Combining dynamic Workflow and transactional semantics using a pattern-based approach

Imed Abbassi [*], Mohamed Graiet [†], Nejib Ben Hadj-Alouane [*] and Walid Gaaloul[‡]
[*]*ENIT, University of Tunis El Manar*
UR-OASIS, Tunisia
abbassi_imed@ymail.com, nejib_bha@yahoo.com
[†]*ISIMM, Monastir University*
MIRACL, Tunisia
mohamed.graiet@imag.fr
[‡]*Computer Science Department Telecom SudParis*
Paris, France
walid.gaaloul@it-sudparis.eu

***Abstract*—In this paper, we propose a new paradigm, called dynamic transactional pattern, for specifying flexible and reliable composite Web services in a pervasive environment. This new paradigm is a convergence concept of dynamic Workflow patterns and advanced transactional model. It can be seen both as a dynamic coordination and a structural transaction. Indeed, it combines dynamic Control-Flow flexibility and transactional processing reliability.**

***Keywords*-Web service; dynamic Web service composition; Dynamic transactional pattern; Dynamic Control-Flow pattern.**

I. Introduction

Web services are modular and self-describing software applications that is advertised, located, and invoked across the Internet using a set of standards such as SOAP, WSDL, and UDDI [1]. It is based on sets of XML standards allowing it to be more portable than previous technologies. One of the most interesting features that this technology offers is the possibility of creating dynamically new value-added Web services by composing those that better meet the customer's requirements [2].

Ensuring reliable (from execution point of view) and flexible Web service composition in a pervasive environment remains one of the most difficult tasks. Various existing studies are proposed trying to resolve this problem. In [3], authors present an approach specifying relaxed atomicity requirements for composite services based on Acceptable Termination States (ATS) model and transactional rules to validate a given composite WS with respect to defined Transactional Requirements. In [4], authors propose a selection mechanism enabling the automatic design of transactional composite service by using the ATS model. However, the main drawback of these approaches is the definition of all the ATS by the user, which is neither simple nor scalable. In this paper, we propose an approach to easily define flexible and reliable Services compositions. To achieve this goal, we introduce a new concept called dynamic transactional pattern. A dynamic transactional pattern is a convergence concept of dynamic Workflow [5] and advanced transactional models [6]. It can be seen both as a dynamic coordination and a structured transaction. Contrary to [3], [4], our approach responds to unexpected events that may occur in a pervasive environment such as the appearance, disappearance or the modification of a component Service. Our approach combines the flexibility of dynamic Workflow models and the reliability of Advanced Transactional Models.

The remainder of the paper is organized as follows. We first motivate our work through an example which we use throughout the paper. Then, we present our Dynamic Web service model which allows for capturing both the control and the transactional flow of composite services. Next, we propose a pattern based approach for modeling dynamic transactional composite services. Then we illustrate how we proceed to ensure reliability. Finally we discuss some related work, draw some conclusions and provide some perspectives related to our future research.

II. Motivating Example

Throughout this article, we will illustrate our ideas using a running example of dynamic Web services composition. Let us consider a Touristic Circuit Reservation (TCR) application (see Figure 1).

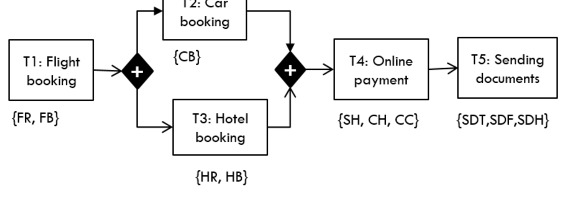

Figure 1. The skeleton of the touristic circuit reservation application

TCR scenario involves multiple tasks often performed by different processing entities (Web services), namely flight booking (T1), car booking (T2), and hotel booking (T3), online payment (T4) and finally sending documents (T5). First, the customer performs the flight-booking task. As shown in Figure 1, several Web services can be selected to execute a task. For instance, both Flight Booking (FB) and Flight Reservation (FR) Web services can provide the flight-booking task. Next, the customer may only perform hotel reservation, as it is not required to book both Hotel and Car for a trip, so the Car booking Service should be optional. Only the Car Booking (CB) Web service provides the car reservation task. For the hotel reservation, either Hotel Reservation (HR), or Hotel Booking (HB) Services can be selected. Whenever, the hotel and the flight booking tasks are performed successfully, the customer is requested to pay. One of the following Web services: Credit Card (CC), Check (CH) and caSH (SH) carries out the payment procedure. Finally, the reservation documents are sent to customers using one of the following Web services: Send Document by Fedex (SDF), Send Document by DHL (SDH) and Send Document by TNT (SDT).

