BPMN Modelling of Services with Dynamically Reconfigurable Transactions*

Laura Bocchi[1], Roberto Guanciale[2], Daniele Strollo[3], and Emilio Tuosto[1]

[1] Department of Computer Science, University of Leicester, UK
[2] Department of Computer Science, University of Pisa, Italy
[3] Istituto di Scienza e Tecnologie dellInformazione "Alessandro Faedo", CNR, Pisa, Italy

Abstract. We promote the use of *transactional attributes* for modelling business processes in the service-oriented scenario. Transactional attributes have been introduced in Enterprise JavaBeans (EJB) to decorate the methods published in Java containers. Attributes describe "modalities" that discipline the reconfiguration of transactional scopes (i.e., of caller and callee) upon method invocation.
An original element of our research programme is the definition and study of modelling and programming mechanisms to control dynamically reconfigurable transactional scopes in Service-Oriented Computing (SOC). On the one hand we give evidence of the suitability of transactional attributes for modelling and programming SOC transactions. As to a proof of concept we show how BPMN can be enriched with a few annotations for transactional attributes. On the other hand, we show how the results of a theoretical framework enable us to make more effective the development of transactional service-oriented applications.

1 Introduction

In this paper we promote the use of *transactional attributes* for modelling transactional business processes in service-oriented systems. An original element of our research programme is the definition and study of modelling mechanisms for dynamically reconfigurable transactional scopes in SOC.

The long-lasting and cross-domain nature of activities in service-oriented systems contributed in characterising a novel powerful paradigm but, on the other hand, imposed to re-think a number of classic concepts. Among these concepts, the notion of transaction had to be adapted in order to fit the requirements imposed by the of the *Service-Oriented Computing* (SOC) paradigm. SOC transactions, which are often referred to as long-running transaction, ensure a weaker set of properties (e.g., no atomicity nor isolation) with respect to the classic ACID transactions used in database systems; on the other hand they do not require to lock resources.

The investigation of formal semantics for SOC transactions has been a topic of focus in the last few years (see § 5 for a non-exhaustive overview). Central to this investigation is the notion of *compensation*, a weaker and "ad hoc" version of the classic rollback of database systems. Most of the research on long-running transactions has

* This work has been partially sponsored by the project Leverhulme Trust Award "Tracing Networks".

been focusing on providing suitable (formal) semantics of compensations while not much attention has been reserved to the inter-dependencies of failure propagation and dynamic reconfiguration. This is a very critical issue in SOC, where dynamic reconfiguration is one of the key characteristics. In fact, the configuration of a service-oriented system can change at each service invocation to include new instances of invoked services in the ongoing computation. Notably, the reconfiguration affects the relationships between existing and newly created transactional scopes.

To the best of our knowledge, these issues have been first considered in [2, 3, 1] where it was proposed a process calculus featuring *transactional attributes* (or attributes for short), inspired to the transactional mechanisms of EJB [14, 12]. The theoretical framework in [2, 3, 1] aims to analyse different semantics of transactional scope reconfiguration, the observable behaviour of service-oriented system, failure propagation, and transactional scope reconfiguration.

We contend that transactional attributes represent a useful conceptual device also in the modelling phases and during the development of transactional SOC applications. This paper proposes a methodology based on the features of transactional attributes and the theoretical framework of [2, 3, 1] to enable software architects and engineers to design and develop distributed applications with transactional properties in the service-oriented scenario. In this paper, we highlight the benefits of using transactional attributes for modelling and programming transactional service-oriented processes. As a proof of concept, we show how the Business Process Modelling Notation (BPMN) [8] can be enriched with transactional attributes and, through the use of a simple case study, we illustrate the suitability of an attribute-aware process modelling.

At design time, the software architect typically abstracts from the distribution of the activities, the communication mechanisms and the technologies that will implement each activity. Also, a BPMN specification can describe a distributed workflow and its transactional properties both from *local* and *global* points of view; as a matter of fact, developers can create service-oriented applications from BPMN models by exploiting different strategies. One strategy consists of modelling a service-oriented process by means of an (e.g. BPEL4WS) *orchestrator* by assembling tasks each representing a call to either a local or outsourced functionality (i.e., service). The orchestrator is a central entity that drives the interaction among services and manages their execution order. The other strategy has a more collaborative flavour and requires processes of BPMN designs to act as an ensemble that separately describe the role played by each participant. In this case there is no single "controlling" process managing the interactions and the activity of each process consists of both invocations to other services and to an interactive conversation with them.

