Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Università degli Studi di Roma "La Sapienza"

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

Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo,
Maurizio Lenzerini, Massimo Mecella

Technical Report n. 9
2004

Aracne

I *Technical Reports* del Dipartimento di Informatica e Sistemistica "Antonio Ruberti" svolgono la funzione di divulgare tempestivamente, in forma definitiva o provvisoria, i risultati di ricerche scientifiche originali. Per ciascuna pubblicazione vengono soddisfatti gli obblighi previsti dall'art. 1 del D.L.L. 31.8.1945, n. 660 e successive modifiche.
Copie della presente pubblicazione possono essere richieste alla Redazione.

Dipartimento di Informatica e sistemistica "Antonio Ruberti"
Università degli studi di Roma "La Sapienza"
Via Eudossiana, 18 - 00184 Roma
Via Buonarroti, 12 - 00185 Roma
Via Salaria, 113 - 00198 Roma
www.dis.uniroma1.it

www.aracne–editrice.it
info@aracne–editrice.it

00173 Roma
via Raffaele Garofalo, 133 A/B
(06) 72672222 – (06) 93781065
telefax 72672233

ISBN 88–7999–696–7

I edizione: aprile 2004

Finito di stampare nel mese di aprile del 2004
dalla tipografia « Grafica Editrice Romana S.r.l. » di Roma
per conto della « Aracne editrice S.r.l. » di Roma
Printed in Italy

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

Daniela Berardi *Diego Calvanese*
Giuseppe De Giacomo *Maurizio Lenzerini*
Massimo Mecella

Abstract

In this paper we study automatic composition synthesis of *e*-Services, based on automated reasoning. The behavior of an *e*-Service is represented 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 a Description 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.

Keywords: *e*-Service Composition, Description Logics, Reasoning about Actions and Change, Automated Reasoning

1 Introduction

e-Services represent a new model in the utilization of the network, in which self-contained, modular applications can be described, published, located and dynamically invoked, in a programming language independent way. This model, sometimes called *Service Oriented Computing* (SOC [14]), enables building agile networks of collaborating business applications, distributed within and across organizational boundaries.[1].

The commonly accepted and minimal architecture for *e*-Services [15] consists of the following basic roles: (*i*) the service provider, which is the subject (e.g., an organization) providing services; (*ii*) the service directory, which is the subject providing a repository/registry of service descriptions, where providers publish their services and requestors find services; and, (*iii*) the service requestor, also referred to as client, which

*Dipartimento di Informatica e Sistemistica "A. Ruberti", Università di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. Technical Report 09-04, 2004.

[1] cf., Service Oriented Computing Net: `http://www.eusoc.net/`

is the subject looking for and invoking the service in order to fulfill some goals. A requestor discovers a suitable service in the directory, and then it connects to the specific service provider and uses the service.

Research on *e*-Services spans over many interesting issues, including description, discovery, composition, synchronization, coordination, and verification [11]. In this paper, we are interested in automatic *e*-Service composition. *e*-Service *composition* addresses the situation when a client request cannot be satisfied by any available *e*-Service, but a *composite e*-Service, obtained by combining "parts of" available *component e*-Services, might be used. Each composite *e*-Service can be regarded as a kind of client wrt its components, since it (indirectly) looks for and invokes them. *e*-Service composition leads to enhancements of the *e*-Service architecture, by adding new elements and roles, such as brokers and integration systems, which are able to satisfy client needs by combining available *e*-Services.

Composition involves two different issues. The first, sometimes called *composition synthesis*, or simply *composition*, is concerned with synthesizing a new composite *e*-Service based on a set of available *e*-Services and the specification of a client request (called client specification). The synthesis process produces a specification of how to coordinate the component *e*-Services to obtain the composite *e*-Service that satisfies 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, often referred to as *orchestration*, is concerned with coordinating the various component *e*-Services according to some given specification, and also monitoring control and data flow among the involved *e*-Services, in order to guarantee the correct execution of the composite *e*-Service, synthesized in the previous phase.