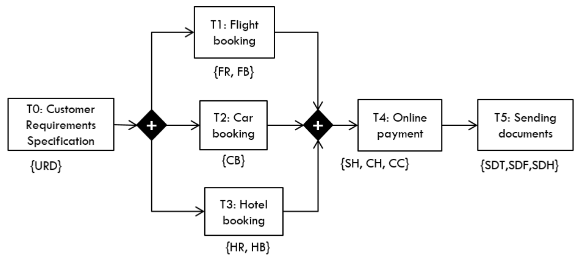

Figure 2. The TCR application skeleton after modification

During the execution of the TCR application, several faults may occur. For instance, if the service performing the hotel-booking task no longer exist or is temporarily out-of-service, the application can select another service, which has exactly the same functionality and satisfying the same requirements. The TCR application can be updated as follows. An activity, called Customer Requirements specification (CRS), is defined (see Figure 2). CRS allows customers to specify their preferences (e.g. cost, execution time, availability) using User Requirements Definition (URD) Web service. Modeling this example using either Advanced Transactional Models, or dynamic Workflow patterns is not easy. Indeed, Advanced Transactional Models is too rigid to enable a control structure, and they do not support bottom-up applications design, starting from predefined business process and using pre-existing systems or Services with diverse semantics, In addition, dynamic Workflow systems require sound mechanisms for reliability and correctness in case of failure. For example, if the task T2 (car booking) or T4 (payment) fails for any reason or became unavailable, then how to do to automatically recover the global transaction?

III. Dynamic Composite Service Model

In this section, we propose a formal model for dynamic Web services composition. Our model integrates the expressiveness and the power of dynamic Workflow models and the reliability of Advanced Transactional Models. The originality of this model is the flexibility offered not only to designers, to specify their requirements in terms of control structure, but also to the customers to specify their needs (for instance, QoS, need for new activities, temporal constraints).

A. Transactional Service

Transactional Web service is a Service that emphasizes transactional properties for its characterization and usage [3]. Therefore, a Web service can be considered as a simple transaction, which can be compensatable (c), retriable (r), or pivot (p) [7]. A Web service, ts, is said to be retriable (r) if it is sure to complete after a finite number of activations. ts is said to be compensatable (c) if it offers compensation policies to semantically undo its effects; whereas it is said to be pivot(p) if once it successfully completes, its effects remain and cannot be semantically undone. In our approach, **SERVICE** denotes the set of the transactional Web services. We use a function, TP ($TP \in SERVICE \to \mathbb{P}(\{c, p, r\})$), to define the transactional behavior of Web services. A Web service, s ($s \in SERVICE$), can combine several transactional properties, which leads to a new behavioral property. For instance, a service can combine the pivot and retriable properties. Similarly, a service can be compensatable and retriable which leads to a new behavioral. However, it cannot be compensatable and pivot. That can be illustrated by the following predicate: $\forall s \cdot s \in SERVICE \Rightarrow TP(s) \in \{\{c\}, \{p\}, \{r\}, \{c, r\}, \{r, p\}\}$.

B. Dynamic Web Services Composition

We define a dynamic transactional composite services (DTCS) as a conglomeration of existing transactional Web services working in tandem to offer a new value-added service [8]. It is composed at run time, i.e., the Web services that participate in composition process are selected and related at run time. This kind of composition is called also objective based composition. It can be automatically reconfigured in case of unexpected situations.

The interaction between components Service expresses several kinds of relation in form of dependencies such as sequence, compensation, abortion and cancellation. These dependencies should respect the following constraints:

R1: an abortion dependency from s1 to s2 can exist only if there is an activation dependency from s1 to s2;

R2: a compensation dependency from s1 to s2 exists if (1) s1 may fail, (2) s2 is compensatable, and (3) either it exists an activation dependency from s2 to s1, or s1 and s2 execute in parallel and are synchronized;

R3: a cancellation dependency from s1 to s2 exists only if s1 may fail and s1 and s2 execute in parallel and are synchronized.