In this paper we illustrate how the development from abstract BPMN designs can be improved by the usage of transactional attributes. Intuitively, attribute-aware designs give information on the transactional support required from the different tasks; noteworthy, such information can be exploited when developing applications distributively. In fact, transactional attributes provide a useful set of preconditions which helps in preventing what is commonly known as defensive programming for those functionalities to be developed internally to an organisation[1]. The use of transactional attributes let pro-

[1] In defensive programming the code is filled with unnecessary controls to validate data.

grammers to rely on a set of preconditions and assumptions on the transactional context in which the task will be executed thus (s)he is relieved form the burden of considering all possible cases. For functionalities provided by external parties, attribute-aware designs provide a useful set of non-functional requirements that allow to select the best match among the available services.

As a second contribution, we illustrate how the extension of BPMN (e.g., with constructs for service invocation/instantiation associated to transactional attributes and constructs for representing distributed transactions) allows to define, at design time, enough information for controlling the run-time reconfiguration of transactional scopes and to model all the possible scope configurations allowed by EJB.

Finally, we show how the results of a recently published theoretical framework [2, 3, 1] enable us to make more effective some common development activities of service-oriented applications. Actually, in [1] it has been proved that, under certain conditions, some of the proposed transactional attributes may be considered equivalent; this can be used to deduce that some scenarios may be ignored in during the development.

Structure of the paper In § 2 we summarise the key ingredients of our paper, namely BPMN, EJB attributes, and our case study. The description of our methodology and the use of attributes in BPMN designs is given in § 3. The extension of BPMN to incorporate service invocation and distributed transactional scope handling is reported in 4. Concluding remarks and considerations on future work are given in § 5.

2 Background

In this section we provide the background information which is used in the rest of the paper. In 2.1 we give a brief introduction of the BPMN notation. In 2.2 we give an intuition of the semantics of scope reconfiguration featured by EJB. Finally, in 2.3 we introduce the case study that will be used in this paper to illustrate the proposed approach.

2.1 The Business Process Modeling Notation

The Business Process Modeling Notation [8] (BPMN) allows us to describe business processes through a graphical notation. The main building blocks of a BPMN design are *flow objects*, which represent activities and events involved in a business process. As illustrated in Figure 1, BPMN processes involve two special events: the *starting point* of the business process graphically represented by an empty single-edge circle, and its *termination point* drawn as an empty double-edge circle; rounded-corner boxes represent tasks to be executed.

Exploiting LRTs abstraction, each BPMN task can be equipped with a *compensation* that is responsible to partially recover the task effects if the execution of the whole process cannot be completed. BPMN compensations are represented by tasks connected to the exception events by dotted dashed lines. For example, the process in Figure 2 describes that the compensation *CompA* can rectify the effects produced by the task *A*. Dashed arrows are used link a task with its corresponding compensation.

Fig. 1: BPMN process with one task

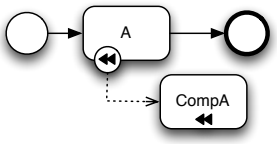


Fig. 2: A BPMN task with compensation

Arrows connecting *flow objects* represents their dependencies, usually referred to as *forward-flow*. Standard arrows describe the temporal order of execution of the business process as opposed to the *backward-flow*, that is the order of execution of compensations. In literature, the order in which compensation must be executed is referred as is usually takes as "the inverse order" of the forward-flow. If the execution of a task fails, the forward-flow is stopped and the backward-flow started by executing first the compensation of the most recent successfully executed task back to the initial one. The BPMN process in Figure 3 describes the usage of arrows to compose tasks and defin-

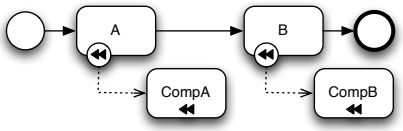


Fig. 3: Sequential composition of BPMN tasks

ing the forward-flow. The process models a sequence of two tasks: the task B can be executed only after the successfully termination of A.

Figure 4 models a concurrent process. After the termination of the task A, both the tasks B and C can be executed independently, the crossed box on the left is used to regulate their parallel activation. The crossed box on the right represents a synchronization barrier, waiting for the termination of all elements connected by an incoming arrow before to propagate the forward-flow execution.

BPMN permits to design nested transactions, as depicted in Figure 5. A double edge box represents a transactional scope. Intuitively, this scoping mechanism allows transactional scopes to hide any fault of a contained task to external processes.

