A component-based approach to automated web service composition

Quoc Bao Vo Lin Padgham
School of Computer Science and Information Technology
RMIT University - Australia
{vqbao | linpa}@cs.rmit.edu.au

Abstract

There is great promise in the idea of having web services available on the internet, that can be flexibly composed to achieve more complex services, which can themselves then also be used as components in other contexts. However it is challenging to realise this idea, without essentially programming the composition using some process language such as WS-BPEL or OWL-S process descriptions. This paper presents a mechanism for specifying the external interface to composite and component services, and then deriving an appropriate internal model to realise a functioning composition. We present a conversation specification language for defining interaction protocols and investigate the issue of synchronous and asynchronous communication between the composite service and the component services.

1 Introduction

Web services have been growing enormously in popularity over the last few years, as people see the potential of the world wide web to provide a repository of program components, in much the same way as it currently provides a repository of information pages. A vision of service-oriented architecture is that, upon a user request for service, an appropriate service should be automatically located and invoked to deliver the required service to the user. Moreover, when more than one services are needed to fulfill a user request, a composition mechanism that combines the components to form composite services is also desired.

There have been a number of languages for modelling and describing web services developed as well as frameworks for web service composition. Process modelling languages such as BPEL4WS (or the newer specification WS-BPEL), BPML or WSCI provide concrete ways for composite services to be manually described. There has been significant work on using workflows to support automated composition (e.g. [1, 2] and references therein). One common approach is to map workflows to Petri-nets as a formal model to allow reasoning. Using a similar idea, Narayanan and McIlraith [3] propose a framework for web service composition in which web service descriptions in OWL-S are mapped to Petri-nets to allow formal verification and simulation and to Situation Calculus [4] to allow automatic composition. This approach requires that the services to be composed be atomic. As the process model description of the composite service must be given, such a mechanism should be more precisely described as service orchestration. On the other hand, the problem of synthesising the process model of a targeted composite service has been largely neglected with the notable exception of the work by Berardi et al [5]. As such, two major goals of the work presented in the present paper are to introduce a language for describing conversational services and to achieve a mechanism to synthesise the process models of the composite services.

Several composition frameworks employ finite state machines or transition systems to formally describe either input/output messages or behaviours with environmental preconditions and effects. These include: the message-based approach (a.k.a. the Mealy model), (e.g. [6]) the activity-based approach (a.k.a. the Roman model) (e.g. [5]), and Traverso and Pistore's [7] approach which performs composition using an AI planner (based on a symbolic model checking approach). For a more thorough discussion of the various approaches, the reader is referred to [8]. Most of these approaches aim to describe web services using formal models such as process algebras, Petri-nets and finite state machines, to allow formal properties of the services to be verified. Our aim is rather to provide a simple language to allow the exported behaviours as well as the interface of a web service to be described in a way that will enable us to automatically produce an executable process model that realises a composite service. The description language proposed in our paper takes a major inspiration from a rich literature on component-based software engineering and communication protocols [9, 10].

Finally, it has been noted [11] that many real-world applications require that certain kinds of *session-oriented* web services respond to asynchronous events during their life-cycles [12]. Although it is simpler to describe a synchronous communication semantics, it is important to facilitate well founded asynchronicity.

The main contribution of this paper is a mechanism that allows for automated synthesis of the internal process of a composite service, given the interface description of both component services and the desired composite service. In order to achieve this we also (i) specify a conversation specification language that allows specification of interactions with other entities; (ii) provide a synchronous semantics for the conversations between services; and (iii) provide a framework for asynchronous communication with adaptors between services.

2 Modelling and describing services

In our model a service is an event-driven component whose events are sending and receiving of messages. As such, the description of a service comprises an *external interface*, i.e. the input/output messages that can be exchanged between a service and its clients, and a *system dynamics* specification of the service. The system dynamics itself comprises an interaction protocol, i.e. the sequences of actions to be invoked, to allow the service to converse with its client applications or users, and event descriptions which allow the preconditions and effects of an event to be clearly specified in a first-order logical language.