The above rules ensure consistent transactional dependencies between Web service. However, they do not enough to guarantee a reliable execution. For this reason, the following rules should be considered:

R4: the successor of a pivot service has to be retriable;
R5: the Web services executing in parallel with a pivot and retriable service should be retriable;
R6: the Web services executing in parallel with a pivot and non-retriable service should be compensable.

Formally, we define the dependency concept as the following relation:$\Re \in SERVICE \leftrightarrow SERVICE$. s1 $\mapsto$ s2 ($s1 \mapsto s2 \in \Re$) means that it exists a dependency from s1 to s2. The activation dependencies, depAct ($depAct \subset \Re$), specify a partial ordering of component services executions. A compensation dependency, depCps ($depCps \subset \Re$), allows expressing a backward recovery mechanism, while a cancellation dependency, depCnl ($depCnl \subset \Re$), allows signaling a service execution failure to other service(s) being carried out in parallel by canceling their execution if necessary. An abortion dependency, depAbt ($depAbt \subset \Re$), allows propagating failures from one service to its successor(s) by aborting them.

The control flow defines a partial ordering of component services activations. Intuitively, it is defined by the set of its activation dependencies and extracted dynamically from tasks dependencies. The transactional flow defines recovery mechanisms in order to ensure that, in case of failure, the partial work already done is compensated and that ensuring the control and transactional flow consistency.

IV. Dynamic Transactional Patterns

In this section, we introduce the concept of dynamic transactional pattern, a new paradigm, which we propose to specify reliable dynamic Web services compositions. Dynamic transactional patterns extend dynamic Workflow patterns [5] with transactional dependencies, thus allowing bridging their transactional lack.

A. Dynamic Workflow patterns

The dynamic Workflow patterns [5] are mainly used for handling situation where decision is made during runtime. They allow decision to be made just before the instantiation of activities. Unlike [9], [10] that mainly focus on Workflow patterns with a static reconfiguration, the dynamic Workflow patterns can be automatically reconfigured at runtime if a subprocess is no longer available or is added. Due to the lack of space, we put emphasis on the following three patterns: DAD-split (Dynamic AnD-split), DAD-join (Dynamic AnD-join) and DSQ (Dynamic SeQuence) to explain and illustrate our approach. However, the concepts presented here can be applied similarly also to others basic pattern like DOR-split (Dynamic OR-split), DOR-join (Dynamic OR-join) and DNM-join (Dynamic N out Of M Join). In the following, we describe the dynamic transactional patterns and also their formal semantic.

A DAD-split pattern (see Figure 3) is a dynamic fork pattern. It is specified as a point in a dynamic process where a single process of control splits into multiple process s of control, which can be executed in parallel, thus allowing Services to be executed simultaneously or in any order. Formally, we define the DAD-split pattern as the following function: DAD-split(S): S=$\{in, s_1, \ldots, s_n\} \mapsto \{in\} \times \{s_1, \ldots, s_n\}$.
As shown in Figure 3, the resulting control flow of the application of the DAD-split pattern can be redefined automatically when a single subprocess is either added (see Figure 3.a), or removed (see Figure 3.c) from the business process. In the later case (see Figure 3.c), a DAD-split can be replaced by a DSQ pattern. Moreover, another case may occur in which a subprocess can be replaced by an equivalent one (see Figure 3.b).

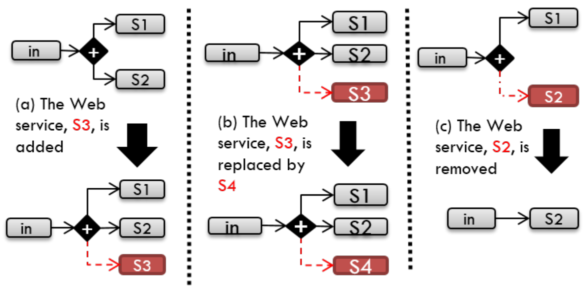
in
S1
S2
(a) The Web service, S3, is added
in
S1
S2
S3
in
S1
S2
S3
(b) The Web service, S3, is replaced by S4
in
S1
S2
S4
in
S1
S2
(c) The Web service, S2, is removed
in
S2