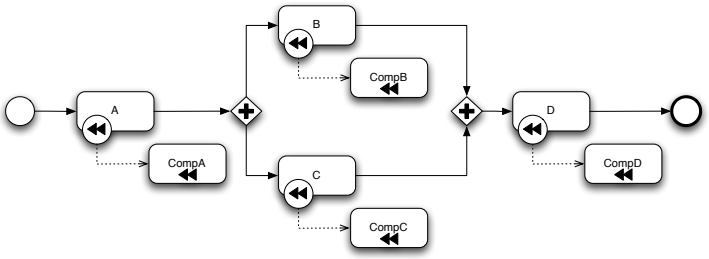


Fig. 4: BPMN concurrent tasks

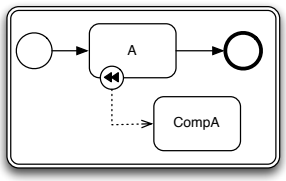


Fig. 5: BPMN sub-transaction

BPMN exploits "pools" to represent interactive participant in a Business to Business design. A pool can contain a process, which must be fully contained within the pool itself. Namely, forward-flow dependencies cannot cross pool (participant) boundaries. Interaction between participants is modeled via the Message flow (e.g. Figure 6). Similarly, transactional boxes cannot cross their pools.

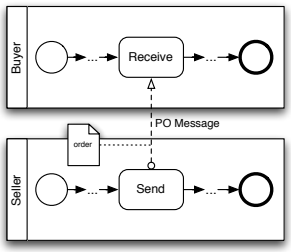


Fig. 6: BPMN pools

2.2 EJB Transactional Attributes

In EJB, objects can be assembled in a specialised run-time environment called *containers*, which can be thought of as configuration tools to set up a few characteristics of distributed applications at deploy time. A *Java bean* can be thought of as on object amenable to be executed in a container (see e.g., [14, 12]). An EJB container supports typical functionalities to manage e.g. the life-cycle of a bean and to make components accessible to other components by binding it to a naming service[2].

EJB containers feature a number of transactional features, namely *Container Managed Transactions* (CMT), whereby a container associates each method of a bean with a *transactional attribute* specifying the modality of reconfiguring transactional scopes. Namely, a transactional attribute determines:

- the transactional modality of the method calls. The modality expresses a requirement the invoking party must satisfy (e.g., "*calling the method* `fooBar` *from outside a transactional scope throws an exception*"),
- how the scope of transactions dynamically reconfigures (e.g., "`fooBar` *is always executed in a newly created transactional scope*").

We denote the set of EJB transactional attributes as

$$\mathcal{A} \stackrel{def}{=} \{\texttt{requires}, \texttt{requires new}, \texttt{not supported}, \texttt{mandatory}, \texttt{never}, \texttt{supports}\}.$$

The intuitive semantics of EJB attributes $\mathcal{A}$ is illustrated in table below, where each row represents the behaviour of one transactional attribute and shows how the transactional scope of the caller and callee behave upon invocation. Scopes are represented by a box, callers by $\bullet$, callee by $\circ$, and failed activities by $\otimes$. Each row shows the behaviour of one attribute. The first two columns show, respectively, invocations from outside and from within a scope.

invoker **outside** a scope	invoker **inside** a scope	callee supports
(1) $\bullet \Longrightarrow \bullet\,\boxed{\circ}$	$\boxed{\bullet} \Longrightarrow \boxed{\bullet\,\circ}$	`requires`
(2) $\bullet \Longrightarrow \bullet\,\boxed{\circ}$	$\boxed{\bullet} \Longrightarrow \boxed{\bullet}\,\boxed{\circ}$	`requires new`
(3) $\bullet \Longrightarrow \bullet\ \circ$	$\boxed{\bullet} \Longrightarrow \boxed{\bullet}\ \circ$	`not supported`
(4) $\bullet \Longrightarrow \otimes$	$\boxed{\bullet} \Longrightarrow \boxed{\bullet\,\circ}$	`mandatory`
(5) $\bullet \Longrightarrow \bullet\ \circ$	$\boxed{\bullet} \Longrightarrow \boxed{\otimes}$	`never`
(6) $\bullet \Longrightarrow \bullet\ \circ$	$\boxed{\bullet} \Longrightarrow \boxed{\bullet\,\circ}$	`supports`

More precisely, (1) a callee supporting `requires` is always executed in a transactional scope which happens to be the same as the caller's if the latter is already running in a transactional scope; (2) a callee supporting `requires new` is always executed in a new transactional scope; (3) a callee supporting `not supported` is always executed outside a transactional scope; (4) the invocation of a method supporting `mandatory`

[2] `http://docs.sun.com/app/docs/doc/819-3658/ablmw?a=view`

fails if the caller is not in a transactional scope (first column of the fourth row in the table, otherwise the method is executed in the transactional scope of the caller; (5) the invocation of a method supporting `never` is successful only if the caller is outside a transactional scope, and it fails if the caller is running in a transactional scope (in this case an exception is triggered in the caller); (6) a method supporting `supports` is executed inside (resp. outside) the caller's scope if the caller is executing in (resp. outside) a scope.