Our focus in this paper is on automatic composition synthesis. On the basis of recent results appeared in the literature [3, 13, 6], we present an effective technique for automatic *e*-Service composition in the case where:

- the available *e*-Services have finite states,
- the client specification allows for incomplete information in the characterization of the requested service.

Notably, the kind of incomplete information allowed in the client specification enables the client to use an advanced form of "don't care" nondeterminism.

More specifically, we present an algorithm that, given (*i*) a client specification of the above nature and (*ii*) a set of available *e*-Services, synthesizes a composite *e*-Service that (*i*) uses only the available *e*-Services and (*ii*) interacts with the client "in accordance" with the input specification. We also study the computational complexity of our algorithm, and we show that it runs in exponential time with respect to the size of the input state machines. The algorithm is based on reducing the problem of checking the existence of a composition into concept satisfiability in a Description Logic (DL) knowledge base [1]. With this reduction, we show that reasoning tools for DLs can be directly used for composition synthesis, in particular by extracting a composite *e*-Service from a model of the DL knowledge base.

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 orchestration (i.e.,

composite e-Service), client specification, and composition. In Section 4 we present the technique to synthesize a composition of e-Services based on reasoning in DLs, and the results we obtained in terms of complexity and features of the obtained composition. Finally, in Section 5 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 [3, 4], and characterize the exported behavior of an e-Service by the set of (possibly infinite) sequences of actions that the e-Service participates in, annotated with the role (either initiator or servant) the e-Service takes in executing each action. In order to represent such a role, we adorn each action symbol in the execution tree 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 $a^{\gg}$.

The set of (annotated) sequences of actions of the e-Service can be represented, by collapsing common prefixes, as a (possibly infinite) *execution tree*. Observe that in such an execution tree, for each node we can have at most one successor node for each annotated action. We annotate the nodes of the execution tree with the information on when a sequence of actions from the root to the node can be considered a completed execution of the e-Service, in the sense that the e-Service can terminate.

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.

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 (i.e., deterministic and finite labeled transition systems). 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;

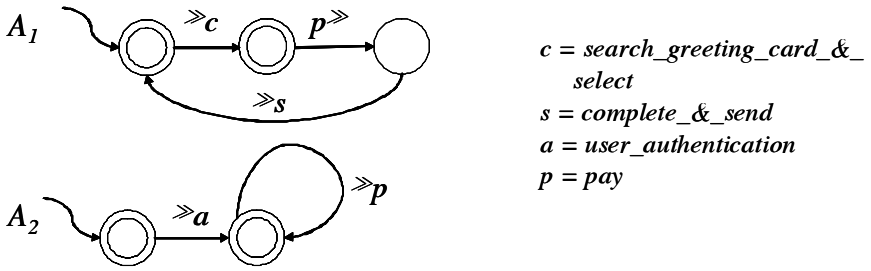

Fig. 1: e-Services of the community

- $F_i \subseteq S_i$ 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 obtained by following in all possible ways the transitions of A_i, and annotating as final those nodes corresponding to the traversal of final states.

In Figure 1 is shown a community of e-Services A_1, A_2,. A user would like to send an e-card and, after a payment is requested (and obtained) by the e-Service A_1, the user can complete the e-card and send it. A_1 repeatedly is servant for searching an e-card and selecting one among those returned (`search_greeting_card_&_select`), is initiator for a `pay` action and servant for writing and sending the e-card (`complete_&_send`). A_2, after validating a (registered) user information (e.g., name, credit card number, account number, ...) (action `user_authentication` it is servant of), repeatedly is ready to act as servant of a `pay` action.

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. An *orchestration* O (also called composite e-Service) of the e-Services in $\mathcal{C}$ can be formalized as a so-called *orchestration tree* $T(O)$.

