Architectural Decisions and Patterns for Transactional Workflows in SOA

Olaf Zimmermann[1], Jonas Grundler[2], Stefan Tai[3], and Frank Leymann[4]

[1] IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
olz@zurich.ibm.com
[2] IBM Software Group, Schönaicher Strasse 220, 71032 Böblingen, Germany
jonas.grundler@de.ibm.com
[3] IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA
stai@us.ibm.com
[4] Universität Stuttgart, IAAS, Universitätsstraße 38, 70569 Stuttgart, Germany
frank.leymann@iaas.uni-stuttgart.de

Abstract. An important architectural style for constructing enterprise applications is to use transactional workflows in SOA. In this setting, workflow activities invoke distributed services in a coordinated manner, using transaction context-propagating messages, coordination protocols, and compensation logic. Designing such transactional workflows is a time-consuming and error-prone task requiring deep subject matter expertise. Aiming to alleviate this problem, we introduce a new analysis and design method that (a) identifies recurring architectural decisions in analysis-level process models, (b) models alternatives for these decisions as reusable, platform-independent patterns and primitives, and (c) maps the patterns and primitives into technology- and platform-specific settings in BPEL and SCA. Our method accelerates the identification of decisions, empowers process modelers to make informed decisions, and automates the enforcement of the decisions in deployment artifacts; tool support is available. We demonstrate value and feasibility of our method in an industry case study.

Keywords: BPEL, BPM, patterns, transactions, MDA, SCA, SOA, workflow.

1 Introduction

Service-Oriented Architecture (SOA) with transactional workflow support is a state-of-the-art architectural style for constructing enterprise applications. In this context, enterprise resources such as databases and message queues are exposed as distributed services, which are invoked concurrently by diverse service consumers including end user applications and executable workflows. The *integrity* of the enterprise resources must be preserved at all times [4]. System-level transaction techniques such as Atomicity, Consistency, Isolation, and Durability (ACID) transactions and business-level solutions such as compensation-based recovery are two ways of addressing this requirement [8]. However, defining transaction boundaries and implementing compensation logic are complex, time-consuming, and error-prone tasks requiring deep subject matter expertise. Neither reusable architectural patterns nor methodological support exist today; development tools do not guide process modelers sufficiently. This lack of support is diametrically opposed to SOA design goals such as increased agility, flexibility, and reusability – in our opinion, a key inhibitor for real-world adoption of transactional workflows in SOA.

In this paper, we introduce a new analysis and design method that aims to eliminate this inhibitor by combining architectural decision modeling techniques, reusable patterns composed of primitives, and mappings of the primitives to concrete technologies such as the Business Process Execution Language (BPEL) and the Service Component Architecture (SCA). Our method covers the entire lifecycle from analysis to conceptual design to technology selection and runtime engine configuration. This end-to-end coverage speeds up the identification of design alternatives for transactional workflows in SOA and helps to make the decision making process repeatable; architectural knowledge can be shared across project and technology boundaries. Pattern-aware design tools can map the primitives to platform-specific technology specifications and deployment artifacts, e.g., in BPEL/SCA engines and other middleware.

The remainder of this paper is organized in the following way. Section 2 defines the context for our work. Section 3 scopes the problem to be solved by identifying recurring architectural decisions in a real-world case study. Section 4 defines three conceptual transaction management patterns and three underlying primitives, along with an exemplary technology- and asset-level transformation. Section 5 discusses related work, and Section 6 concludes with a summary and an outlook to future work.

2 Background

The objective of our work is to support the design and development of *enterprise applications* that require transactional semantics. An example is a Customer Relationship Management (CRM) system that serves many concurrent users via multiple access channels and processes, including an Internet self-service and a call center. In this CRM, business-relevant customer profile information is persisted in databases and accessed via Web-accessible services; external systems also have to be integrated.

