Handling Transactional Properties in Web Service Composition

Marie-Christine Fauvet[1], Helga Duarte[1], Marlon Dumas[2], and Boualem Benatallah[3]

[1] Joseph Fourier University of Grenoble, CLIPS-IMAG Laboratory, **
BP 53, 38041 Grenoble Cedex 9 - France
{Marie-Christine.Fauvet,Helga.Duarte}@imag.fr
[2] Centre for Information Technology Innovation, QUT
GPO Box 2434, Brisbane QLD 4001, Australia,
m.dumas@qut.edu.au
[3] School of Computer Science and Engineering, UNSW
Sydney 2052, Australia
boualem@cse.unsw.edu.au

Abstract. The development of new services by composition of existing ones has gained considerable momentum as a means of integrating heterogeneous applications and realising business collaborations. Services that enter into compositions with other services may have transactional properties, especially those in the broad area of resource management (e.g. booking services). These transactional properties may be exploited in order to derive composite services which themselves exhibit certain transactional properties. This paper presents a model for composing services that expose transactional properties and more specifically, services that support tentative holds and/or atomic execution. The proposed model is based on a high-level service composition operator that produces composite services that satisfy specified atomicity constraints. The model supports the possibility of selecting the services that enter into a composition at runtime, depending on their ability to provide resource reservations at a given point in time and taking into account user preferences.

1 Introduction

Web services constitute a rapidly emerging technology that promises to revolutionise the way in which applications interact over the Web. Established organisations are discovering new opportunities to conduct business by providing access to their enterprise information systems through Web services. This trend has led to a paradigm known as Service-Oriented Computing (SOC) wherein information and computational resources are abstracted as (Web) services which are then interconnected using a collection of Internet-based standards.

** H. Duarte is financially supported by the European Union Programme of High Level Scholarships for Latin America (id. E03D13487CO) and the National University of Colombia.

In this setting, the development of new services by composition of existing ones has gained considerable momentum as a means of integrating heterogeneous applications and realising business collaborations. The execution of a service obtained by composition, also known as a *composite service*, involves a series of interactions with the underlying *component services* in order to access their functionality. The logic of a composite service (also known as the *orchestration* model) determines, among other things, the interactions that need to occur, the order of these interactions, and the escalation procedures that should be triggered under specified exceptional situations (e.g. faults arising during the interactions). Importantly, the components of a composite service retain their autonomy, that is, they are free to enter into interactions with other services (whether composite or not).

Some services, for example in the area of electronic commerce, possess inherent transactional properties [1]. This is the case in particular of services associated with the management of resources with limited capacity (e.g. accommodation booking), the management of shared physical or human assets (e.g. short-term rental of equipment, hiring of professional services), or the sales of goods where demand tends to exceed supply (e.g. ticket sales services). In principle, these transactional properties can be exploited during service composition in order to fulfil constraints and preferences specified by the designer or the user of the composite service. At present however, the language and platforms supporting the development of transactional applications over (Web) services do not provide high-level concepts for : (i) expressing the transactional properties that composite services are required to fulfil; and (ii) automatically ensuring that these properties are fulfilled by exploiting the transactional properties of the underlying component services.

The aim of the work reported in this paper is to propose a model for the composition of Web services with transactional properties that takes into account the above requirements. Specifically, the paper focuses on ensuring atomicity properties. The main contribution of the paper is the definition of a service composition operator that exploits the atomicity properties and reservation functionality of the component services in order to ensure atomicity properties at the level of the resulting composite service. In particular, the operator supports the modelling of minimality and maximality constraints over the set of services entering into a transaction in the context of an execution of a composite service. Using the proposed model, it is possible to capture constraints such as "between X and Y component services must execute successfully up to completion or else no service must execute successfully up to completion" (i.e. up to the point where the underlying resource has been assigned to the composite service). For example, given four services A, B, C and D, the user of a service obtained by composition of these four services can specify that at least 2, but not more than 3 services among {A, B, C, D} must complete or none must complete, by setting the minimum and maximum bounds to 2 and 3 respectively. In addition, the proposed operator is parameterised so as to allow the end user to impose that certain domain-specific constraints and preferences are satisfied by the executions of the composite service.

The execution model underpinning the proposed composition operator relies on the *Tentative Hold Protocol* (THP) [2] to tentatively reserve the resources managed by the component services, as well as a variant of the Two-Phase Commit (2PC) protocol to perform the transaction once all the tentative reservations have been obtained. Note that this is not a fundamentally restrictive assumption, since tentative holds, as opposed to definite reservations do not force the service to lock a resource for a given client. So regardless of its atomicity properties (or absence thereof) the operations associated to THP can be implemented on any given service. This combination maximises the chances of achieving a successful transaction while avoiding blocking resources for relatively long periods of time. In addition, the execution model incorporates a runtime service selection mechanism, so that if a given component service is not able to provide a given resources, it can be replaced by another one among a set of candidate component services. This selection takes into account the constraints and preferences provided as parameter to the composition operator.