- The root ε of the tree represents the fact that no action has been executed yet.
- Each node x in the orchestration tree $T(O)$ 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 $I \in [0..n]$ [2] (0 stands for the client and $1, \ldots, n$ stand for the e-Services $A_1, \ldots, A_n$, respectively), $T(O)$ contains at most one successor node $x{\cdot}(a, I)$.

- Some nodes of the orchestration 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 $T(O)$ 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 an orchestration tree $T(O)$ and a path p in $T(O)$ starting from the root, we call the *projection* of p on an e-Service A_i the path obtained from p by:

- removing each edge whose label (I, a, S) is such that $i \notin \{I\} \cup S$, and collapsing start and end node of each removed edge;

- replacing, for each edge labeled by (I, a, S), with $I = i$, the label with $a^{\gg}$;

- replacing, for each edge labeled by (I, a, S), with $i \in S$, the label with ${}^{\gg}a$.

We say that an orchestration O is *coherent* with a community $\mathcal{C}$ if for each path p in $T(O)$ from the root to a node x and for each e-Service A_i of $\mathcal{C}$, the projection of p on A_i is a path in the execution tree $T(A_i)$ from the root to some node y, and moreover, if x is final in $T(O)$, then y is final in $T(A_i)$.

We define as *client specification* a specification of the orchestration tree that the client would like to have realized using the e-Services in the community. Of the orchestration tree the client only specifies the actions, and whether it is the initiator of an action or not. Notably, we allow for incomplete information on the tree specified by the client. In other words the client may underspecify the sequences of actions of which the client is not the initiator, allowing the orchestrator to fill in the details left unspecified. Moreover the client may allow for don't care nondeterminism in the specification as shown below.

In this paper we consider underspecified specifications that can be expressed using a finite number of states. Formally, the client specification is 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;

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

- $F_0 \subseteq S_0$ is the set of final states.

Observe that the nondeterministic FSM $\mathcal{A}_0$ specifies a set $\mathcal{T}(\mathcal{A}_0)$ of orchestration trees, and the client requires the orchestrator to realize one (any one) among such trees. Specifically, each orchestration 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 orchestration 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 orchestration is free to specify a finite sequence of interactions initiate by whatever e-Service except for the client;
- choosing for each edge a set of servants, and adding it to the label of the edge.

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

Given a community $\mathcal{C}$ of e-Services, and a client specification A_0, the problem of *composition existence* is the problem of checking whether there exists an orchestration that is coherent with $\mathcal{C}$ and that realizes A_0. The problem of *composition synthesis* is the problem of synthesizing an orchestration that is coherent with $\mathcal{C}$ and that realizes A_0.

Since we are considering e-Services that have a finite number of states, we would like also to have an orchestration 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.

Figure 2 shows a possible client specification $\mathcal{A}_0$, which specifies that the client would like to act as initiator of a `user_authentication` action, followed by a `search_greeting_card_&_select` action. At this point the client allows the orchestration to act in a way not requiring interaction with the client itself, and then is interested to act as initiator of a `complete_&_send` action.

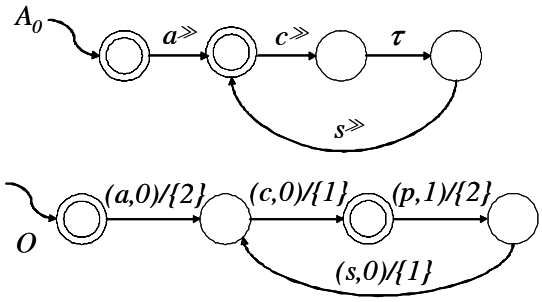

Fig. 2: Client specification and orchestration

O is the MSFM[3] of an orchestration coherent with the e-Services of Example 1 and realizing the client specification $\mathcal{A}_0$. The orchestration specifies the client as initiator of the action `user_authentication` with A_1 as servant, then specifies the client as initiator of the action `search_greeting_card_&_select` with A_2 as servant; at this point, the orchestration specifies A_1 as initiator of the action `pay` served by A_2 (note that, correctly, the client is not involved, as specified by the τ action in $\mathcal{A}_0$), and finally the orchestration specifies the client as initiator of the action `complete_&_send` with A_1 as servant.