SOA and Web services. SOA reinforces general software architecture principles such as separation of concerns and logical layering. A defining element of SOA as an architectural style is the possibility to introduce a *Service Composition Layer (SCL)* [18], which promises to increase flexibility and agility and to provide better responsiveness to constantly changing business environments. (Re-)assembling workflows in the SCL does not cause changes on the underlying service and resource layers; computational logic and enterprise resource management are separated from the service composition. We refer to a SOA with such a SCL as a *process-enabled SOA*.

XML-based Web services are a state-of-the-art implementation option for process-enabled SOAs [19]. The Web Services Description Language (WSDL) [15] describes service interfaces, SOAP [12] service invocation messages. BPEL [14] is a workflow language with operational semantics that can be used to realize the SCL. Component models for the implementation of services are emerging; SCA is such a model [9]. Service components in SCA are defined from several perspectives: an *interface* describing the input and output parameters of the operations of a component, *references* to other components, and component *implementations*. Via *imports*, a component implementation can reference external services.

In the CRM example, let us assume that process-enabled SOA has been chosen as the architectural style. The business processes to be implemented are modeled explicitly during requirements analysis; their execution as SCL workflows is later automated using a *BPEL engine*. The tasks in the processes are realized as atomic and composed Web services, which have a WSDL interface and can be invoked at runtime through transport protocol bindings, e.g., SOAP/HTTP. We further assume that these services are implemented as SCA components or integrated via SCA imports.

Transactional workflows. In the CRM system, relational database tables and message queues provided by integration middleware [5] serve as *enterprise resources* persisting and exchanging customer profiles. Concurrent and distributed access to transactional enterprise resources can be coordinated by *transaction managers*, which are in charge of ensuring the ACID properties; Relational Database Management Systems (RDBMS) and queue managers then take a local *resource manager* role subordinate to a transaction manager [8].

The SCL in a process-enabled SOA can be seen as a workflow application. If a BPEL engine in the SCL serves as a transaction manager, its process flows become *transactional workflows* [8]. Transactional workflows coordinate the outcome of the local and remote service invocations that access and manipulate the enterprise resources. Transactional workflows in process-enabled SOA are particularly challenging to design due to the potentially long-lived nature of processes, the loose coupling and autonomy of services, the existence of non-transactional resources, and the diversity in coordination and communication protocols (synchronous and asynchronous message exchange patterns). Traditional system transactions alone are not directly applicable in a SOA setting; a more decentralized *coordination* model and application-level *compensation* strategies have to be added. To address these needs, WS-Coordination, WS-AtomicTransaction (WSAT), and WS-Business Activity Framework (WBAF) complement the Web services specifications introduced above [17].

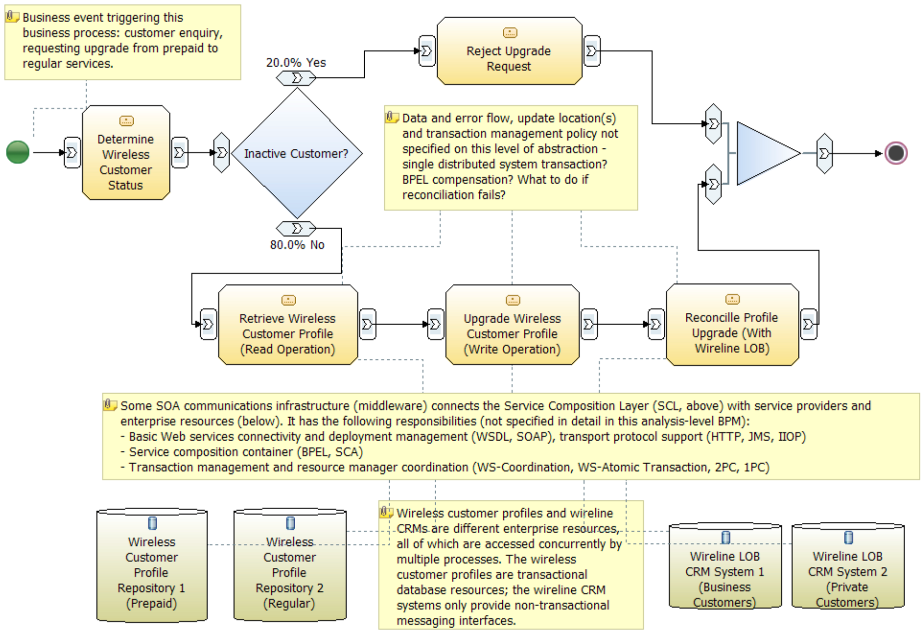
3 Recurring Architectural Decisions in Process-Enabled SOA