The *interaction protocol* describes a set of *sequencing constraints*, i.e. legal orderings of messages, by means of a finite-state grammar. The finite-state grammar consists of a set of named states and a set of transitions, one transition for each message that can be received or sent from a particular state. In order to incorporate the event descriptions into the interaction protocol to obtain the system dynamics, we encode the preconditions and the effects of the events to their respective transitions. Formally, a transition is of the form: `<S> : <dir><msg>-> <S>`
where `<S>` is the symbolic name of a state; `<dir>` is the direction of the message which can be either "`!`" (*send*) or "`?`" (*receive*); `<msg>` is the name of a message described in the external interface.

The above description of a service comprises the *external schema* which is made available on the Internet to allow a user or a client application to discover the service and to correctly interact with it.

Example 1 The following gives a ***Banking Service*** specification, describing how a client (of the bank) who has an account with the bank can interact with the service to carry out certain transactions.

```
Service Banking {
 Interface {
  RECEIVE enterPIN(Account acc, EncryptedPIN PIN);
  RECEIVE requestTransfer(Account toAcc, float amount);
  RECEIVE requestBalance();
  SEND invalidPIN();
  SEND authorised();
  SEND overdrawn();
  SEND transactionApproved();
  SEND currentBalance(float balance);
 };
 Protocol {
  States { 0(init,final), 1, 2, 3, 4, 5(final), 6(final) };
  Transitions {
   0 : ?enterPIN         -> 1;
   1 : !invalidPIN       -> 0;
   1 : !authorised       -> 2;
   2 : ?requestTransfer  -> 4;
   2 : ?requestBalance   -> 3;
   3 : !currentBalance   -> 0;
   4 : !transferApproved -> 5;
   4 : !overdrawn        -> 6;
  };
 };
};
```

Figure 1 represents the interaction protocol of the ***Banking Service***:

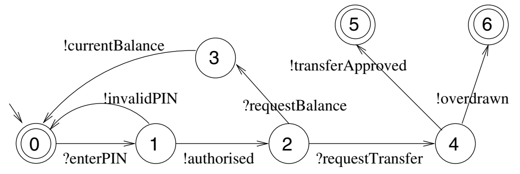

Figure 1. The interaction protocol of a banking service.

2.1 A model for composite services

There are situations in which a client request can not be satisfied by any single available service, but a *composite service* obtained by combining some available services might fulfill such a request. The services used to form a composite service are referred to as *component services*. When an organisation wishes to introduce a composite service based on a collection of existing services, at least two basic tasks need to be accomplished. In the first task, the organisation must produce a specification of how to coordinate the component service to allow the client request to be fulfilled. It is normally required that the specification be executable, i.e. there is an execution engine to execute the specification. Secondly the composite service must be made available as a normal service, i.e. its external schema must be exported and made available on the Internet to allow potential clients to discover and deploy it. It is the former task which is the focus of our current work, though we will also briefly discuss how an external schema is extracted from a composite service at the end of the paper.

Example 2 We now show a simple example of a composite service, namely ***AirTicket Sale Service***, involving the banking service discussed in Example 1 and an ***Airline Service***:

To realise the composite service ***AirTicket Sale Service***, on the one hand an external schema consisting of an interface and an interaction protocol with messages such as offer(), payment(), transactionNotApproved(), etc. must be exported and made available to the clients. On the other hand, a mechanism to coordinate the component services

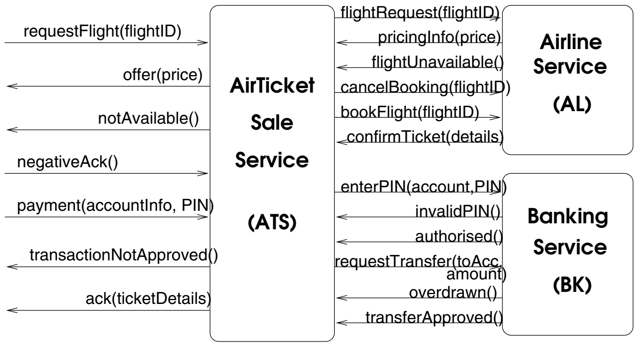

Figure 2. A simple composite service for on-line Air Ticketing.