The remainder of the document is structured as follows. In Section 2 we frame the problem addressed in this paper and discuss related work. A scenario illustrating the proposed approach is then presented in Section 3. In Sections 4 and 5 we introduce the service composition model and the underlying execution model respectively. Finally, Section 6 draws some conclusion and discusses directions for future work.

2 Motivation and related work

The execution of composite services with transactional properties is generally based on long-lived complex distributed transactions that may involve compensation mechanisms. A significant number of transactional models have been proposed in various contexts, including database systems, cooperative information systems, and software configuration management [4–7]. However, these models are not designed to address the following combination of challenges posed by the environment in which (Web) services operate:

Heterogeneity of transactional properties. Whereas it is reasonable to assume that certain services will provide the operations required to participate in 2PC procedures and will expose these operations in a standardised manner as part of their interfaces, it is not realistic to expect that all services will do so. Some services will simply not provide transactional properties, or will provide them but not to the same level and in the same way as others. This may be due to purely technical issues (e.g. a given service provides operations corresponding to those required by 2PC but in a non-standard manner), or it may result from the choice of business model (e.g. the service provider is not willing to let clients lock its resources) or it may ensue from the inherent nature of a service (e.g. the notion of locks does not make sense for a given service). In addition, certain services will not offer the operations associated with 2PC, but may will instead offer compensation operations, possibly with associated costs like in the case of services for which the rights can be rescinded with associated penalties.

Limited duration of reservations and cost of compensation. Even when a service does offer operations allowing it to participate in 2PC procedures (or when it offers compensation operations), it is crucial to minimise the duration between the two phases of the 2PC protocol (or to minimise the need for compensations to avoid the associated costs). In the field of electronic commerce for example, certain services only allow one to lock resources for a very limited period of time. This raises coordination problems when the number of services involved in a transaction is high and they need to be synchronised.

Need for dynamic service selection support. The choice of services that will complete a given transaction may be performed dynamically (during the execution of the transaction) and hence it is not realistic to assume that this information will be known in advance. This is partly due to the fact that several services may offer the same functionality (or rather *capability*). Thus, when the execution of a given service fails during the first phase of the 2PC protocol, it is possible to replace it with another service providing the same capability so that the transaction can still proceed.

It is commonly accepted that traditional approaches for achieving transactions with ACID (*Atomicity, Consistency, Isolation, Durability*) properties are not suitable for long-running transactions like those found in the area of Web services, since it is not acceptable to lock a given resource managed by a service while work is being performed in parallel by other services and/or until locks for the resources managed by other services (which may end up refusing to grant these locks) are obtained. In addition, the 2PC protocol which is commonly used in distributed systems is not applicable in the context of composite service execution since this protocol assumes that all the partners participating in a transaction support the operations of preparation and validation essential to this protocol, and this assumption does not generally hold in the case of Web services as explained in the previous section. Furthermore, given the competing/substitutable (i.e. one or the other) relations that may exist between services that provide the same capability, it is appropriate not to restrict atomicity to the traditional all-or-none property, but instead to consider minimality and maximality constraints over the set of services that participate in a transaction as part of the execution of a composite service. Finally, integration issues need to be considered since the services participating in a composition may potentially each rely on a different type of transaction management system and/or expose its transactional operations in a different manner. This latter aspect is the main motivation for protocols such as *WS-Coordination* [8], *WS-AtomicTransaction* [9] which aim at defining standardised ways for services to interact with transactional coordinators in order to set up, join, and validate transactions.

In this paper we address the issues identified above by proposing a high-level operator that allows designers to compose sets of services (with or without transactional properties) and to specify atomicity constraints of the form *between X* *and Y component services must validate (i.e. complete their execution up to the validation phase), or else none of them should validate.* Such constraints are relevant in a broad range of applications, and in particular in electronic commerce applications as motivated in [10]. In addition to being parameterised by minimality and maximality constraints (i.e. variables X and Y in the statement above), the operator admits as parameters, restriction and preference functions that guide the (runtime) selection of services that are to participate in the selection of component services (among the pool of possible component services) as well as the replacement of services when some of the selected services fail to arrive up to the validation phase (e.g. because the resource that they manage is unavailable under the constraints captured by the composition).

The execution model associated with this composition operator relies on the THP protocol [2] to acquire tentative reservations on the resources managed by the component services and thus increase the probability of completing the transaction successfully with the subset of component services initially selected. A variant of the 2PC protocol is then used to complete the transaction. This variant takes into account the fact that some services do not provide the operations required by the 2PC protocol, i.e. services that do not provide atomicity properties, services for which atomicity is irrelevant, and services that offer compensation operations rather than supporting definitive reservations.

