Overview of Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services

Sami Bhiri, Khaled Gaaloul, Olivier Perrin, Claude Godart

Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services

Sami Bhiri, Olivier Perrin, Claude Godart, and Khaled Gaaloul

LORIA - INRIA - CNRS - UMR 7503
BP 239, F-54506 Vandœuvre-ls-Nancy Cedex, France
{bhiri, operrin, godart, kgaaloul}@loria.fr

Abstract. In this paper, we present an approach to easily define flexible and reliable services compositions. We introduce a new concept called *transactional patterns* to specify flexible and reliable composite Web services. A *transactional pattern* is a convergence concept between workflow patterns and advanced transactional models. It can be seen as a coordination pattern and as a structured transaction. Thus, it combines workflow flexibility and transactional processing reliability. Designers can simply connect together a set of *transactional patterns* to define a composite Web service. We use a set of techniques to ensure control and transactional coherence between patterns inside a services composition.
Keywords: Web services compositions, Workflow patterns, transactional processing.

1 Introduction

Web services are a great technology for dealing with B2B business processes, such as e-procurement for instance, but handling failures using the traditional transactional model for long running, asynchronous, and decentralized activities has been proven to be unsuitable. Advanced Transaction Models (ATMs) [1] have been proposed to manage failures, but, although powerful, ATMs are too database-centric, providing a nice theoretical framework but limiting their possibilities and scope [2] (e.g. their inflexibility to incorporate different transactional semantics as well as different behavioral patterns into the same structured transaction [3]).

In the same time, workflow [4] has became gradually a key technology for business process automation [5], providing a great support for organizational aspects, user interface, monitoring, accounting, simulation, distribution, and heterogeneity [2].

In this paper, we propose an approach to specify flexible and reliable Web services compositions based on the concept of *transactional patterns*.A transactional pattern combines workflow flexibility and transactional reliability, and we are using them to specify and orchestrate composite Web services. Then, we use a set of techniques to ensure the control and the transactional consistency for a composition.

Section 2 presents a motivating example. Section 3 and 4 present transactional composite Web services and workflow patterns, while section 5 introduce the concept of transactional patterns. In section, we show how to use them to specify the composite Web service, guaranteeing the consistency. Section 6 concludes.

2 Motivating example

We consider an application for online travel arrangement, carried out by a composite service illustrated in figure 1. The customer specifies its requirements for destination and hotels. The composite service launches in parallel the hotel and the flight bookings. Then, the customer is requested to pay online. Once this is done, the travel documents are sent to the customer. To avoid failures, the designers of the composite service may want to augment the control flow described above with a set of transactional requirements. For instance, they may require the services FB and TDU to be sure to complete, but also the service FB to be compensatable (when an hotel booking is cancelled, or when it fails). They may also specify that service TDU is an alternative for $TDFE$.

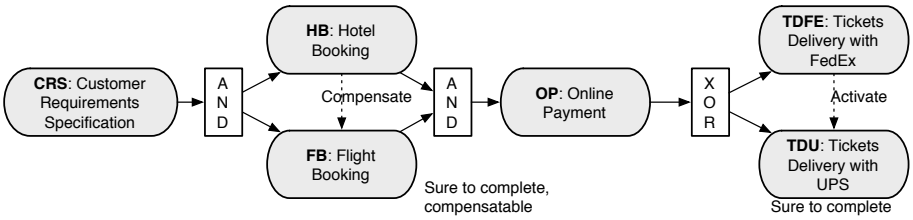

Fig. 1. A composite service for online travel arrangement.

Modeling this example with ATM or workflow systems is not easy because ATM are too rigid to enable a such control structure, and ATM do not support bottom-up applications design, starting from predefined business process and using pre-existing systems or services with diverse semantics [3]. On the other hand, workflow systems lack functionalities to assess that the specified transactional behavior ensure the required reliability. In our example, the service OP can eventually fail, causing the travel arrangement abortion, while the flight and the hotel bookings are always maintained.