4 Composition Technique Based on DLs

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 concept in a knowledge base expressed in the well known Description Logic (DL) $\mathcal{ALCQ}_{reg}$ [7].

4.1 The Description Logic $\mathcal{ALCQ}_{reg}$

In $\mathcal{ALCQ}_{reg}$, starting from a set of *atomic concepts* and *atomic roles*, one can build complex concepts and complex roles by applying the *constructs* shown in Figure 3. We also use the following abbreviations to increase readability: $\top$ for $A \sqcup \neg A$, $\bot$ for $\neg\top$, $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$, and $\exists R.C$ for $\neg\forall R.\neg C$.

In DLs, 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 concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$, respecting the conditions specified in Figure 3.

An $\mathcal{ALCQ}_{reg}$ *knowledge base* is a set of assertions of the form $C_1 \sqsubseteq C_2$, where C_1 and C_2 are arbitrary $\mathcal{ALCQ}_{reg}$ concepts without any restrictions. We use $C_1 \equiv C_2$ as an abbreviation for the pair of assertions $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. An interpretation

[3] 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.

Concepts C	**Syntax**	**Semantics**
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
univ. quantif.	$\forall R.C$	$\{o \mid \forall o'.(o,o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
qual. num. restr.	$(\leq n\, P.C)$	$\{o \mid \sharp\{(o,o') \in P^{\mathcal{I}} \mid o' \in C^{\mathcal{I}}\} \leq n\}$
Roles R	**Syntax**	**Semantics**
atomic role	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	$R_1 \circ R_2$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$
refl. trans. clos.	R^*	$(R^{\mathcal{I}})^*$
identity	$id(C)$	$\{(o,o) \mid o \in C^{\mathcal{I}}\}$

Fig. 3: Syntax and semantics of $\mathcal{ALCQ}_{reg}$.

$\mathcal{I}$ *satisfies* the assertion $C_1 \sqsubseteq C_2$ if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. An interpretation is a *model* of a knowledge base $\mathcal{K}$ if it satisfies all assertions in $\mathcal{K}$.

$\mathcal{ALCQ}_{reg}$ 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 knowledge base can be unwound to a (possibly infinite) tree-shaped model (considering domain elements as nodes and role instances as edges). The second is the *small model property*, which says that every knowledge base that admits a model, admits a finite one whose size (in particular the number of domain elements) is at most exponential in the size of the knowledge base itself.

4.2 Composition Synthesis

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 $\mathcal{ALCQ}_{reg}$ knowledge base $\mathcal{K}$ as follows. As set of atomic concepts in $\mathcal{K}$ we have (*i*) one atomic concept s for each state s of A_j, for $j \in [0..n]$, which intuitively denotes that A_j is in state s; (*ii*) atomic concepts F_j, for $j \in [0..n]$, denoting whether A_j is in a final state; (*iii*) atomic concepts served_j, for $j \in [1..n]$, denoting whether (component) FSM A_j is a servant of a transition; (*iv*) atomic concepts $\mathsf{initiated}_j$, for $j \in [0..n]$, denoting whether FSM A_j is a servant of a transition; (*v*) an atomic concept Init representing the initial state of the required service; (*vi*) one atomic concepts a for each action $a \in \Sigma^+$. We have a single role $trans$ in $\mathcal{K}$, such a role will be used to denote state transitions caused by actions. The knowledge base $\mathcal{K}$ is formed by the following assertions.

- For the client specification $\mathcal{A}_0 = (\Sigma, S_0, s_0^0, \delta_0, F_0)$ we assert:
  - $s \sqsubseteq \neg s'$, for all pairs of states $s, s' \in S_0$; these say that atomic concepts representing different states are disjoint.
  - $s \sqsubseteq \bigsqcup_{s' \in \delta_0(s, a^{\gg})} \exists trans.(\mathsf{initiated}_0 \sqcap a \sqcap s')$, for each $a \in \Sigma$ and s with $\delta_0(s, a^{\gg}) \neq \emptyset$; these encode the transitions different from τ.