2.3 The Car Repair Case Study

A car manufacturer offers a service that supports the driver in case his/her car breaks down. Figure 7 illustrates a BPMN process modelling the car repair service.

The process is included in a transactional scope. Once the user's car breaks down, the system attempts to locate a garage, a tow truck and a rental car service so that the car is towed to the garage and repaired, and meanwhile the car owner may continue his travel.

First, before any service lookup is made, the credit card of the driver is charged with a security amount. Second, the process searches for a garage. The outcome of the search for the garage poses additional constraints to the candidate tow trucks. Third, a tow truck is called. If the search for a tow truck fails, the garage appointment must be revoked. The interdependencies between the bookings made by the service make it necessary to equip the orchestration with compensations. Fourth, a car rental (which must in a reasonable proximity with respect to the garage) is arranged. If renting a car succeeds and finding either a tow truck or a garage appointment fails, the car rental must be redirected to the broken down car actual location. We model *OrderRentalCar* inside a nested transactional scope because its failure should not affect the tow truck and garage appointments.

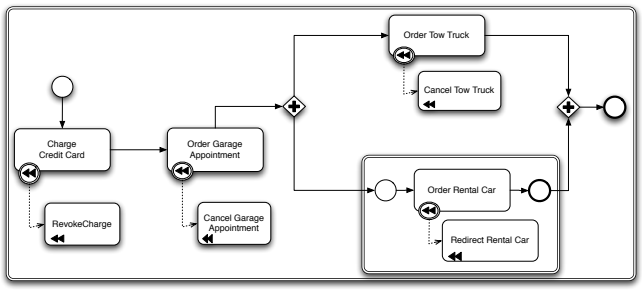


Fig. 7: The BPMN process for the Car Repair Scenario

The BPMN design in Figure 7 models the car repair case study by using a single BPMN pool. In fact, all activities and compensations are executed within a BPMN transactional scope, which can involve activities of only one participant. Namely, the

design describes an orchestration that, whenever executed by a *leader* participant, regulates the execution of partner services. In the following section we extend the BPMN model in order to simplify modeling more collaborative approaches.

3 Modelling Transactional Processes with BPMN and Attributes

We propose a methodology for designing distributed transactions in BPMN consisting of the following phases:

phase 1 definition of the design of the system
phase 2 specification of (compensation and) transactional attributes
phase 3 refinement of transactional aspects of the design.

Before clarifying our methodology, it is worth emphasizing that, in phase 1, designs are supposed to provide a *local view* of the system where activities are supposed to reside within a same pool and the coordination strategy relies on an orchestrator (which is specified by the design). For example the design in Figure 7 yields such a local view for the car repair scenario. The methodology could also be applied by using a diagram like the one in Figure 7 as high level model for a global distributed process with no central orchestrator, abstracting form the mechanisms used by the participants to coordinate. This would require further extensions of BPMN; for instance, instead of using forward-flow connectors, a new kind of connector subsuming both message and sequence flows. Such model could be then transformed into a global, more detailed, design where tasks are partitioned into pools and their coordination explicitly represented. In the next section we discuss global designs where service invocations and distributed scope handling are introduced.

A key element of transactional attributes is the separation of concerns they provide; EJB programmers, in fact, may develop their code independently of the transactional behaviour because attributes do not directly affect the behaviour of an object. Likewise, our methodology capitalises on this separation of concerns and makes attributes largely independent of other aspects of designs. In other words, transactional attributes can be specified once the software architect has designed the system. In a certain sense, we extend the design strategy that BPMN enables on compensations to attributes; indeed the BPMN architect could in principle give an initial model of a system without considering transactional aspects and introduce compensations at a later stage. Similarly, phase 2 of our methodology allows us to decorate designs with attributes as well as refine them (if necessary) subsequently.