3 Transactional composite Web services

3.1 Transactional Web service

A Web service is a self-contained modular program built with XML, SOAP, WSDL and UDDI specifications that can be discovered and invoked across the Internet ([6, 7]). A *transactional Web service* is a Web service that emphasizes transactional properties for its characterization and correct usage.

The main transactional properties of a Web service we are considering are *retriable, compensatable, pivot* [8] and *with effects*. A service s is said to be ***retriable*** (s^r) if it is sure to complete after a finite number of activations. s is said to be ***compensatable*** (s^{cp}) if it offers compensation policies to semantically undo its effects. Then, s is said to be ***pivot*** (s^p) if once it successfully completes, its effects remains for ever and cannot be semantically undone. Naturally, a service can combine properties, and the set of all possible combinations is $\{\emptyset; r; cp; p; (r, cp); (r, p)\}$.

Given the transactional properties of a service, a set of operations is available. For instance, a pivot service has a minimal set *abort(), activate(), cancel(), fail(), terminate()* allowing respectively its abortion before activation, its activation, its cancellation during its execution, its failure and its successful termination. A compensatable service has in addition a *compensate()* operation for its compensation. A retriable service has a *retry()* operation allowing to activate it after each failure.

3.2 Transactional composite Web service

A composite Web service is a conglomeration of existing Web services working in tandem to offer a new value-added service [5]. It coordinates a set of services as a cohesive unit of work to achieve common goals. A *Transactional Composite (Web) Service* (TCS) emphasizes transactional properties for composition and synchronization of component Web services. It takes advantage of services transactional properties to specify mechanisms for failure handling and recovery. A TCS defines orchestration between its services using dependencies to specify how services are coupled and how the behavior of given services influences the behavior of some others. These dependencies are used to express the relationships (sequence, alternative, compensation,. . .) between component services. In our proposition, we consider the following dependencies: activation, alternative, abortion, compensation, cancellation. More details on dependencies can be found in [9, 10].

3.3 Control flow and transactional flow of a TCS

Using dependencies, we separate the TCS control flow and the TCS transactional flow. The **control flow** of a TCS specifies the partial ordering of component services activations. Thus, only activation dependencies between component services define the corresponding TCS control flow. In short, we define the *control flow* of a composite service as the set of dependencies in which dependencies are only activation dependencies.

The **transactional flow** of a TCS specifies interactions for failures handling and recovery. Transactional dependencies (compensation, cancellation and alternative) define the transactional flow. In short, we define the *transactional flow* of a composite service as the set of dependencies where dependencies are only transactional dependencies.

Of course, transactional dependencies depend on activation dependencies semantics. Thus, a *transactional flow* is always defined according to a given *control flow*.

4 Workflow patterns

As defined in [11], a pattern "is the abstraction from a concrete form which keeps recurring in specific non arbitrary contexts". A workflow pattern ([12]) can be seen as an abstract description of a recurrent class of interactions based on (primitive) activation dependency. For example, the *AND-join* pattern (see figure 2.b) describes an abstract services orchestration by specifying services interactions as follows: *a service is activated after the completion of several other services*. Thus, a pattern explicitly defines activation dependencies (i.e. the control flow) a given set of services.

In this paper, we put emphasis on the following three patterns: *AND-split*, *AND-join* and *XOR-split*[1]. Figure 1 illustrates the patterns *AND-split* applied to *(CRS, HB, FB)*, *AND-join* applied to *(HB, FB, OP)*, and *XOR-split* applied to *(OP, TDFE, TDU)*.

We argue that once defined, a workflow pattern implicitly defines a new class of dependencies, called potential transactional dependencies, i.e. a set of dependencies not initially defined by the pattern that can be used/added in order to tailor (or modify) the control flow (see figure 2). In fact, these dependencies are directly related to the semantics of the activation dependencies of the pattern.