Today's SOA tools use default transaction management settings when translating analysis-level process models into BPEL workflows, Web services and SCA components [20]. Often, these settings are inappropriate and have to be changed during the later development steps. This is error-prone, platform-specific work; software quality issues arise and technical project risk increases. This problem can be overcome by:

A method for the systematic design of transaction management settings in process-enabled SOA, which (a) identifies the required architectural decisions in analysis-level process models, (b) captures proven design options as patterns which facilitate the decision making, and (c) transforms the patterns to platform-specific settings.

Sample process. Refining our CRM example, we now discuss the SOA enablement of an existing system of a telecommunication service provider that is organized into several Lines of Business (LOB), including wireline and wireless telephony. The business event triggering the sample process is a customer requesting an upgrade from prepaid to regular wireless service, e.g., by calling a call center agent.

Figure 1 outlines this key business process in the CRM system, *Upgrade Customer*:


Reject Upgrade Request
20.0% Yes
Determine Wireless Customer Status
Inactive Customer?
80.0% No
Retrieve Wireless Customer Profile (Read Operation)
Upgrade Wireless Customer Profile (Write Operation)
Reconcille Profile Upgrade (With Wireline LOB)
Wireless Customer Profile Repository 1 (Prepaid)
Wireless Customer Profile Repository 2 (Regular)
Wireline LOB CRM System 1 (Business Customers)
Wireline LOB CRM System 2 (Private Customers)

Business event triggering this business process: customer enquiry, requesting upgrade from prepaid to regular services.

Data and error flow, update location(s) and transaction management policy not specified on this level of abstraction - single distributed system transaction? BPEL compensation? What to do if reconciliation fails?

Some SOA communications infrastructure (middleware) connects the Service Composition Layer (SCL, above) with service providers and enterprise resources (below). It has the following responsibilities (not specified in detail in this analysis-level BPM):
- Basic Web services connectivity and deployment management (WSDL, SOAP), transport protocol support (HTTP, JMS, IIOP)
- Service composition container (BPEL, SCA)
- Transaction management and resource manager coordination (WS-Coordination, WS-Atomic Transaction, 2PC, 1PC)

Wireless customer profiles and wireline CRMs are different enterprise resources, all of which are accessed concurrently by multiple processes. The wireless customer profiles are transactional database resources; the wireline CRM systems only provide non-transactional messaging interfaces.

Fig. 1. Sample CRM process: analysis-level BPM including enterprise resources

The analysis-level Business Process Model (BPM) specifies that first the customer status has to be determined (*Determine Wireless Customer Status*), so that the customer profile, an enterprise resource spread over several repositories, can be retrieved (*Retrieve Wireless Customer Profile*). Next, a tentative *Upgrade Wireless Customer Profile* task is executed; however, the status change can only be finalized if a subsequent *Reconcile Profile Upgrade* task completes successfully. This task sends approval request messages to the two CRM systems of the wireline LOB. If any of these CRM systems declines the upgrade or does not respond within a working day, the upgrade process has failed, and the wireless customer profile must remain unchanged. Other business processes work with the customer profile while this process is running.