Airline Service and ***Banking Service*** must be introduced to allow the functionalities of the composite service to be correctly achieved. This is known as the problem of *composition synthesis* which is concerned with producing a specification of how to coordinate the existing services to realise the functionality of a desired composite service.

2.2 Problem formalisation

Gerede et al [13] formalise the activity-based composition synthesis problem which was originally proposed by Berardi [8] based on finite state automata whose transitions between states are labelled by activities. The problem of message-based service composition has been discussed in the work of Hull et al [6, 14]. Given a set of available services whose external schemata are made available, we would like to construct a composite service that meets certain criteria. While there could be several way in which such criteria could be expressed, we will require that the external schema of the composite service be provided. This leads us to a similar starting point to that of Gerede et al's [13] formalisation of the activity-based composition synthesis problem, *viz.* the composition system.

Definition 1 A *composition system* $\mathcal{C}$ is a pair $(S^T, \mathcal{S})$ where S^T is the external schema of the *target* (or, *desired*) service to be composed and $\mathcal{S} = \{S^1, \ldots, S^n\}$ is a set of external schemata specifying the available component services to be used in the composition of the desired service.

Essentially, in a composition system $\mathcal{C}$, the set of component services is fixed with their external schemata required to be fully specified. We require that the target service to be composed also be clearly specified in terms of its input/output messages and its interaction protocol. It is the task of a composer to construct a mechanism to coordinate the component services so that the specification of the target services is satisfied. We henceforth refer to this mechanism as the internal model of the target composite service. For convenience, we introduce the following notations: Given the external schema S of a service, $Messages(S)$ denotes the set of messages declared in the interface of S, and let $\tau \in Transitions(S)$, the function $Dir(S, \tau)$ will be ? if τ is a receiving message in S and will be ! otherwise. We will also write a transition as $s : m \rightarrow s'$ where $s, s' \in States(S)$ and $m \in Messages(S)$.

The internal model M of a composite service will have both message based transitions, with which it communicates with component services and the user, and also internal transitions to allow control of internal processing to capture the required business logic of the service.

Definition 2 A *realisation* of the composite service S^T, within a composition system $\mathcal{C} = (S^T, \mathcal{S})$, is a finite state machine (FSM) $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, Σ denotes the set of transitions, $\delta : Q \times \Sigma \rightarrow 2^Q$ is the transition function, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ is the set of *final states*. The realisation M of a composite service S^T is required to satisfy the following conditions:

1. $\Sigma = comm^M \cup trans^M$ where $comm^M = \{<?\,|\,!>m : m \in msg^M\}$ is the set of communicative acts such that $msg^M \subseteq Messages(S^T) \cup \bigcup_{S \in \mathcal{S}} Messages(S)$, and $trans^M$ consists of the set of transitions denoting the internal computations of the composite service.

2. There is a surjective function $\imath : Q \rightarrow States(S^T)$ such that: (i) $\imath(q_0) = init_{S^T}$, and (ii) for each transition $(u : m \rightarrow v) \in Transitions(S^T)$, there exist two states $q, r \in Q$ such that $\imath(q) = u$ and $\imath(r) = v$, and $q \xrightarrow{\chi^*} r$, where χ^* is a sequence of transitions from $\Sigma \setminus Messages(S^T)$ such that m occurs in χ^*.

Moreover, in order for the composite service to behave correctly, certain properties need to be guaranteed.