The THP protocol is intended to facilitate the automated coordination of multi-process business transactions. It is an open protocol operating over a loosely coupled architecture based on message exchanges between processes prior to a transaction. The idea is to allow a process to register an intention to acquire a resource managed by a service (i.e. a *tentative hold*) and from that point on, to be notified of changes in the availability of this resource. When the resource is definitely acquired by a process, a message is sent to all other registered processes to inform them that the resource is no longer available. This protocol thus allows processes to maintain up-to-date information regarding the ability for a set of resource management services to enter into a given transaction, thus maximising the chances that the transaction is successfully completed.

The idea of separating between a tentative reservation phase and a "definitive" reservation phase has previously been considered in the area of distributed component transactions. In [11] for example, a protocol similar to THP is employed during the so-called *negotiation* phase while the 2PC protocol is used to complete the transaction. This prior proposal differs from ours in at least two ways. Firstly, the proposed approach takes into account services that do not provide the operations required by the 2PC protocol. For a number of reasons as outlined above, such *non-transactional* services are numerous in the area of electronic commerce and business process management which are prominent application areas of SOC [12, 1]. Because of this, we do not use the traditional 2PC protocol but rather an extension of it. Secondly, our model allows designers to express maximality and minimality constraints over the set of services that are expected to go through the validation phase, as well as restriction and preference functions that are used to perform dynamic service selection. In contrast, the model put forward in [11] is limited to all-or-none transactions and does not take into account dynamic selection of the components entering into a composition.

The same comparative remarks apply to previously proposed web service transaction models based on THP and 2PC such as the one proposed in [13].

In [14] the authors present a multi-level model for service composition that covers the specification of composite services from the user-interface level of abstraction down to the implementation of transactional operations. However, this approach supports only “all or none” transactions whereas our model supports minimality and maximality constraints as discussed above. On the other hand, the model in [14] supports ”all or none” executions of composite services that include sequentially ordered tasks. To achieve this, the model in [14] distinguishes tasks with different transactional characteristics, namely atomic, compensable and pivot tasks as defined in [12] and characterises a class of sequential arrangements of such tasks for which all or none execution can be guaranteed.

Another related work is [15], where a workflow-based approach is proposed to deal with workflow flexibility issues. This approach has a different scope than ours as it relies on transactions to enforce the atomicity of a set of modifications to be applied to a workflow specification.

3 Scenario

As a working and motivating example, we consider a scenario that arises when one wants to organise a meeting to be held at a given day. Potential participants must be invited, a room must be booked and catering must be arranged. To keep the example simple we assume the meeting to last only one day.

The organiser of the meeting determines that the meeting should be organised according to the following rules, which capture either restrictions or preferences:

- Restrictions: the person X should participate, consequently she has to be available at the given day, otherwise the meeting is postponed. To make sense, a minimum number of persons must participate (at least min but not more than max). Eventually the global cost (catering and room cost) cannot exceed a given budget.
- Preferences: the number of participants should be maximised and the global cost minimised.

For the sake of simplicity, both catering and room are arranged for an average number of people ((min+max)/2).

The organisation of the meeting succeeds only if all restriction rules given above are met, otherwise the meeting is cancelled. In addition, the organisation is done in a way that satisfies the preferences as much as possible.

To model this example, we define Meeting Organisation (MO in short) as a composite service type that captures the process of organising meetings as described above. MO relies on three other processes, each of which is modelled by a service type: Room for booking function rooms and Caterer for arranging food and drinks. The process of contacting participants to arrange a meeting is also modelled as a composite service type, namely Participant Invitation (PI in short).

The type PI results from the composition of service types each of which is associated to a potential participant's diary. Let pi be a service of type PI: services that compose pi have been selected in order to ensure pi to satisfy the restriction rules associated with it. pi's executions are performed as follows: pi processes requests to diary services hoping to obtain a successful booking at the given day. We assume here, that all diary services support the operations needed to manage such booking. When pi interacts with its service components, it does so according to their transactional properties and in a way that satisfies as much as possible the preferences defined at composition time by the designer. The outcome of this interaction process is to build a suitable set of diary services, such that each service in this set provides an operation to perform a booking on a given day (and possibly also other operations). If the resulting set contains X's diary and if its size is greater than min and less than max, then the service pi can complete successfully, otherwise it fails.

The service mo of type MO, responsible for organising the meeting, coordinates the execution of its component services: pi of type PI, r of type Room and c of type Caterer. r and c successfully complete only if they could be booked at the given day and if the total cost is less than the budget decided at design time. r and c have been chosen in order to minimise expenses.

4 Composition Model

This section introduces a composition model which aims at providing features for composing services and associating transactional properties to the resulting composite services. The proposed model handles the transactional properties of a composite service according to those exposed by the component services.

Our aim is not to define yet another model for Web service composition or another language for service description, but rather to abstract usual concepts used in many approaches (see for example [16–18]) and standards (see for example [19, 9, 20]). We introduce here an abstract notation that will enable some form of reasoning on transactional properties of Web services. In Section 4.1 we introduce an abstract data type-based notation for the purpose of Web service modelling. In Section 4.2 we then study a composition operator. Finally, Section 4.3 discusses transactional properties of services resulting from a composition.