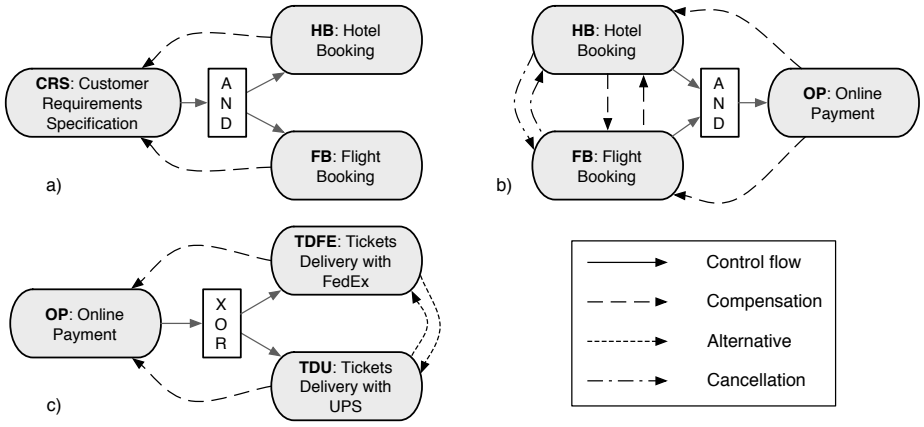

Fig. 2. AND-split, AND-join, and XOR-split patterns and their corresponding potential dependencies.

5 Web services composition using transactional patterns

5.1 Transactional patterns

Given the transactional properties of a service, and a workflow pattern for a composite service, we are able to deduce a new pattern, called a *transactional pattern*, that will be used to specify both the control and transactional flows. The control flow is inherited from the workflow pattern (i.e. the activation dependencies), while the transactional flow is specified using a set of transactional dependencies for managing alternatives, compensation, or cancellation.

From a transactional pattern, one can define several *transactional pattern instances* which are the application of a transactional pattern to a given set of services, and where the transactional dependencies of the instance is a subset of the set of transactional dependencies of the transactional pattern. For instance, on figure 2.c, a designer may choose to keep the alternative dependency for the delivery, while another may choose the compensation dependency. The choice depends not only on the designer, but also on the transactional properties of the compenent services.

[1] Our approach also considers the following list of patterns: sequence, *AND-split*, *OR-split*, *XOR-split*, *AND-join*, *OR-join*, *XOR-join* and *m-out-of-n*.

5.2 Composition

Transactional patterns are an interesting way to compose a set of Web services to obtain a transactional composite service (TCS) that is reliable on an execution point of view. We specify a TCS as a set of transactional patterns instances connected together (sharing some component services). Figure 3 shows how we can specify a TCS for the online travel arrangement as the following transactional patterns composition: *Trans*$_{AND-split}$*(CRS, HB, FB)*, *Trans*$_{AND-join}$*(HB, FB, OP)*, *Trans*$_{XOR-split}$*(OP, TDFE, TDU)*.

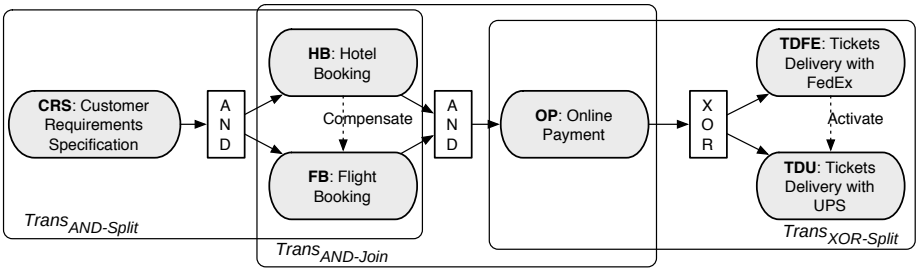

Fig. 3. A TCS is defined as a composition of a set of transactional patterns instances.