Definition 3 *Let S be an external schema. The complement schema of S, denoted by $\mathbb{C}(S)$, is defined as follows: (i) there is an isomorphism $\jmath$ between $States(S)$ and $States(\mathbb{C}(S))$; and (ii) $s : ?m \rightarrow s' \in Transitions(S)$ iff $\jmath(s) : !m \rightarrow \jmath(s') \in Transitions(\mathbb{C}(S))$, and $s : !m \rightarrow s' \in Transitions(S)$ iff $\jmath(s) : ?m \rightarrow \jmath(s') \in Transitions(\mathbb{C}(S))$.*

A *composition state* for a composition system $\mathcal{C} = (S^T, \{S^1, \ldots, S^n\})$ with a realisation $M = (Q, \Sigma, \delta, q_0, F)$ is a tuple $\langle s^1, \ldots, s^n, s^{cT}, q\rangle$ where $s^i \in States(S^i)$ $(i = 1, \ldots, n)$, $s^{cT} \in \mathbb{C}(S)$, and $q \in Q$. An *execution trace* over $(\mathcal{C}, M)$ is a (possibly infinite) sequence $\sigma_0 \rightarrow_{m_1} \sigma_1 \rightarrow_{m_2} \sigma_2 \rightarrow \ldots$, where

- each σ_i is a composition state for $(\mathcal{C}, M)$;

- $\sigma_0 = \langle init_{S^1}, \dots, init_{S^n}, \jmath(init_{S^T}), q_0\rangle$, and
- $\sigma_{i+1} = \langle s^1_{i+1}, \dots, s^n_{i+1}, s^{cT}_{i+1}, q_{i+1}\rangle$ iff $\sigma_i = \langle s^1_i, \dots, s^n_i, s^{cT}_i, q_i\rangle$, and there exists $\kappa \in \{1, \dots, n, cT\}$ such that (i) $\tau = (s^\kappa_i : m_i \to s^\kappa_{i+1}) \in Transitions(S^\kappa)$; (ii) $q_{i+1} \in \delta(q_i, !m_i)$ if $Dir(S^\kappa, \tau) = ?$ and $q_{i+1} \in \delta(q_i, ?m_i)$ otherwise; and (iii) for each $\ell \in \{1, \dots, n\}$ such that $\ell \neq \kappa$, $s^\ell_{i+1} = s^\ell_i$.

By definition, $Exec\_Tree(\mathcal{C}, M) = \{\sigma : \sigma$ is an execution trace over $(\mathcal{C}, M)\}$. $Exec\_Tree(\mathcal{C}, M)$ is a tree whose root is the initial composition state σ_0. Two desirable properties of a realisation of a composition system $\mathcal{C}$ are *freedom of deadlock* and *freedom of unspecified receptions*.[1]

Intuitively, a composition system $\mathcal{C}$ with a realisation M has no unspecified receptions if and only if (i) whenever an execution trace σ can reach a point where the realisation M is in a state where it can send a message m then some service from the composition system $\mathcal{C}$, be it a component service or the constructed composite service, will be in a state where it can receive that message, and (ii) whenever an execution trace σ can reach a point where a service from the composition system $\mathcal{C}$ is in a state where (ii.a) it can send a message m and (ii.b) it can not receive any message, then the realisation M will be in a state where it can receive that message.

We can show that there is a poly-time algorithm to check whether a realisation M of a given composition system $\mathcal{C}$ is free of deadlock and free of unspecified receptions.

3 Asynchronous communication

Under our formulation of the synchronous semantics, the finite-state machines describing the protocols of the component services and the composite service on the one side and the internal model of the composite service on the other side are required to advance atomically. That is, when a message m is sent, one side must be in a state that enables it to send m and the other side is in a state that enables it to receive m. Hence, the finite-state machines describing one of the components and the internal model advance synchronously, so that the sending and receipt of a message are considered an atomic action under this abstraction. However, as discussed by Yellin and Strom [9], the synchronous semantics can be implemented without requiring the components to send and receive messages atomically. The only requirement is that the communicating components always agree on the execution trace, i.e. the order of messages sent and received.