For the moment, we just decorate BPMN designs with attributes assuming their intuitive semantics (cf. 2.2); in § 4 we extend BPMN so to illustrate the semantics of each attribute in a more precise way.

An attribute-aware design is simply a design where forward-flow arrows are labelled with a finite number of attributes. The idea is that if two activities T and T' are connected by a forward-flow arrow decorated with a set of attributes $A = \{\mathtt{a}_1, \ldots, \mathtt{a}_n\}$, the control is passed from T to T' if the latter "supports" at least one of the attributes in A. In other words, T and T' explicitly "agree" on their combined transactional behaviour; T requires T' to behave according to the set of transactional behaviours specified by

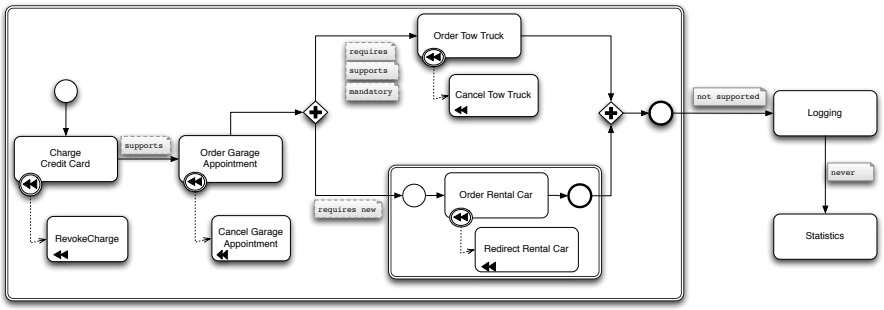


Fig. 8: Extended Car Rental with Attributes Annotations

A and, dually, T' guarantees that it is supporting some of such behaviours. Put in other terms, T' is aware of the transactional requirements of T and T is aware that the possible transactional behaviours of T' are included those specified by $\mathtt{a}_1, \ldots, \mathtt{a}_n$.

We illustrate our methodology and the use of how to add transactional attributes to BPMN designs considering the car repair scenario described in § 2.3. First, we extend the design of Figure 7 to incorporate the invocation of a new logging service which outsources the calculations of some statistics; such extension allows us to explain the utility of the attributes `not supported` and `never`. The idea is that, upon completion of the transaction, some logging information are sent to a logging service which also calculates some statistics by means of an external service.

The extended design is given in Figure 8 together with the transactional attributes. The BPMN design stays the same as in § 2.3 except for the newly added tasks and the attributes decorating the connectors. For simplicity, we assign a single attribute to each arrow except to the connector for invoking the tow truck which has three attributes.

The assignment exploits all the attributes.

- The attribute on the connector between the credit card check and garage activities is `supports`; it stipulates that the latter activity should be capable of engaging in the same transaction (if any) of the former one. In our scenario, since the credit card check is executed in a transactional scope, the garage endpoint will be part of the transaction of the check task.
- Once the garage is fixed, a car rental able to start a new transaction is invoked; the attribute `requires new` specifies that the rental endpoint will be activated in new transaction whose failure will not cause the failure of the main transaction. Notice that in the global view, this correspond to confine the rental service in a sub-transaction however, in the local view, the car rental will be a remote transactional transactional scope (this issue will be considered in § 4).
- in parallel to the car rental activity, a two truck is searched; such service is required to support either of the attributes `mandatory`, `supports`, or `requires`; in fact, the expected behaviour is that a failure of the tow truck order has to trigger the failure of the whole transaction and such attributes will let the tow truck endpoint to be added in the scope of the transaction. Hence, a service supporting any of the remaining attributes should be ruled out.

- The logging service, on the other hand, is specified to support the `not supported` attribute that will execute the logging endpoint outside any transaction as its failure is not crucial and should not affect the other transactional activities.
- The attribute `never` assigned to the statics service is instead specifying that the statistics service should never be invoked from a transaction.

It is worth to remark that, if an orchestrator were due to realise the workflow, it will also take into account possible failures of services and react according to the attributes assign to the invocation; in other words, the orchestrator will also act as (or liaise with the) transaction manager.

Once attributes have been assigned to a design, the phase 3 of our methodology would allow some refinement both at the design level and at later stages of the development. This refinements hinge on the theoretical framework of [1] where it has been proved that, in certain contexts, some attributes exhibit the same observational behaviour. Again we illustrate this on our running scenario.