An analysis-level BPM such as Figure 1 is typically created by a business domain expert, not a software architect or workflow technology specialist. Such a BPM is not directly executable in a workflow engine; typically it does not cover design concerns such as data flow, resource protection, and error handling sufficiently. In the CRM example, the customer profiles are the enterprise resources to be protected with system and/or business transactions.[1] Another transactional enterprise resource might be the process instance state maintained by the engine; a BPEL engine in transaction manager role may have to roll back process parts when handling errors, even if

[1] Not all resources have to be protected by transactions, e.g. immutable resources meet the ACID characteristics trivially. On the other hand, not all resources worth protecting can actually be protected by transaction managers, e.g., due to legacy system constraints.

activities that do not participate in the same transaction (the one in which the BPEL process runs) have been committed on the system level already.

Recurring architectural decisions. It is technically feasible to transform the analysis-level BPM from Figure 1 into a design-level process model, e.g., via basic BPEL/SCA export utilities provided by commercial SOA tools [7]. However, such predefined transformations do not obey any *architectural decisions* that are made in response to project-specific requirements [20]. Many of these architectural decisions must be made for any process-enabled SOA, not just our CRM example: Which *composition paradigm* and *resource protection* approach should be selected? Who *coordinates* the transactions? Which *invocation protocols* are best suited to invoke services from the process activities in the SCL? Should the process activities and the service invocations run in separate *transaction islands* or form a *transaction bridge*? Which *compensation technology* should be used, and where should it be *placed*?

As step (a) of our method, Figure 2 organizes these recurring decisions by their *abstraction level* and *scope*. The abstraction level refines from conceptual issues such as selection of a composition paradigm (here: workflow) to technology and asset selection (here: BPEL language and BPEL engine). The scope of a decision assigns it to design model elements; in the CRM example, the activity *transactionality* has to be decided for Reconcile Profile Upgrade and the other four tasks shown in Figure 1.