4.1 Service Design

In the study reported here, we focus on Web services that offer resources to clients (e.g. flight tickets that can be purchased by customers). Interactions between the service that offers a resource and the clients interested in it, are usually encapsulated within a transaction. In the following we intend to characterise a Web service according to the properties of the transactions it can handle [12]:

– A service is said to be **atomic** (e.g. associated with the "all or nothing" semantics) when it provides the following operations: resource reservation (equivalent to the preparation phase of the 2PC protocol), cancellation, and validation.
– A service is **quasi-atomic**, when it supports a validation operation (i.e. an operation that performs the work involving the resource such as booking a flight seat), and a compensation function which undoes the effect of the validation (e.g. releasing the flight seat and applying a penalty).
– A service is defined as **non-atomic**, when the only operation it offers on resources is validation. It does not support neither (definite) reservation nor cancellation nor compensation.

In the sequel, we formally capture the notion of service and other related notions by describing a set of abstract types and operations. We adopt a functional style as it gives a high level and syntax-independent framework to describe types and operations. The following notations are used: T1 $\longrightarrow$ T2 stands for the type of all functions with domain T1 and range T2. {T} denotes the type of sets of T. The following functional specification introduces the Service abstract data type (ADT) and its elementary selectors.

type Service
  {*The type Service models an offer of a set of operations (e.g. books sells and buys). An instance of type Service models a software entity capable of providing these operations. Clients can access this entity using the operations it offers.*}
type Client
  {*An instance of the type Client models a customer (whether it is a person, an organisation or a service instance) who, after appropriate authentication, is allowed to access service instances.*}
property: Service $\longrightarrow$ {atomic, quasi-atomic, non-atomic}
  {*property(s) returns the transactional characteristic of service s (see Section 2).*}

With these conventions, the working example (see Section 3) can be formalised as follows:

Caterer: sub-type of Service
  {*An instance of Caterer represents an enterprise providing catering functions.*}
cost : Service, Resource, Client $\longrightarrow$ real
  {*cost(s, r, c) returns the price the customer c has to pay to purchase the resource r supplied by s.*}
Room: sub-type of Service
  {*An instance of Room represents an enterprise that proposes function rooms for rent.*}
Diary: sub-type of Service
D_1, ..., D_n: sub-types of Diary
  {*A Diary type D_i ($i \in [1..n]$) is associated to each potential participant. We associate a type Diary to each potential participant. Doing so allows us to offer an homogeneous notation, even though there will not be more than a service associated to each type.*}
holder: Diary $\longrightarrow$ string
  {*holder(d) returns the name of d's holder.*}

Below we describe the operations supported by all services. The semantics of these operation is defined according to the transactional properties of the services. The following operations are provided by atomic services:

ReserveR, ReleaseR, ValidateR: Service, Resource, Client $\longrightarrow$ boolean
{*ReserveR (s, r, c) $\Longleftrightarrow$ resource r is reserved by service s for client c (this is a definite reservation as opposed to a tentative hold).*}
{*ReleaseR (s, r, c) $\Longleftrightarrow$ client c has released resource r managed by service s. Pre-condition : ReserveR (s, r, c)*}
{*ValidateR (s, r, c) $\Longleftrightarrow$ client c has definitely acquired resource r managed by s (Note: it is not mandatory for a client to reserve a resource first).*}

The operations provided by services whose executions are quasi-atomic are ValidateR introduced above and CompensateR describes below:

CompensateR: Service, Resource, Client $\longrightarrow$ nil
{*CompensateR (s, r, c) compensates the purchase by client c of resource r managed by service s.*

Finally, non-atomic services provide only one operation, namely ValidateR defined above.

4.2 Parallel Composition Operator

Composite service types can be built by using the parallel composition operator among others. As a first stage, and to focus on transaction issues we chose to limit the scope of this paper to parallel compositions only. Extending our approach to a set of operators (such as condition or sequence) is one of our further studies. Applying this operator to a set of service types with transactional properties defines a new composite service type whose transactional properties are derived from the types of the component services, at composition time.

It is worth noting that in this paper we do not consider sequential composition of services (e.g. dependencies imposing that a service must fully complete before another one can start). In order to support such control dependencies in the context of a transactional service composition model, techniques from the area of long-running database transactions and transactional business processes (see for example the concept of Sagas [21, 12]) would need to be incorporated. Dealing with such issues is outside the scope of this paper and integration in our model of a more complete set of composition operators is left as a direction for future work.

The parallel operator is parameterised by:

– The set of component service types {T1, T2, .., Tn}.
– The range of values for the number of component services which must successfully complete. This range is specified by the minimum and the maximum values (min $\leq$ max $\leq$ n), thereby offering more flexibility for executing the composition and increasing chances for it to successfully complete.
– Two functions: one Boolean function which specifies restrictions (e.g. *X must agree to participate*) and a scoring function that returns, for each potential instance of the composition, the score that the user wishes to maximise (e.g. *the number of participants in the case of the working example*).