Since an external observer cannot distinguish services supporting `mandatory`, `supports`, or `requires` when they are invoked from transactional scopes, the design of Figure 8 can be refined into one where only one of the attributes is assigned to the invocation of the tow truck service. Notice that this may provide a great simplification in the realisation of the design. For instance, the team developing the tow truck order may avoid to consider testing cases corresponding to the different attributes and chose the attribute that guarantees the smoother development.

Also, in a more complex scenario an activity may be invoked from many different other activities with different transactional requirements. The equivalences proved in [1] may again help in the refinement phase as the developers may factor out the invocations with equivalent attributes so limiting the development efforts.

4 Modelling Attribute-Based Services Invocation and Instantiation

An important feature in the service-oriented scenario is to allow invokers to specify their own requirements on the invoked method. In fact, a service invocation does not target a specific service but is resolved at run-time by selecting one of the available implementations matching a given description. Typically, both the service requester and the service provider express a set of requirements that need to be matched when defining the Service Level Agreement (SLA) for an invocation.

We consider a generalisation of the EJB mechanism for managing scope reconfiguration allowing both service requester and service provider to specify the transactional modality of each service instantiation. Namely, a service-oriented system is described as follows:

1. a process can contain a number of service invocations. Each service invocation specifies an abstract reference s[3]. Furthermore, service invocations specify a set of acceptable attributes, say $A \subseteq \mathcal{A}$. The execution of the invocation triggers, at run-time, the discovery/selection/binding of a service matching with s and A. Figure 9

[3] The service reference s can be thought as a service description that specifies the desired functional properties. Hereafter we refer to s simply as a *service*.

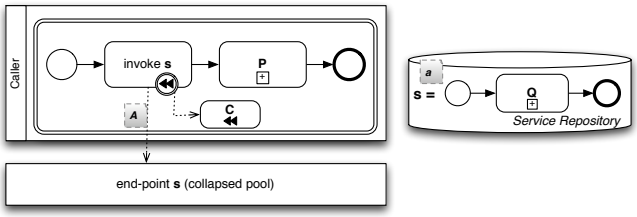


Fig. 9: Invocation of end-point **s**

(left-hand side) represents the process "*caller*" consisting of a transactional scope. The scope includes task `invoke` s, associated to compensation C, followed by sub-process P.

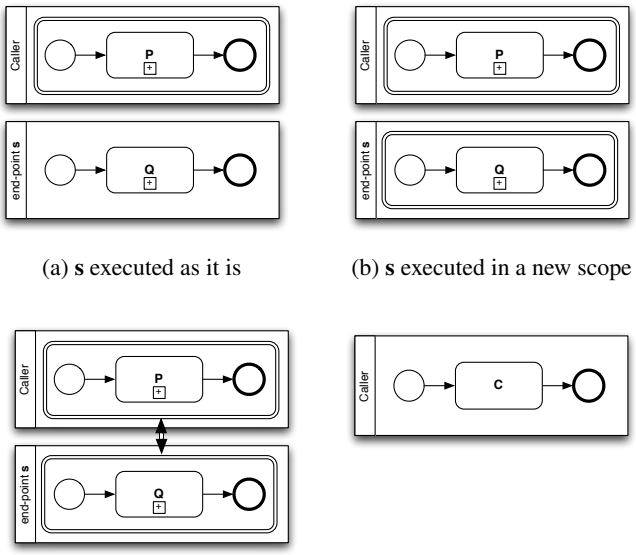
2. each provider can publish a number of services. The provider associates each service s to an implementation (e.g., a process) and a transactional attribute (e.g., $\mathtt{a} \in \mathcal{A}$). In Figure 9 the provider, represented by the repository o the right-hand side, implements s as a process that consisting of a sub-process Q, and associates it to attribute a.
3. upon service invocation, the matching/binding between caller and callee on s happens only if $\mathtt{a} \in A$. In this case, the invocation of s triggers a (possibly remote) new instance s.

In other words, we threat attributes as non-functional properties that can be included as part of the SLA.

We present below a few examples of service invocation with attributes using a smooth extension of BPMN and following the semantics of EJB attributes. The BPMN extension will be described in more detail in the second part of this section. Intuitively, the system in Figure 9 may evolve to a number of different configurations depending on which attribute a the provider associates to s[4]. The possible configurations follow from the semantics of transactional attributes illustrated in Section 2.2, when the invocation occurs inside a transactional scope.