- $s \sqsubseteq \bigsqcup_{s' \in \delta_0(s,\tau)} \exists R^*_\tau.s'$, for each s with $\delta_0(s,\tau) \neq \emptyset$, where R_τ stands for

$$id(s \sqcap (\leq 1\ trans.\neg \mathsf{initiated}_0)) \circ trans \circ id(\neg \mathsf{initiated}_0)$$

  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 (thus remaining in the same state); the qualified number restriction is used to ensure that there is a single sequence.
- $s \sqsubseteq \forall trans.(\neg a \sqcup \neg \mathsf{initiated}_0)$, for each a such that $\delta(s, a^{\gg})$ is not defined; these say that $a^{\gg}$ is not a possible transition.
- $s \sqsubseteq \forall trans.\mathsf{initiated}_0$, if $\delta(s,\tau)$ is not defined; these say when a τ transition is not possible.
- $F_0 \equiv \bigsqcup_{s \in F_0} s$; this highlights final states of $\mathcal{A}_0$.

- For each component FSM $A_i = (\Sigma, S_i, s^0_i, \delta_i, F_i)$, we assert:
  - $s \sqsubseteq \neg s'$, for all distinct pairs of states $s, s' \in S_i$.
  - $s \sqsubseteq \forall trans.(\neg a \sqcup (\mathsf{served}_i \sqcap s'))$, 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.
  - $s \sqsubseteq \forall trans.(\neg a \sqcup (\mathsf{initiated}_i \sqcap s'))$, 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.
  - $s \sqsubseteq \forall trans.(\neg a \sqcup \neg \mathsf{served}_i)$, for each s and a such $\delta_i(s, {}^{\gg}a)$ is not defined.
  - $s \sqsubseteq \forall trans.(\neg a \sqcup \neg \mathsf{initiated}_i)$, for each s and a such $\delta_i(s, a^{\gg})$ is not defined.
  - $s \sqsubseteq \forall trans.(\mathsf{served}_i \sqcup \mathsf{initiated}_i \sqcup s)$, for each $s \in S_i$; this encodes that when A_i does not participate to an action, it does not change state.
  - $F_i \equiv \bigsqcup_{s \in F_i} s$; this highlights final states of A_i.

- Finally, to encode the general structure of models, we assert:
  - $\top \sqsubseteq (\leq 1\ trans.(a \sqcap \mathsf{initiated}_0))$, for each action $a \in \Sigma$.
  - $\top \sqsubseteq \forall trans.(\bigsqcup_{a \in \Sigma} a)$; to represent that each transition is caused by an action.
  - $\top \sqsubseteq \forall trans.(\bigsqcup_{i \in [1..n]} \mathsf{served}_i)$; to represent that each transition must have some e-Service as servant.
  - $\top \sqsubseteq \forall trans.(\bigsqcup_{i \in [0..n]} \mathsf{initiated}_i)$; to represent that each transition must have an initiator, either an e-Service or the client.
  - $\top \sqsubseteq \forall trans.(\neg \mathsf{initiated}_i \sqcup \neg \mathsf{initiated}_j)$, for each $i, j \in [0..n]$ with $i \neq j$; to represent that transitions have a single initiator (but possibly several servants).
  - $F_0 \sqsubseteq \bigsqcap_{i \in [1..n]} F_i$; this says that when the target e-Service is in a final state also all component e-Services must be in a final state.

- $\mathsf{Init} \sqsubseteq s_0^0 \sqcap \bigsqcap_{i\in[1..n]} (s_i^0)$; to represent that initially all e-Services are in their initial state.
- $\mathsf{Init} \sqsubseteq \bigsqcap_{a\in\Sigma} (\neg a) \sqcap \bigsqcap_{i\in[0..n]} (\neg \mathsf{initiated}_i) \sqcap \bigsqcap_{i\in[1..n]} (\neg \mathsf{served}_i)$; to represent that initially no action has been executed yet.