To each service type Ti ($i \in [1..n]$) given as a parameter, is associated a set of potential service instances that may vary from an execution to another. For example, let us consider the service instances FrenchTable.com, PizzaSolutions.org and Doyles.org of service type Caterer accessible at a given time. At another time, some might no longer be reachable, and/or others might become accessible.

At an abstract level it is necessary to consider all potential sets of service instances that may enter in the composition. Such a set is called *option*. Let us consider the service instances {t1, t2, ..., tp} (p = 1, ..., n), this set of options is defined as follows (let us assume that type(s) returns the service type associated to s):

(1) $\|\{\text{type(t1), type(t2), ..., type(tp)}\}\| = p \wedge$

(2) $\{\text{type(t1), type(t2), ..., type(tp)}\} \subset \{\text{T1, T2, .., Tn}\}$

Rule (1) expresses that the component service types are distinct e.g. each service type is represented by one service instance. Rule (2) enforces that each service type is one of those given as a parameter. $\|$ E $\|$ denotes the size of E.

The set of options is then reduced to contain only options that evaluate the restriction to true. Eventually, the score is calculated for each option and associated to it.

Summarising the discussion above, the parallel operator is formalised as follows:

Restriction: {Service} $\longrightarrow$ Boolean
Score: {Service} $\longrightarrow$ real
T1, .., Tn : sub-types of Service
// : {T1, .., Tn}, integer>0, integer>0, Restriction, Score, Client $\longrightarrow$ Service
{*Let S = // ({S1, S2, .., Sn}, mi, ma, R, SC, c). S is a service type composed of service types, each belonging to {S1, S2, .., Sn}. Let s be a service of type S: an execution of s may lead to either a success or a failure. In the former case, among all options satisfying R, s is one that maximises SC. s is executed for the client c. mi* $\neq 0$ *and ma* $\geq$ *mi. s is composed of at least mi service instances but not more and ma.* }

By using the operator specified above, the service type Participant Invitation (PI for short) introduced in the working example (see Section 3) can be described as follows:

Let D_1, ..., D_n be the Diary service types of the n potential participants to the meeting. The following statement specifies the process of inviting participants for a meeting scheduled at time d: the person X must agree to attend the meeting, and at least 10 but not more then 15 persons must agree to attend as well:

PI $\longleftarrow$ // ({D_1, ..., D_n}, 10, 15, $\lambda o \bullet \exists x \in o$, holder(x) = X, $\lambda o \bullet \| o \|$, c)

c is the client who made the request, holder being a function that applies to diaries, $\lambda o \bullet \exists x \in o$, holder(x) = X is a Boolean function that evaluates to true if and only if o (a composition option) contains X's diary. The score associated to each option is its size (e.g the number of participants, to be maximised).

In this expression, the number of service instances entering in the composition is bounded by 10 and 15, therefore relaxing the “all or nothing” rule associated to executions of the composed service: the execution is successful even if the number of participants that complete does not reach 15, as soon as this number is greater than 10.

PI is then used as a component type in the expression defining MO (Meeting Organisation):

$$
\begin{aligned}
&\mathsf{MO} \longleftarrow // (\{\mathsf{PI, Caterer, Room}\}, 3, 3,\\
&\quad \lambda \mathsf{o} \bullet \mathsf{cost(o.Caterer, client(o.PI), (min(o.PI)+max(o.PI))/2)}\\
&\quad\quad + \mathsf{cost(o.Room, client(o.PI), (min(o.PI)+max(o.PI))/2))} \leq \mathsf{b},\\
&\quad \lambda \mathsf{o} \bullet -\mathsf{(cost(o.Caterer, client(o.PI), (min(o.PI)+max(o.PI))/2))}\\
&\quad\quad + \mathsf{cost(o.Room, client(o.PI)), (min(o.PI)+max(o.PI))/2)),}\\
&\quad \mathsf{client(o.PI))}
\end{aligned}
$$

o being an option, o.Caterer (respectively o.Room and o.PI) returns the service, instance of type Caterer (respectively Room and PI), that participates in o.

min(o.PI) (respectively max(o.PI)) returns the lower bound (respectively the upper bound), used as parameters in the expression given to specify PI and which define the range of values for the number of component services which must successfully complete. The function described by the first lambda expression specifies the restriction. The global cost (e.g. calculated by the means of the second lambda expression) specifies the preference, it is calculated for each option o. The opposite of the resulting value (which is a negative number) forms the score associated to o. Both minimum and maximum values are set to 3: to successfully complete, an option must be composed of exactly 3 components, each of which must successfully complete. For instance, if X does not participate, the service Participant Invitation (PI) fails, and MO must then propagate this failure to the two other services by using the cancellation operation associated to the 2PC protocol.