- If $\mathtt{a} =$ `not supported` then the new instance of s is be executed "as it is" outside any additional transactional scope. In this case the provider, by associating s with a specifies that no transactional support i provided and, reversely, the requester accepts such condition by including `not supported` in A. Figure 10a shows the reached configuration where the activity Q is executed outside any transactional scope.
- If $\mathtt{a} =$ `requires new` then the new instance of s is executed in a different, newly created scope. In this case both provider and requester agree on the fact that the service will be executed in a newly created scope. Figure 10b shows the reached configuration where the activities P and Q are executed in different transactional scopes.

[4] Recall that the reconfiguration semantics, according to EJB, is decided by the provider through the definition of attribute a. We allow also the service requester to specify the desired semantics by defining the set A.



(a) **s** executed as it is

(b) **s** executed in a new scope

(c) **s** executed in the caller's scope

(d) The invocation triggers the execution of the compensation

Fig. 10: Scope reconfigurations following from the invocation of *s* in Figure 9

- If $\mathtt{a} = \mathtt{requires}$ or $\mathtt{a} = \mathtt{mandatory}$ or $\mathtt{a} = \mathtt{supported}$ then the caller and the callee are executed in the same scope of the caller. Namely, caller and callee participate to the same distributed transaction. In Figure 10c both activities P and Q are executed in the same scope.
- If $\mathtt{a} = \mathtt{never}$ then the invocation raises an exception which causes the execution of the compensation C.

Notice that teach alternative is desirable under certain circumstances, and has an impact on failure propagation. The configurations deriving from the invocation of *s* from outside a transactional scope can be defined similarly.

As mentioned before we used a smooth extension to BPMN to represent attribute aware service instantiation and distributed transactions.

Attribute-aware service instantiation. We represent the instantiation of a service as a task *invoke s* that sends a message to a collapsed pool. The collapsed pool represents the abstract reference of the service to invoke. In general, the caller process can have an interactive conversation with the invoked services which is modelled as a message exchange (not shown in the examples for simplicity) with the collapsed pool. We extend BPMN message flow by annotating the messages that spawn new processes to allow the caller to specify transactional requirements. Several implementations of a service can be available, each of them implemented by a different sub-process and satisfying different

transactional properties. Upon an activation request, the callee activates only one of the processes satisfying caller requirement inside the right transactional scope.

Distributed transactions. Even if BPMN directly supports design of transactional processes, this features is limited to activities owned by a unique participant (e.g. confined into a pool). Architects must care about exceptions and explicitly model interactions performed by participants to implement distributed transactions. Moreover, BPMN provides a limited support to model service requirements and dependencies. In SOC systems participants dynamically activates partners services (e.g. processes) that must respect the global transactional requirements.

To handle these issues we propose to extend BPMN to support the expressiveness of [2, 3]. In order to represent distributed transaction, we introduce the double arrow connection. This artifact allows to represent that two (or more) participant transactions are confined by the same scope. Namely, a fault of one participant activity can automatically start the backward-flow of the linked transactions. Equipping BPMN with the distributed transaction artifact allows designer to statically define non functional properties of systems abstracting from the implementation details of the mechanisms for synchronising the outcomes.

In fact, this artifact abstract from the interactions required among participant to implement the correct behavior, that is demanded to participant SOC frameworks (e.g. WS-Tx [11] and BTP [10]).

5 Concluding Remarks and Related Work

We proposed an approach for the modelling of transactional service-oriented business processes that centres on the notion of transactional attribute. Our approach

1. models the EJB mechanisms for the management of transactional scopes upon invocation. Notably this allows a straightforward implementation of models as Java programs.
2. is based on a generalisation of EJB transactions, that have been adapted to the service-oriented scenario and can be used for the development of models in different technologies.
3. builds on the theoretical results in [2, 3] that allow to enhance and optimise software development and testing.
4. can be adapted to a number of notations. As a proof of concept we have shown our approach integrated with BPMN.

Service development can benefit from transactional attributes for a number of reasons. In the implementation phase, transactional attributes provide the developer with a stronger set of preconditions on the (transactional) context in which a piece of software will be executed thus preventing defensive programming.

A service provider may want to publish different versions of the same service guaranteeing different transactional properties. This would allow the provider to maximise the number of matches for his/her portfolio of services. In [1] we proved equivalence of different transactional attributes under specific context. Starting from a registry that

describes service versions, each associated with one transactional attribute, our theoretical results allow drive the implementation of the minimum set versions required to respect the BPMN design. Also, relying on the testing theory in [1] we can ease the testing of services-oriented artifacts since under certain conditions different transactional configuration have the same observed behaviour.