Figure 3. Dynamic reconfiguration of the DAD split pattern

A DAD-join pattern (see Figure 4) is a dynamic join pattern. It is specified as a point in a dynamic process where multiple parallel subprocess converge into one single process of control, thus synchronizing multiple process. Formally, we define the DAD-join pattern is as follow function:DAD-join(S): S=$\{s_1, \ldots, s_n, out\} \mapsto \{s_1, \ldots, s_n\} \times \{out\}$
As shown in Figure 4, the resulting control flow of the application of the DAD-join pattern can be redefined automatically when a single sub-process is either added (see Figure 4.a), or removed (see Figure 4.c) from the business process. In the later case, a DAD-join can be replaced by a DSQ pattern. Moreover, the Figure 4.b shows that subprocess can be substituted by another one.

A DSQ (see Figure 5) is a dynamic sequence pattern. It is specified as a point in a dynamic process. It specifies that a single subprocess may be activated after the termination of another one. It provides a dynamic control flow since a single subprocess can be added (see figures 5.a, 5.b) to the business process, or another can replace a single subprocess (see Figure 5.c). Formally, we define the DSQ pattern as

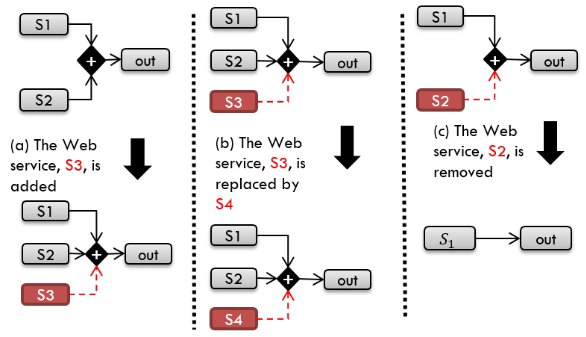
S1
S2
out
(a) The Web service, S3, is added
S3
(b) The Web service, S3, is replaced by S4
S4
(c) The Web service, S2, is removed
S_1

Figure 4. Dynamic reconfiguration of DAD join pattern

follow: DSQ({in, out}) = {in ↦ out}.

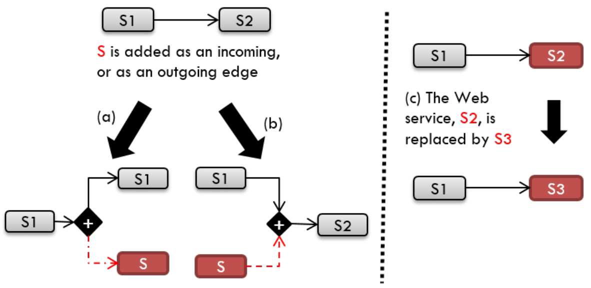
S1
S2
S is added as an incoming, or as an outgoing edge
(a)
(b)
S
(c) The Web service, S2, is replaced by S3
S3

Figure 5. Dynamic reconfiguration of DSQ pattern

B. Extending Dynamic Workflow Patterns with Transactional Dependencies

In this section, we extend the dynamic Control-Flow pattern that we have proposed with transactional dependencies.

Given the transactional properties of Web services and the dynamic Control-Flow patterns that we have proposed for a composite Web service, we are able to deduce a new concept, called dynamic transactional patterns that will be used to specify both the control and transactional flows. A dynamic transactional pattern extends the dynamic Control-Flow pattern by transactional dependencies. The control flow (skeleton) is defined from the dynamic Control-Flow pattern (i.e., the activation dependencies), while the transactional flow is specified using a set of transactional dependencies such as abortion, cancellation and compensation dependencies. In this work, we introduce the following Dynamic transactional patterns DTAD-split (Dynamic Transactional AnD split), DTAD-join (Dynamic Transactional AnD join) and DTSQ (Dynamic Transactional SeQuence). These patterns are formally defined as function tacking as an input parameter a set of Web service S ($S \in \mathbb{P}(SERVICE)$) and returning a set F ($F \in \{act, cps, cnl, abt\} \nrightarrow \mathbb{P}(SERVICE \times SERVICE)$). F is a partial function that defines the activation (F(act)), compensation (F(cps)), abortion (F(abt)) and cancellation (F(cnl)) dependencies. According to our formalization, an instance is the application of dynamic transactional pattern to a set Web service S (*e.g.,* DTAD-split(S) is an example of instance).