While the synchronous semantics significantly simplifies the reasoning about communicating systems, in particular, composite services and their internal models, this restriction may severely hamper the applicability of our model to most application domains in which services are required to be dynamically discovered and plugged in to obtain the composite services. The standard way to achieve asynchronism is to use unbounded memory to store the parameters sent from one component to another without requiring the sending component to halt its process to wait for its mate to receive the messages it sends. Although the asynchronous semantics are easier to implement in comparison to the synchronous semantics, it is hard to reason about systems of communicating components under these semantics. In general, properties of the system such as deadlock or existence of unspecified receptions are undecidable [15].

Given a composition system $\mathcal{C} = (S^T, \{S^1, \dots, S^n\})$ and a realisation $M = (Q, \Sigma, \delta, q_0, F)$ for $\mathcal{C}$, properties of the composite service embodied by M and $\mathcal{C}$ such as deadlock, unspecified receptions, etc. can be investigated by considering the product automaton constructed from $S^1, \dots, S^n$, and S^T and M. This is the approach taken by, e.g. Gerede et al. [13]. In our representation, we will have to take into account not only the interaction states the FSMs are in, but also the state of the FIFO channels containing the (asynchronous) messages exchanged between different components of the system.[2] A *configuration* is a pair $\langle \mathcal{I}, \mathcal{M}\rangle$ where

- $\mathcal{I}$ denotes the (global) interaction state, called *i-state*, of the production machine constructed from $S^1, \dots, S^n$, and S^T and M;
- $\mathcal{M}$ denotes the (communication) medium state, call *m-state*, of the composite service.

There will be n duplex FIFO channels to allow messages between the components $S^1, \dots, S^n$ and the realisation M to be stored. We denote the content of the queue storing the messages from M to S^i (resp. from S^i to M) by ω^i (resp. $\overline{\omega^i}$). The symbol ϵ denotes the empty sequence. Finally, an i-state is a tuple $(s^1, \dots, s^n, q)$ where $s^i \in S^i$ $(i = 1, \dots, n)$ and $q \in Q$.

Without loss of generality we will assume that the sets of messages of $S^1, \dots, S^n$ and S^T are disjoint. A transition from one configuration $c = \langle (s^1, \dots, s^n, q), (\omega^i, \overline{\omega^i})^n_{i=1}\rangle$ to another configuration c' is labelled by α which is either an activity from the set of activities Σ of the realisation M or a message from $\bigcup^n_{i=1} Messages(S^i)$ satisfying the following conditions:

1. If $\alpha = \, !m$ and $m \in Messages(S^k)$ for some $k \in \{1, \dots, n\}$, then

(a) If $(s^k : \alpha \to w^k) \in Transitions(S^k)$, then (i) the i-state of c' is the same as that of c except for s^k being

[1]The notions of freedom of deadlock and freedom of unspecified receptions was first introduced by Brand and Zafiropulo [15].

[2]Another attribute of a configuration is the state of the world, e.g. the content of databases, ticketing and reservation systems, etc. This will be part of the future work to extend the framework introduced in this paper.

replaced by w^k; and (ii) the content of the FIFO queue from S^k to M is updated by $m.\overline{\omega^k}$ (was $\overline{\omega^k}$ before the update).

(b) If $q' \in \delta(q, \alpha)$, then (i) the i-state of c' is the same as that of c except from q being replaced by q'; and (ii) the content of the FIFO queue from M to S^k is updated by $m.\omega^k$.

2. If (i) $\alpha = ?m$ and $m \in Messages(S^k)$ for some $k \in \{1, \ldots, n\}$, (ii) $(s^k : \alpha \to w^k) \in Transitions(S^k)$, and (iii) the content of the FIFO queue from M to S^k is $m.\omega$ for some sequence of messages ω, then (a) the i-state of c' is the same as that of c except from s^k being replaced by w^k; and (b) the updated content of the FIFO queue from M to S^k becomes ω.