Finally we proposed an extension to the BPMN notation that focuses on service invocation and defines how the system should reconfigure at run-time. Notice that the structure of the transactional scopes describe the configuration of the whole execution of the transaction but does not details on how such configuration is achieved. Anyway, the information included in the models annotated with attributes can be used to define a semantics for dynamically reconfiguring BPMN processes. An interesting approach, that we leave as a future work, would be to use graph rewriting [13] to this aim.

Related Work. A number of formal models for long running transactions have been proposed in the last years. Saga [7] is one of the earlier proposal to manage LRTs by exploiting the notion of compensations. A recent work [9] provides a comparison of the expressiveness of different approaches to specify compensations. A formal model for failure propagation in dynamically reconfiguring systems has been proposed in [2] as a CCS-like process calculus called ATc (after *Attribute-based Transactional calculus*). The primitives of ATc are inspired to EJB [14] and allow to *determine* and *control* the dynamic reconfiguration of distributed transactions so to have consistent and predictable failure propagation. In [3] it has been proposed an observational theory (based on the theory of testing in [4]) yielding a formal framework for analysing the interplay between communication failures and the observable behaviour of a service-oriented system. In fact, the main result in [3] shows that the choice of different transactional attributes causes different system's behaviours and system's reactions to a failure. A comparison of the linguistic features of ATc wrt other calculi featuring distributed transactions has been given in [2].

BPMN allows to statically define the transactional activities. More expressive models has been investigated. For example the *dynamic recovery* approach allows compensations to be dynamically updated and replaced. It should be interesting evaluating effectiveness of BPMN artifacts to express dynamic recovery. A number of work tackled the lack of a formal semantics for BPMN (e.g., [6, 16]). Our aim was rather to propose a methodology for the design of transactional processes that relies on a theoretical framework. The methodology centres on the fact that service invocations cause a reconfiguration of transactional scopes in a service-oriented scenario. Up to our knowledge, such aspects have not been included by existing formal models of BPMN. A promising approach to provide a formal account of BPMN in reconfiguring system would be to use graph rewriting techniques [13]. Some other work address the execution of BPMN models (e.g., [5] encodes them in executable YAWL [15] processes). We address an orthogonal issue by proposing a general approach for attribute-aware software development, that can be applied to many development techniques.

References

1. L. Bocchi and E. Tuosto. A Java Inspired Semantics for Transactions in SOC (extended report), 2009. Available at `http://www.cs.le.ac.uk/people/lb148/javatransactions.html`.
2. L. Bocchi and E. Tuosto. A Java inspired semantics for transactions in SOC. In *TGC 2010*, LNCS. Springer-Verlag, 2010. To appear.
3. L. Bocchi and E. Tuosto. Testing attribute-based transactions in SOC. In *FORTE 2010*, LNCS. Springer-Verlag, 2010. To appear.
4. R. De Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34(1–2):83–133, Nov. 1984.
5. G. Decker, R. Dijkman, M. Dumas, and L. García-Ba nuelos. Transforming BPMN Diagrams into YAWL Nets. In *BPM '08: Proceedings of the 6th International Conference on Business Process Management*, pages 386–389, Berlin, Heidelberg, 2008. Springer-Verlag.
6. R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information & Software Technology*, 50(12):1281–1294, 2008.
7. H. Garcia-Molina and K. Salem. Sagas. In U. Dayal and I. L. Traiger, editors, *SIGMOD Conference*, pages 249–259. ACM Press, 1987.
8. O. Group. Business Process Modeling Notation. `http://www.bpmn.org`, 2002.
9. I. Lanese, C. Vaz, and C. Ferreira. On the expressive power of primitives for compensation handling. In A. D. Gordon, editor, *ESOP*, volume 6012 of *Lecture Notes in Computer Science*, pages 366–386. Springer, 2010.
10. Business Transaction Protocol (BTP), 2002.
11. Web Services Transaction (WS-TX), 2009.
12. D. Panda, R. Rahman, and D. Lane. *EJB 3 in action*. Manning, 2007.
13. G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation: volume I. foundations*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
14. Sun Microsystems. Enterprise JavaBeans (EJB) technology, 2009. `http://java.sun.com/products/ejb/`.
15. W. van der Aalst and A. H. M. T. Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30:245–275, 2003.
16. P. Y. Wong and J. Gibbons. A Process Semantics for BPMN. In *ICFEM '08: Proceedings of the 10th International Conference on Formal Methods and Software Engineering*, pages 355–374, Berlin, Heidelberg, 2008. Springer-Verlag.