The DTAD-split pattern extends DAD-split with a set of transactional dependencies for handling failure and recovery mechanism. It supports abortion, cancellation and a compensation dependencies. A compensation (or a cancellation) dependency from s1 to s2 can be defined only if s1 and s2 are synchronized. The DTAD-join pattern extends DAD-join with a set of any kind of transactional dependencies (*e.g.,* abortion, cancellation, or compensation). The DTSQ pattern extends DSQ with a set of abortion and compensation dependencies.

V. Services Composition using Dynamic Workflow Patterns

In this section, we describe how we use the concept of dynamic transactional pattern to specify reliable and flexible composite service. In particular, we show how a DTCS is automatically reconfigured.

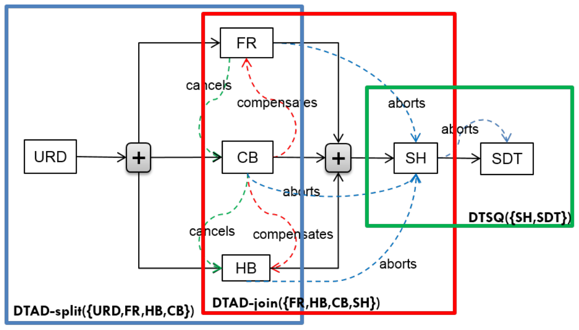

Figure 6. An example of a composite service for the touristic circuit reservation

A. Composition

We define an instance of a dynamic transactional pattern "P" the resulting control and transactional flows of the application "P" to a given set of Web services. In our approach, a DTCS is specified as the union of a set of instances.

Back to our motivating example, Figure 6 illustrates how we can specify a composite services for the touristic circuit reservation by connecting three pattern's instances : DTAD-split({URD,FR,CB,HB}); DTAD-join({FR,CB,HB,SH}) and DTSQ({SH,SDT}).

Unfortunately, connecting a set of dynamic transactional patterns instances can lead to a control flow and/or a transactional flow inconsistency. For instance, control consistency problem can rise when instances are disjoined without shared Services allowing connecting them. Similarly, transactional inconsistency can rise when a component Service fails causing the entire DTCS abortion, without compensating the

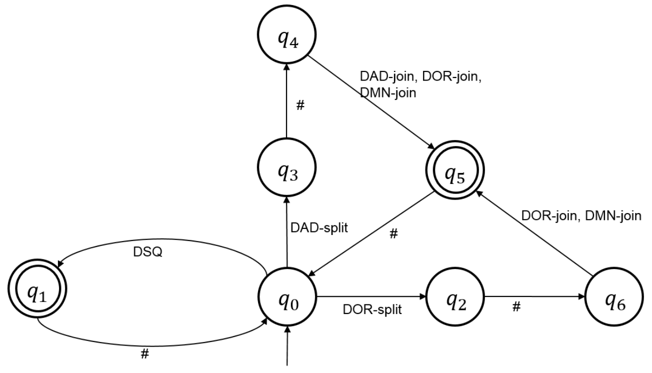

Figure 7. A deterministic finite-state automaton defining the language L(A) of the consistent control flows for Web services composition

partial work already done. A DTCS is reliable when both the control and transactional flows are consistent.

B. Ensuring Control and Transactional flows Consistency

The first step to ensure the Web service composition reliability is in ensuring a consistent skeleton (control flow). In order to help designers to specify a consistent skeleton, we propose a deterministic finite-state automaton A = ($\sum$,Q,q_0, $\{q_1, q_5\}$, σ), where:

- $\sum$ ={DAD-split, DAD-join, DSQ, DOR-split, DOR-join, DMN-join, #}
- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$
- q_0 is the initial state.
- $\{q_1, q_5\}$ is a set of final states.
- $\sigma \in (Q \times \sum) \nrightarrow Q$ is a partial transition function.

