Services are seen as white-boxes and hence they can be interleaved if needed. The composition deals with suitably controlling such an interleaving so as to realize the client request. Note that, most work on composition is based on the idea of *sequentially* composing the available e-Services, which are considered as black boxes, and hence atomically executed [3, 26, 28, 39, 1]. Such an approach to composition is tightly related to Classical Planning in AI [19]. The second basic feature is that *the client request is a specification of the transition system that the client wants to be able to execute*. This feature is, to the best of our knowledge, unique to the proposal in [6] and hence to the one here. Indeed even [9, 29] actually focus on realizing a single execution fulfilling the client request. Notice that such an execution may depend on conditions to be verified at run time, but not on further choices made by the client. Only the proposal in [35] has some similarities with our: indeed, there, the client goal is expressed in a specific branching-time logic, that allows to specify alternative paths of execution under the control of the client. Though, their goals are still essentially based on having a main execution to follow, plus some side paths that are typically used to resolve exceptional circumstances.

Wrt architectural distinctions, in [25] two main different kinds of composition are identified: (i) the peer-to-peer approach in which the individual e-Services interact among themselves and with the client directly, and (ii) the mediated approach in which the control over the available e-Services is centralized. With respect to such a classification, the research works reported in [29, 11, 37, 30] can be classified into the mediated approach to composition. Conversely in [15] the enactment of a composite e-Service is carried out in a decentralized way, through peer-to-peer interactions. In [9], a peer-to-peer approach is considered, and the interplay between a composite e-Service and component ones is analyzed, also in presence of unexpected behavior. Our proposal naturally fits in the mediated approach. However, using standard distribution and replication techniques developed by the Distributed Systems community, it is possible, in principle, to partition the synthesized composition by suitably projecting it over the peers (the available e-Services) and to distribute the various partitions, thus obtaining a peer-to-peer approach. This issue will be further investigated in future work.

The last point that we want to discuss here regards the distinction between data and process that often shows up in the e-Service literature. Indeed we have two extremes in dealing with data and process. One end of the spectrum is well explored by the literature on data integration that fully takes into account the data, but not the process [22, 27]. Interestingly, there are some proposals that base e-Service composition for data intensive e-Services on such a literature, avoiding to talk about the process as much as possible [20]. The other end of the spectrum is much less studied. Our proposal, together with those in [9, 29, 35], tries to explore such an end of the spectrum. Observe that, introducing data in a naive way in our setting is in fact possible, but would make composition exponential in the data. This is to be considered unacceptable, since the amount of data is typically huge (wrt the size of the e-Services) and hence one wants to keep the computations polynomial in the data. More generally, both ends of this spectrum (only data and only process) deal with problems that are quite difficult. Finding a good way to integrate the two, without multiplying the complexities, is probably going to become one of the key problems in e-Service composition in the future.

6. CONCLUSIONS

In this paper we have developed a *sound*, *complete*, and *terminating* technique for automatic composition synthesis of e-Services, starting from a client specification, which allows for underspecification of the transition system that the client wants to be able to execute. The technique is based on reasoning in a variant of PDL.

In e-Service composition, there is a clear distinction between the role of the client asking for a service, and the role of the software artifact that realizes the service. In our model, such distinction is reflected in the fact that a client is always considered as an initiator of actions, and not as a servant, while an e-Service is seen mostly as a servant. By blurring such distinction, one makes the notion of e-Service similar to the notion of agent. Under this perspective, the results presented here become relevant for automatic synthesis of agents. In the future, we aim at investigating this perspective in detail.

Acknowledgement

This work has been supported by MIUR through the “FIRB 2001” project *MAIS* (`http://www.mais-project.it`, Workpackage 2), and “Società dell' Informazione” sub project SP1 “Reti Internet: Efficienza, Integrazione e Sicurezza”. It has been also supported by the European projects SEWASIE (IST-2001-34825), EUPUBLI.com (IST-2001-35217) and INTEROP Network of Excellence (IST-508011).

7. REFERENCES

[1] M. Aiello, M. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, and P. Traverso. A Request Language for Web-Services Based on Planning and Constraint Satisfaction. In *Proc. of VLDB-TES 2002*.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services (Version 1.1). `http://www-106.ibm.com/developerworks/library/ws-bpel/`.