4.3 Transactional Properties of a Service Composition

It is important to note that when min ≥ 2, the minimality constraint implies an atomicity constraint. For example, when min $= 2$, either at least two component services must successfully complete, or none must complete. This means that the situation where only one out of the two components completes can not arise. To enforce this constraint, each option must necessarily contain at least min services which are either atomic or quasi-atomic. The resulting composite service will itself provide the operations associated to the 2PC protocol (preparation, cancellation and validation). The implementation of these operations relies on the operations provided by the component services participating in the composition. The mapping between operations offered by the composite service and those provided by its components is defined by the execution model detailed in the next section.

On the other hand, min $= 1$ does not entail any atomicity constraint. The resulting composite service may be atomic, quasi-atomic or non-atomic depending on the properties of the component services in the selected option. The first situation arises when all participants of the selected option are atomic and provide preparation, validation and cancellation operations. Similarly, when all participants are quasi-atomic, the composite service is then capable of offering a compensation operation. In other cases, the resulting composite service is non-atomic. Note that the option to be selected is not known in advance since it depends on the evaluation of the restriction and scoring functions, and therefore in this situation it is not possible to know at composition time whether the resulting composite service will be atomic, quasi-atomic, or non-atomic. In other words, the exact transactional nature of the composite service can only be known at runtime.

In this paper, we do not consider the possibility of deriving the transactional properties of composite service at runtime (i.e. we restrict the model to design-time derivation). Deriving such properties at runtime (as opposed to design time) will necessarily imply that the set of operations supported by a service (e.g. its interface) can dynamically evolve, since atomic services do not have the same set of operations as quasi-atomic or non-atomic. In contrast, in the context of Web services, the interface of a service is statically defined at design time and exposed as a WSDL document (among other languages) when the service is published. Hence, composite services for which $\mathsf{min} = 1$ are considered to be non-atomic.

5 Execution Model

This section presents the execution model associated to the operator previously introduced. The execution of a composite service is carried out in two steps:

(1) Options exploration: the procedure *selection/reservation* is meant to build so-called *service composition options*, each of which is composed of a set of component services that together satisfy the restrictions specified by the designer. This process involves identifying which services will participate in an option. A service is selected only if a tentative reservation has been successfully obtained (according to the Tentative Hold Protocol). In addition, each option must satisfy the minimality and maximality constraint as well as the restriction function. Each option is then associated with its score (which reflects the designer/user preferences) and the set of options is then sorted in descending order by score. This step is detailed in Section 5.1.

(2) Execution phase: the procedure *execution/validation* considers the options built during the first step and ordered according to their score (from the higher to the lower score). Each option is considered in turn and its execution is attempted (see Section 5.2).

The process summarised above is supported by an architecture detailed in [22].

5.1 Dynamic Selection Process

An important feature of our approach is that the services that will participate in a given execution of a composite service are selected at runtime rather than this choice being hard-coded in the specification of the composite service. Most of the time, several services can be of the same service type, in the sense that they provide the same capability, and are thus interchangeable except for the fact that they may possess different non-functional properties for which the values may vary over time (e.g. cost, location). This feature is similar to the one provided by the concept of "service community" defined in the SELF-SERV service composition system [16, 23].

In our approach we push this idea further: during the execution of a composite service, tentative hold requests are sent to several instances of the same service type. As a result, reservations are obtained from several of these services (which are interchangeable), thereby enabling the execution framework to defer, until validation time, the choice of the instance of a given service type (called a *instance service*) that will participate in the composition.

The selection of services is performed in two steps. First, a thread is created for each of the service types passed as parameters to the operator. Threads are meant to seek potential services supporting the THP protocol and offering the capabilities corresponding to the service type they are associated to.

The second step aims at building the set of all options according to the composition specification. For each of the options built in this way, the restriction function must evaluate to true (i.e. it must satisfy the domain-specific constraints). In addition, each option must contain at least mi component services but not more than ma, where mi and ma are parameters of the composite service. Moreover, if mi's value is two or more, then at least mi services in each option must be atomic or quasi-atomic, otherwise satisfying the atomicity constraint cannot be achieved. This is because non-atomic services do not provide neither a preparation nor a compensation operation which is necessary to ensure atomicity of the composite service execution (i.e. to ensure that at least mi services validate successfully or no service validates successfully). The only operation supported by non-atomic services is validation and the outcome of this operation is not under the control of the composite service: when the validation operation is invoked on a service, the service may either complete its execution successfully (and thus the associated resource is definitely acquired) or it may complete unsuccessfully (meaning that the associated resource could not be acquired).

At the end of the second step, resources associated to services which do not belong to any option and which have been reserved (in the sense of the THP protocol) are released.

Subsequently, the scoring function is evaluated for each composition option. The set of options is then sorted in descending order according to the option's score and the resulting list is passed on to the execution/validation phase described in the next section.

5.2 Execution Process

The second phase aims at executing and validating the services involved in one of the options identified in the previous phase. Options are considered in turn starting with the option with the highest score. For each option, the corresponding transaction is executed as explained below. If the execution of the selected option completes successfully (up to and including the validation phase) then the process stops and the composite service execution is considered to have completed with success. If on the other hand the transaction fails, then the next option in the list is considered. The process continues until either an option has completed successfully, or all the options have been exhausted. In the latter case, the composite service execution is considered to have failed.