The graphical presentation of A is described in Figure 7. This automaton defines a language L(A) of the consistent control flows. A control flow CF is consistent if and only if it is a word of L(A) ($CF \in L(A)$) and respecting a set of constraints C={C1,C2,C3,C4}. The first constraint (C1) postulates that CF must start either with a DSQ, a DOR-split, or a DAD-split, The second one (C2) expresses that a DSQ can be followed by one of the following symbols: DSQ, DOR-split, or DAD-split. (C3) assumes that a DAD-split can be followed by DAD-join, DOR-join, or DMN-join. The fourth constraint (C4) a DOR-split symbol can be followed only by, a DOR-join or a DNM-join pattern. From the constraints C3 and C4, we can deduce that a word ω belonging to L(A) should terminate either by DSQ, DAD-join, DOR-join, or DMN-join. The word ω_{TCR} = DAD-split#DAD-join#DSQ corresponding to the skeleton of the touristic circuit reservation is consistent belongs to L(A). A component service in a given DTCS can be itself a dynamic composite service where its control flow is consistent, thus allowing to use dynamic transactional patterns in a nested way inside a dynamic composition.

The next step in defining reliable composition is to specify a consistent transactional flow. A consistent control flow and the transactional properties of the Web services are required to generate it. A transactional flow is said to be consistent if it respects the following properties: (P1) it compensates the work already done in case of a fatal error (substitution process is failed, etc.), and (P2) it cancels all running executions in parallel. We propose the following three rules, Rule-1, Rule-2 and Rule-3, to generate the transactional flow :

- Rule-1: If it exist an activation dependency from s1 to s2, then define an abortion dependency from s1 to s2;
- Rule-2: If (1) s2 is compensatable, (2) s1 is not retriable and (3) there is an activation dependency from s1 to s2, or s1 and s2 run in parallel and are synchronized, then define a compensation dependency from s1 to s2;
- Rule-3: If s1 is not retriable, s1 and s2 run are synchronized, then define a cancellation dependency from s1 to s2.

The composite service presented in Figure 6 is unreliable since it doesn't respect the properties P1 and P2. Indeed, the Web services HB and CB are unreliable because they are not retriable. Therefore, if HB fails for any reasons, the TCR service is aborted with car and flight are booked. This means compensation/cancellation dependencies from HB to CB and from HB to FR are required in order to guarantee that it doesn't exist remaining effects in case of fatal errors causing the abortion of the global composite service.

VI. Related Work

Workflow [11] has become a key technology for business process automation, providing a great support for organizational aspects (business-to-business interaction), user interface, monitoring accounting, distribution and heterogeneity. In [9], the authors propose a collection of control-flow patterns that can be categorized into six categories: basic control flow patterns, advanced branching and synchronization patterns, structural patterns, multiple instance patterns, state-based patterns and cancellation patterns. Unlike [9], [10] that mainly focused on static Workflow patterns, we will focus in this work on dynamic Workflow patterns as well as dynamic fork, dynamic join and dynamic sequence [5]. Other closely related works include [12], [13] that deal with Workflow data patterns and Service interaction patterns.

Several existing standards such as BTP [14] and WS-transaction [15] support a two-phase Web services coordination. They are based on a set of extended models to specify Web services interaction. Firstly, the participants (Web services) should agree on a specific model before starting interactions. Then, the corresponding coordination layer technologies support the appropriate messages exchanges according to chosen transactional model. These proposals inherit the rigidity of the extended transactional models. In [3], the authors specified flexible and reliable composite Services. It is true that this solution is reliable. However, it is not clear how it aggregates new transactional properties and responds to foreseen and unforeseen events in context of pervasive environment. For example, how to do when a component Service is no longer available at runtime? Our approach can complement these efforts and overcome these issues. Indeed, it can be used to specify reliable and flexible compositions with a dynamic reconfiguration. Thus, if a component becomes unavailable, another Web service providing the same output and satisfy the same transactional requirements can automatically substitute it.