[3] 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 ISWC 2002*.

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

[5] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69(1):5–40, 2001.

[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 ICSOC 2003*.

[7] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. A Foundational Vision of *e*-Services. In *Proc. of WES 2003*.

[8] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable Reasoning in Tterminological Knowledge Representation Systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.

[9] 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 the WWW 2003 Conference*.

[10] D. Calvanese and G. De Giacomo. Expressive description logics. In Baader et al. [4], chapter 5, pages 178–218.

[11] F. Casati and M. Shan. Dynamic and Adaptive Composition of *e*-Services. *Information Systems*, 6(3), 2001.

[12] G. De Giacomo and F. Massacci. Combining Deduction and Model Checking into Tableaux and Algorithms for converse-PDL. *Information and Computation*, 160(1–2):117–137, 2000.

[13] A. Elmagarmid and W. McIver Jr. The Ongoing March Towards Digital Government (Special Issue). *IEEE Computer*, 34(2), 2001.

[14] M. Fattorosi-Barnaba and F. De Caro. Graded modalities I. *Studia Logica*, 44:197–221, 1985.

[15] M. Fauvet, M. Dumas, B. Benatallah, and H. Paik. Peer-to-Peer Traced Execution of Composite Services. In *Proc. of VLDB-TES 2001*.

[16] K. Fine. In so many possible worlds. *Notre Dame Journal of Formal Logic*, 13(4):516–520, 1972.

[17] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[18] D. Georgakopoulos, editor. *Proc. of RIDE-VE'99*

[19] M. Ghallab, D. Nau, and P. Traverso. *Automated Task Planning: Theory & Practice*. Morgan Kaufmann, 2004.

[20] 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*.

[21] V. Haarslev and R. Möller. RACER system description. In *Proc. of IJCAR 2001*, volume 2083 of *LNAI*, pages 701–705. Springer-Verlag, 2001.

[22] A. Y. Halevy. Answering Queries Using Views: A Survey. *J. of Very Large Data Bases*, 10(4):270–294, 2001.

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

[24] I. Horrocks. The FaCT system. In *Proc. of TABLEAUX'98*, volume 1397 of *LNAI*, pages 307–312. Springer-Verlag, 1998.

[25] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Behind the Curtain. In *Proc. of the PODS 2003 Conference*.

[26] U. Kuter, E. Sirin, D. Nau, B. Parsia, and J. Hendler. Information Gathering during Planning for Web Service Composition. In *Proc. of ICAPS-P4WGS 2004*.

[27] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS 2002*, pages 233–246.

[28] E. Martinez and Y. Lesperance. Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning. In *Proc. of ICAPS-P4WGS 2004*.

[29] S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2), 2001.

[30] M. Mecella and B. Pernici. Building Flexible and Cooperative Applications Based on *e*-Services. Technical Report 21-2002, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy, 2002. (available on line at: `http://www.dis.uniroma1.it/~mecella/publications/mp_techreport_212002.pdf`).

[31] M. Mecella, B. Pernici, and P. Craca. Compatibility of *e*-Services in a Cooperative Multi-Platform Environment. In *Proc. VLDB-TES 2001*.

[32] R. Möller and V. Haarslev. Description logic systems. In Baader et al. [4], chapter 8, pages 282–305.

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

[34] T. Pilioura and A. Tsalgatidou. *e*-Services: Current Technologies and Open Issues. In *Proc. of VLDB-TES 2001*.

[35] F. Pistore, M.and Barbon, P. Bertoli, and P. Shaparau, D.and Traverso. Planning and Monitoring Web Service Composition. In *Proc. of ICAPS-P4WGS 2004*.

[36] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and Composing Service-based and Reference Process-based Multi-enterprise Processes. In *Proc. of CAiSE 2000*.

[37] G. Shegalov, M. Gillmann, and G. Weikum. XML-enabled workflow management for *e*-Services across heterogeneous platforms. *J. of Very Large Data Bases*, 10(1), 2001.

[38] W. Van der Hoek. On the semantics of graded modalities. *Journal of Applied Non-Classical Logics*, 2(1):81–123, 1992.

[39] J. Yang and M. Papazoglou. Web Components: A Substrate for Web Service Reuse and Composition. In *Proc. of CAiSE 2002*.