All along this phase, the coordination module dynamically updates the list of options. Specifically, when unavailability notifications are received by the coordination module from the component services, meaning that the corresponding resources are no longer available as per the rules of the THP protocol, all options relying on such unavailable services are withdrawn from the list of options. In particular, if a notice of unavailability concerns an option which is in the process of being executed, the corresponding transaction is cancelled (provided that it is still in the "preparation" phase). As the time passes, the score associated to an option may change, thereby modifying its ranking. Studying the evolutions of options over time is out of the scope of this paper, although we acknowledge that studying this issue could lead to identifying various optimization possibilities. The aim here is merely to provide an execution semantics for the composition operator and so, possible optimisations are not considered.

Each option is executed as follows. The transaction manager sends first a *prepare* message to each atomic service in the option (as per the 2PC protocol). If the number of atomic services that positively acknowledges this request (i.e. services that return a message with a "ready" status) is enough to achieve the minimality constraint, then a *validate* message is sent to all the component services (again as per the 2PC protocol), this time including both atomic and non-atomic services. If on the other hand the number of atomic services which acknowledge positively does not reach the required minimum, the validation cannot complete; hence, a *cancel* message is sent to all the atomic services that acknowledged positively thereby releasing the resources reserved during the preparation phase. However, all tentative reservations made during the first phase described Section 5.1 are kept until the end of the execution process, once all the composition options have been tried. Of course, if at a given point during the execution it is found that a given service does not appear in any of the options that have not yet been tried, then the THP on this service may be released.

Quasi-atomic services are treated in a similar way as atomic services, except that during the "preparation" phase of the composite service execution they are sent a *validate* message, as opposed to a *prepare* message. This is because quasi-atomic services do not provide a preparation operation (they only provide two operations: validation and compensation). Following the *validate* request, a quasi-atomic service may reply with either a *success* or a *fail* message. For the purpose of satisfying the minimality constraint, a quasi-atomic service that replies with a *success* message is equated to an atomic service that replies with a *ready* message at the end of the preparation phase. If later on, cancellation of the transaction is necessary because not enough atomic/quasi-atomic services are available to complete the transaction, then a *compensate* message will be sent to all quasi-atomic services that replied with a *success* message during the preparation phase (this is seen as equivalent to sending a *cancel* request to an atomic service). Since quasi-atomic services do not provide a validation operation, they do not get involved in the validation phase of 2PC.

Non-atomic services are considered as extra participants: each non-atomic service that validates during the validation phase is seen as a bonus on top of the atomic/quasi-atomic services, and they can be included in the transaction so long as the number of component services that validate does not exceed the maximum threshold specified for the composite service.

The algorithm that implements the phases described above is detailed in [22].

6 Conclusion

We have presented a composition model for Web services. The model aims at exploiting transactional properties of services which enter in a composition in order to ensure transactional properties on the composite service. The proposed model approach overcomes some key limitations of existing models as identified in Section 2:

Support for heterogeneous transactional properties. The model allows atomic, quasi-atomic, and non-atomic services to be involved in a composition. If an atomicity constraint is specified on the composite service, a certain minimum number of atomic and quasi-atomic services must participate in the composition, but otherwise, atomic, quasi-atomic and non-atomic services can be composed in arbitrary ways.

Provision for limited duration of reservations and cost of compensations. The model aims at (1) maximising the chances of acquiring the resources managed by the selected services during the preparation phase of the 2PC protocol, but without blocking these resources for long periods of time (in the case of atomic services); (2) reducing the risk of having to perform compensations on services because not enough other resources are obtained (in the case of quasi-atomic services); and (3) optimising the chances of acquiring the desired resources (in the case of non-atomic services). To achieve this, the model relies on the Tentative Hold Protocol for placing tentative resource reservations during the execution of the composite service before engaging in the final stage of the transaction.

Support for dynamic selection and substitution. The parallel composition operator which lies at the core of the proposed model does not apply to individual services but rather to service types, that is, sets of substitutable services that provide the same capability. In this way, the operator provides direct support for service selection. Moreover, the operator is parameterised by a restriction and a scoring (i.e. preference) function which guide the selection of composition options.

In order to give a semantics to the composition operator, we have defined an execution model based on an algorithm which explores all composition options that satisfy the minimality and maximaility constraints and the restriction function in order to identify which options maximise the scoring function [22].

In this paper, we focused on atomicity constraints, and more specifically on service coordination under such constraints. In order to be applicable in a broad range of setting, the proposed basic model needs to be extended in at least two directions:

- *Data mediation*: a composite service's data model depends on those of its component services. The composition operator must be extended so as to allow the designer to define the data model of the resulting composite service as well as the dependencies between the data model of the composite service and those of the components. This could be achieved by using, for instance, mediation and reconciliation functions.
- *Control dependencies*: so far we have only considered parallel execution of services (at which point only tentative holds are obtained). Services are then synchronised at validation time, at which point an attempt is made to acquire the underlying resources definitely. In practice however, sequential dependencies may be imposed over the execution of the services that enter in a composition (e.g. service A must complete its execution before service B can start). In order to take into account such dependencies, we plan to extend the model presented here based on the principles of well-known transactional process models such as Sagas [21], which rely on the compensation capabilities of quasi-atomic services.