Theorem 1: The concept Init is satisfiable in the $\mathcal{ALCQ}_{reg}$ knowledge base $\mathcal{K}$, constructed as above, if and only if there exists an orchestration that is coherent with $A_1, \ldots, A_n$ and that realizes the client specification $\mathcal{A}_0$.

Proof (sketch). "$\Leftarrow$" From an execution tree T of $\mathcal{A}_0$ and a composition labeling of T, we can construct a tree-like model of $\mathcal{K}$ such that Init is satisfied in the root.

"$\Rightarrow$" If Init is satisfiable in $\mathcal{K}$, then there exists a tree-like model of $\mathcal{K}$ where Init is satisfied in the root. From such a model, one can derive an execution tree $T(O)$ of an orchestration O, which is coherent with $A_1, \ldots, A_n$ and realizes $\mathcal{A}_0$. □

Observe that, the size of $\mathcal{K}$ is polynomially related to the size of $\mathcal{A}_0$, $A_1, \ldots, A_n$. By the small model property of $\mathcal{ALCQ}_{reg}$, if Init is satisfiable in $\mathcal{K}$, then it is satisfiable in model that is at most exponential in the size of $\mathcal{K}$. From such a model one can extract an orchestration that realizes $\mathcal{A}_0$ by composing $A_1, \ldots, A_n$, and 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 $\{j_1, \ldots, j_\ell\}$;
- $F_c = F_0^{\mathcal{I}}$.

As a consequence of this, we get the following result.

Theorem 2: If there exists an orchestration 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 1, if there exists an orchestration, then Init is satisfiable in the $\mathcal{ALCQ}_{reg}$ knowledge base $\mathcal{K}$ constructed as above. In turn, if Init is satisfiable in $\mathcal{K}$, for the small-model property of $\mathcal{ALCQ}_{reg}$, there exists a model $\mathcal{I}$ of size at most exponential in $\mathcal{K}$, and hence in $\mathcal{A}_0$ and $A_1, \ldots, A_n$. From $\mathcal{I}$ we can construct a MFSM A_c as above. □

By the theorems above and the EXPTIME-completeness of concept satisfiability in $\mathcal{ALCQ}_{reg}$ knowledge bases, we get the following complexity results.

Theorem 3: Checking the existence of a (MFSM) orchestration that is coherent with $A_1, \ldots, A_n$ and that realizes a client specification $\mathcal{A}_0$ can be done in EXPTIME in the sizes of $\mathcal{A}_0, A_1, \ldots, A_n$.

Exploiting reasoning methods for DLs based on model construction, such as tableaux algorithms [5, 8, 2], one can actually construct such a MFSM orchestration. Notice that such algorithms need to be able to deal with reflexive transitive closure, introduced in $\mathcal{K}$ 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 DL systems, such as FaCT [10] and Racer [9, 12], to check existence of a composition, while these systems are not yet usable for the actual construction of the MFSM orchestration, since they do not return the constructed model. A prototype implemented in Java that actually generates a composition using DL-based reasoning (not supporting transitive closure yet) is available[4].

5 Conclusions and Future Work

In this paper we have developed a technique for automatic orchestration synthesis of e-Services, starting from a client specification that allows for incomplete information. The techniques is based on reasoning in DLs.

We observe that in this paper we have not considered compact representations, as those that can be provided by formalisms for reasoning about actions, for finite state machines associated to e-Services and client specifications. If such a compact representation is available, it can easily be exploited to derive an equally compact encoding of composition synthesis in DLs.

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 considered as an initiator of actions, and not as a servant, while an e-Service is seen essentially 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`), in the context of the Workpackage 2 activities.

References

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

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

[4] URL removed to preserve anonymity.