However, connecting a set of transactional patterns instances can lead to a control flow and/or a transactional flow inconsistencies. For instance, control consistency problem can raise when instances are disjoined (without shared services allowing to connect the instances) or when an *XOR-split* instance is followed by an *AND-join* instance. Likewise, transactional inconsistency can raise when a component service fails, causing the entire TCS abortion, with remaining effects of the partial execution. For example, if we suppose that OP is not retriable (it can fail) in the TCS defined in figure 3, this means that FB should be compensated in order to be sure that it does not exist a remaining effect (a flight is booked) after the abortion of the TCS. This implies that it exists the compensate transactional dependency between OP and FB, and that FB is compensatable.

To manage these problems, we define a composition as *valid* if it ensures both the control flow consistency and the transactional flow consistency. In order to guarantee the reliability of the TCS, we are using a set of rules to check both the control flow and the transactional flow consistency. Some of these rules are described in [10]. Briefly summarized, the algorithm we are using is as follows:

1. after a component service failure, we are looking for an alternative dependency, if it exists,
2. after a composite service failure, we try to compensate what can be compensated given the transactional properties of the component services,
3. then, after a composite service failure, we cancel all the current executions of the TCS.

In order to compute the transactional consistency, we need not only these rules, but also the transactional properties of each service introduced in 3.1. Thus, there are two

ways to use our approach. The first one is to fix the transactional dependencies for the TCS, and to compute what are the possible transactional properties of the component services. The second one is to choose a set of services and the transactional patterns, and to detect the transactional inconsistencies.

6 Conclusion

In this paper, we propose a solution to ensure reliable and flexible Web service compositions. The main idea of our approach is combining workflow flexibility and transactional processing reliability. We introduce an extension of workflow patterns, the *transactional patterns*, which can be seen as a convergence concept between workflow systems and transactional models to easily define flexible and reliable composite Web services. Then, we propose a set of rules in order to avoid inconsistencies which can result from the composition of the patterns.

References

1. A. Elmagarmid. *Transaction Models for Advanced Database Applications*. Morgan-Kaufmann, 1992.
2. G. Alonso, D. Agrawal, and A. El Abbadi. Process Synchronisation in Workflow Management Systems. In *8th IEEE Symposium on Parallel and Distributed Processing (SPDS'97)*, New Orleans, Louisiana, October 1996.
3. Nektarios Gioldasis and Stavros Christodoulakis. Utml: Unified transaction modeling language. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering*, pages 115–126. IEEE Computer Society, 2002.
4. W. M. P. van der Aalst and K. M. van Hee. *Workflow Management: models, methods and tools*. Cooperative Information Systems. MIT Press, 2002.
5. B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. K. Elmagarmid. Business-to-business interactions: issues and enabling technologies. *The VLDB Journal*, 12(1):59–85, 2003.
6. Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing*, 6(2):86–93, 2002.
7. Paulo F. Pires, Mario R. F. Benevides, and Marta Mattoso. Building reliable web services compositions. In *Web, Web-Services, and Database Systems*, pages 59–72, 2002.
8. Sharad Mehrotra, Rajeev Rastogi, Henry F. Korth, and Abraham Silberschatz. A transaction model for multidatabase systems. In *ICDCS*, pages 56–63, 1992.
9. Sami Bhiri, Claude Godart, and Olivier Perrin. Reliable web services composition using a transactional approach. In *to appear in IEEE International Conference on e-Technology, e-Commerce and e-Service, Hong Kong*, March-April 2005.
10. Sami Bhiri, Olivier Perrin, and Claude Godart. Ensuring required failure atomicity of composite web services. In *to appear in the 14th International World Wide Web Conference, Japan*, May 2005.
11. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.
12. W. M. P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Workflow Modeling using Proclets. In O. Etzion and Peter Scheuermann, editors, *5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00)*, number 1901 in LNCS, pages 198–209, Eilat, Israel, September 6-8, 2000. Springer-Verlag.