References

1. Baïna, K., Benatallah, B., Casati, F., Toumani, F.: Model-driven web services development. 16th International Conference on Advanced Information Systems Engineering (CAISE'04), Riga, Latvia. Springer-Verlag **3084** (2004)
2. Roberts, J., Srinivasan, K.: Tentative hold protocol. http://www.w3.org/TR/{tenthold-1,tenthold-2} (2001)
3. Roberts, J., Collier, T., Malu, P., Srinivasan, K.: Tentative hold protocol - part 2. http://www.w3.org/TR/tenthold-2 (2001)
4. Elmagarmid, A.K., ed.: Database Transactions Models for Advanced Applications. Morgan Kaufmann Publishers (1990)
5. Gray, J., Reuter, A.: Transaction Processing: concepts and Tecniques. Morgan Kaufmann Publishers (1993)
6. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications. Springer Verlag (2003)
7. Papazoglou, M.: Web services and business transactions. Technical Report 6, Infolab, Tilburg University, Netherlands (2003)
8. Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., Langworthy, D., Orchard, D.: Web service coordination, ws-coordination. http://www.ibm.com/developerworks/library/ws-coor/ (2002) IBM, Microsoft, BEA.
9. Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T., Thatte, S.: Web service transaction, ws-transaction. http://www.ibm.com/developerworks/library/ws-transpec/ (2002) IBM, Microsoft, BEA.

10. Si, Y., Edmond, D., H. M. ter Hofstede, A., M., D.: Property propagation rules for prioritizing and synchronizing trading activities. In: CEC, IEEE International Conference on Electronic Commerce (CEC 2003), Newport Beach, CA, USA, IEEE Computer Society (2003) 246–255
11. Arregui, D., Pacull, F., Riviere, M.: Heterogeneous component coordination: The clf approach. In: EDOC, 4th International Enterprise Distributed Object Computing Conference (EDOC 2000), Makuhari, Japan, IEEE Computer Society (2000) 194–2003
12. Hagen, C., Alonso, G.: Exception handling in workflow management systems. Software Engineering, IEEE Transactions on **26** (2000) 943–958
13. Limthanmaphon, B., Zhang, Y.: Web service composition transaction management. In Schewe, K.D., Williams, H., eds.: ADC. Volume 27 of CRPIT., Database Technologies 2004, Proceedings of the Fifteenth Australian Database Conference, ADC 2004, Dunedin, New Zealand, Australian Computer Society (2004)
14. Vidyasankar, K., Vossen, G.: A multi-level model for web service composition. Technical report, Dept. of Information Systems, University of Muenster (2003) Tech. Report No. 100.
15. Halliday, J., Shrivastava, S., Wheater, S.: Flexible workflow management in the OPENflow system. In: EDOC, 5th IEEE/OMG International Enterprise Distributed Object Computing Conference, IEEE Computer Society Press, USA (2001) 82–92
16. Benatallah, B., Dumas, M., Maamar, Z.: Definition and execution of composite web services the self-serv projet. IEEE Computer Society Technical Committee on Data Engineering Bulletin **25** (2002)
17. Mikalsen, T., Tai, S., Rouvellou, I.: Transactional attitudes: Reliable composition of autonomous web services. In Workshop on Dependable Middleware-based systems (WDMS 2002) (2002) Washington, D.C., USA.
18. Dalal, S., Temel, S., Little, M., Potts, M., Webber, J.: Coordinating business transactions on the web. IEEE Internet Computer Society Journal **7** (2003) 30–39
19. for the Advancement of Structured Information Standards OASIS, O.: Business transaction protocol - btp. http://www.oasis-open.org/ (2002)
20. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I.: Business process executions langage for web services. version 1.1. http://www.ibm.com/developerworks/library/ws-bpel (2003) BEA, IBM, Microsoft, SAP AG, Siebel Systems.
21. Garcìa-Molina, H., Salem, K.: Sagas. In: Proc. of the ACM SIGMOD International Conference on Management of Data, San Francisco, CA, USA (1987) 249–259
22. Fauvet, M.C., Duarte, H., Dumas, M., Benatallah, B.: Handling transactional properties in web service composition. Technical report, CLIPS-IMAG, Joseph Fourier University of Grenoble (2005) http://www-clips.imag.fr/http://www-clips.imag.fr/mrim/User/marie-christine.fauvet/Publis/wise05_full.pdf.
23. Benatallah, B., Sheng, Q.Z., Dumas, M.: The SELF-SERV Environment for Web Services Composition. IEEE Internet Computing (2003) IEEE Computer Society.