Advanced Transaction Models [6] are proposed to support new database applications by relaxing transaction isolation and atomicity to better match the new requirements. Their limitations come mainly from their inflexibility to incorporate different transactional semantics as well as different behavioral patterns into the same structured transaction [16]. In addition, Workflow systems [9], as the key technology for business process automation [17], lack rigorous mechanisms for reliability and correctness. To overcome these limitations, several works were proposed to ensure reliability of Web services compositions. [18] proposed a transactional Workflow system supporting multitask and multisystem activities where: (i) different tasks may have several execution behaviors or properties, (ii) designers define the coordination of the different tasks, and (iii) specify their required failure atomicity. In [3], the failure atomicity requirements are defined by specifying a set of Accepted Termination States (ATS). However, these proposals do not respond to unexpected events. Moreover, their main drawback is in the definition of ATS set by user, which is neither simple nor scalable.

VII. Conclusion

In this paper, we have proposed a solution to easily design reliable and flexible Web service composition in a pervasive environment. The main idea of our approach is to combine dynamic Workflow flexibility and transaction processing reliability. We introduce an extension of dynamic Workflow patterns, the dynamic transactional patterns, which can be seen as a convergence concept between dynamic Workflow systems and transactional models to easily define flexible and reliable composite Web services.

References

[1] *Programming Web services with SOAP*. O'reilly, 2009.

[2] F. Mustafa and T. L. McCluskey, "Dynamic web service composition," in *Proc. Int. Conf. Computer Engineering and Technology ICCET '09*, vol. 2, 2009, pp. 463–467.

[3] S. Bhiri, O. Perrin, and C. Godart, "Ensuring required failure atomicity of composite web services," in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 138–147.

[4] F. Montagut and R. Molva, "Augmenting web services composition with transactional requirements," in *Web Services, 2006. ICWS'06. International Conference on*. IEEE, 2006, pp. 91–98.

[5] V. S. W. Lam, "Dynamic workflow patterns," in *Enterprise Information Systems and Web Technologies*, 2008, pp. 160–166.

[6] A. K. Elmagarmid, "Transaction models for advanced database applications," 1991.

[7] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz, "A transaction model for multidatabase systems," in *ICDCS*, 1992, pp. 56–63.

[8] M. Graiet, I. Abbassi, L. Hamel, M. T. Bhiri, M. Kmimech, and W. Gaaloul, "Event-b based approach for verifying dynamic composite service transactional behavior," in *Proceedings of the 2013 IEEE 20th International Conference on Web Services*, ser. ICWS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 251–259. [Online]. Available: http://dx.doi.org/10.1109/ICWS.2013.42

[9] A. H. M. t. H. W. M. P. van der Aalst, A. P. Barros and B. Kiepuszewski., "Advanced workflow patterns." in *O. Etzion and Peter Scheuermann, editors, 5th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'00)*, vol. number 1901 in LNCS, 2000, pp. 18–29.

[10] N. Russell, A. H. Ter Hofstede, and N. Mulyar, "Workflow controlflow patterns: A revised view," 2006.

[11] S. Jablonski and C. Bussler, "Workflow management: modeling concepts, architecture and implementation," 1996. [Online]. Available: http://www.citeulike.org/group/6987/article/3401448

[12] A. Barros, M. Dumas, and A. H. Ter Hofstede, "Service interaction patterns," in *Business Process Management*. Springer, 2005, pp. 302–318.

[13] N. Russell, A. H. Ter Hofstede, D. Edmond, and W. M. van der Aalst, "Workflow data patterns: Identification, representation and tool support," in *Conceptual Modeling–ER 2005*. Springer, 2005, pp. 353–368.

[14] A. Ceponkus, S. Dalal, T. Fletcher, P. Furniss, A. Green, and B. Pope, "Business transaction protocol," *Change*, 2002.

[15] I. O. C. D. A. OUT, "Web services atomic transaction (ws-atomictransaction)," 2004.

[16] G. Nektarios and S. Christodoulakis, "Utml: Unified transaction modeling language," in *Web Information Systems Engineering, 2002. WISE 2002. Proceedings of the Third International Conference on*, 2002, pp. 115–126.

[17] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. Ngu, and A. K. Elmagarmid, "Business-to-business interactions: issues and enabling technologies," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 12, no. 1, pp. 59–85, 2003.

[18] M. Rusinkiewicz and A. P. Sheth, "Specification and execution of transactional workflows." *Modern database systems*, vol. 1995, pp. 592–620, 1995.