3. If (i) $\alpha = ?m$ and $m \in Messages(S^k)$ for some $k \in \{1, \ldots, n\}$, (ii) $q' \in \delta(q, \alpha)$, and (iii) the content of the FIFO queue from S^k to M is $m.\omega$ for some sequence of messages ω, then (a) the i-state of c' is the same as that of c except from q being replaced by q'; and (b) the updated content of the FIFO queue from S^k to M becomes ω.

4. Otherwise, the system reaches the special configuration, denoted by error.

The above definition of configurations constitutes the states of the production FSM of a composition system $\mathcal{C}$ and its realisation. Based on this FSM and its reachability graph, standard notions such as deadlocks and unspecified receptions can be defined. Intuitively, error configurations allow a system designer to mark down all execution traces that lead to deadlock or an unspecified reception.

4 Conclusion and future work

The approach to the problem of Web service composition introduced in this paper is based on a rich literature in component-based software engineering. Our composition algorithm requires service specifications of the component services and the target composite service. We further require that a set of parameter mapping rules be provided to allow messages to be sent by the internal models of the composite service to be synthesised. We examine the conditions under which a configuration (i.e. the state of the production FSM) is an error configuration. The algorithm has then been constructed in such a way that error configurations and execution traces leading to error configurations are eliminated.

There are several directions we could pursue to extend the framework presented in this paper: (i) integrating the preconditions and effects of a service to its protocol specification; and (ii) supporting exception handling for composite service execution. An interesting and ambitious research direction would be to allow the description of the target composite service to be under-specified. Then, based on the available component services combined with some ontological reasoning, the system could construct an internal computational model which accomplishes some high-level task, subsequently enforcing a fully specified interface and interaction protocol for the composite service.

References

[1] W. M. P. van der Aalst et al. "Adaptive workflow-on the interplay between flexibility and support," in *Intnl. Conference on Enterprise Information Systems*, 1999, pp. 353–360.

[2] S. Lu, "The semantic correctness of transactions and workflows," Ph.D. dissertation, Computer Science Dept., SUNY at Stony Brook, USA, 2002.

[3] S. Narayanan and S. A. McIlraith, "Simulation, verification and automated composition of web services," in *Procs. WWW '02*. ACM Press, 2002, pp. 77–88.

[4] R. Reiter, *Knowledge in Action*. Cambridge, Massachussets: MIT Press, 2001.

[5] D. Berardi, D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of E-services That Export Their Behavior," in *Procs. of Service-Oriented Computing - ICSOC 2003*. Springer-Verlag, 2003, pp. 43–58.

[6] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation specification: a new approach to design and analysis of e-service composition," in *Procs. of WWW '03*. ACM Press, 2003, pp. 403–410.

[7] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in *Proceedings of International Semantic Web Conference 2004*. Springer-Verlag, 2004, pp. 380–394.

[8] D. Berardi, "Automatic service composition: Models, techniques and tools," Ph.D. dissertation, Università di Roma, "La Sapienza", 2005.

[9] D. M. Yellin and R. E. Strom, "Protocol specifications and component adaptors," *ACM Trans. Program. Lang. Syst.*, vol. 19, no. 2, pp. 292–333, 1997.

[10] K. Saleh, "Synthesis of communications protocols: an annotated bibliography," *SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 5, pp. 40–59, 1996.

[11] R. Breite, P. Walden, and H. Vanharanta, "C-commerce virtuality - will it work in the internet?" in *Proc. of SSGRR 2000*, 2000.

[12] V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong, "Beyond discrete e-services: Composing session-oriented services in telecommunications," in *TES '01*, pp. 58–73.

[13] C. E. Gerede, R. Hull, O. H. Ibarra, and J. Su, "Automated composition of e-services: lookaheads," in *Procs. ICSOC '2004*, pp. 252–262.

[14] R. Hull, M. Benedikt, V. Christophides, and J. Su, "E-services: a look behind the curtain," in *Procs. PODS '2003*, pp. 1–14.

[15] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *Journal of the ACM*, vol. 30, no. 2, pp. 323–342, 1983.