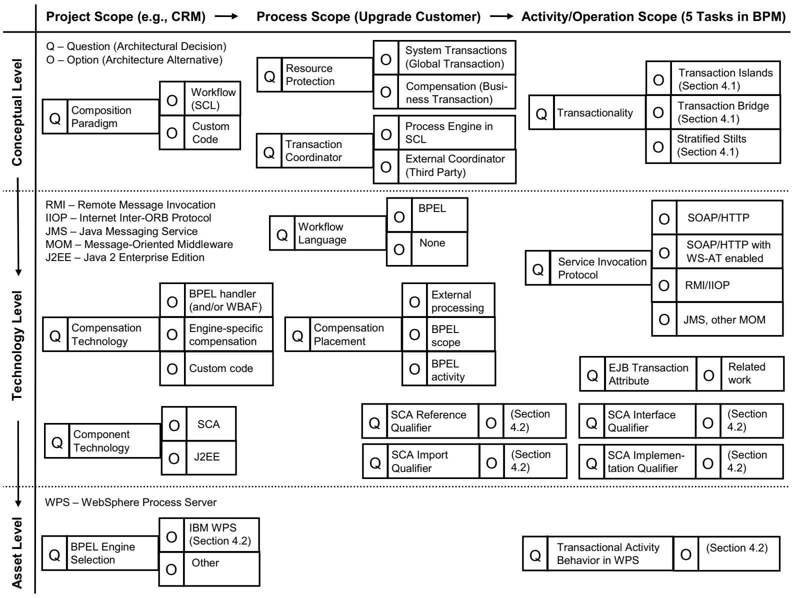

Project Scope (e.g., CRM)
Process Scope (Upgrade Customer)
Activity/Operation Scope (5 Tasks in BPM)
Conceptual Level
Technology Level
Asset Level
Q – Question (Architectural Decision)
O – Option (Architecture Alternative)
Q Composition Paradigm
O Workflow (SCL)
O Custom Code
Q Resource Protection
O System Transactions (Global Transaction)
O Compensation (Business Transaction)
Q Transaction Coordinator
O Process Engine in SCL
O External Coordinator (Third Party)
Q Transactionality
O Transaction Islands (Section 4.1)
O Transaction Bridge (Section 4.1)
O Stratified Stilts (Section 4.1)
RMI – Remote Message Invocation
IIOP – Internet Inter-ORB Protocol
JMS – Java Messaging Service
MOM – Message-Oriented Middleware
J2EE – Java 2 Enterprise Edition
Q Workflow Language
O BPEL
O None
Q Service Invocation Protocol
O SOAP/HTTP
O SOAP/HTTP with WS-AT enabled
O RMI/IIOP
O JMS, other MOM
Q Compensation Technology
O BPEL handler (and/or WBAF)
O Engine-specific compensation
O Custom code
Q Compensation Placement
O External processing
O BPEL scope
O BPEL activity
Q EJB Transaction Attribute
O Related work
Q Component Technology
O SCA
O J2EE
Q SCA Reference Qualifier
O (Section 4.2)
Q SCA Import Qualifier
O (Section 4.2)
Q SCA Interface Qualifier
O (Section 4.2)
Q SCA Implementation Qualifier
O (Section 4.2)
WPS – WebSphere Process Server
Q BPEL Engine Selection
O IBM WPS (Section 4.2)
O Other
Q Transactional Activity Behavior in WPS
O (Section 4.2)

Fig. 2. Architectural decisions and alternatives for transactional workflows in SOA

4 Architectural Patterns as Decision Alternatives

As step (b) of our method, we now introduce three conceptual patterns as solution options (architecture alternatives) for the activity *transactionality* decision from

Figure 2. These conceptual patterns comprise of platform-independent primitives that we map to BPEL and SCA technology and engine deployment artifacts in step (c). The primitives are designed in such a way that other mappings can also be provided.

4.1 Conceptual Patterns and Primitives

The tasks from Figure 1 require different transactional treatment: Determine Wireless Customer Status does not change any enterprise resource; transactional execution is not required. The retrieval should execute as fast as possible. Upgrade Wireless Customer Profile updates wireless customer profiles; the service operation is co-located with that realizing the Retrieve Wireless Customer Profile task. Changes must be executed with all-or-nothing semantics. The CRM systems contacted in Reconcile Profile Upgrade offer messaging interfaces and may take days to respond. Still, all-or-nothing semantics is required; if any of the reconciliation request messages returns an error or times out, the updates to the wireless customer profile made by Upgrade Wireless Customer Profile must be undone.

TRANSACTION ISLANDS, TRANSACTION BRIDGE, and STRATIFIED STILTS are three patterns commonly used to address resource protection requirements such as those in the CRM. In theory, more design options exist; however, faithful to established pattern capturing principles, we only present patterns observed and proven in practice.

Figure 3 illustrates the patterns on an abstract level; a more detailed pattern description follows later. The SCL is represented by the white boxes. It implements the tasks in the analysis-level BPM as *process activities* that are part of executable workflows; two invoke activities I_1 and I_2 enclose a third activity U, which for example may be a BPEL assign activity or another utility. S_1 and S_2 represent *service providers* exposing operations. Service operation invocations are displayed as dotted lines. A contiguous light grey area represents a single *global transaction* as defined in [8], which may be extended if it is not enclosed by a solid black line.

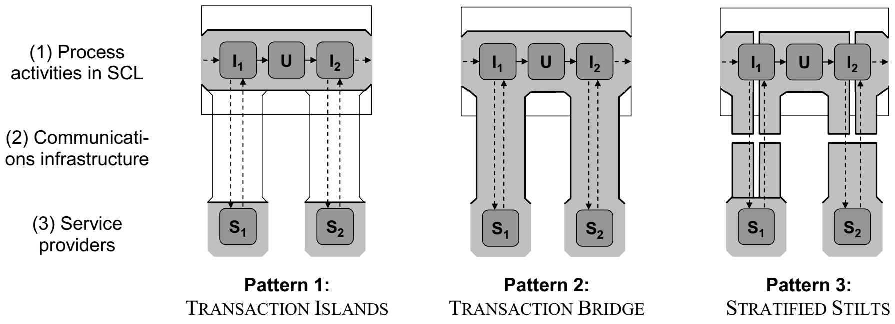

(1) Process activities in SCL
(2) Communications infrastructure
(3) Service providers
I1
U
I2
S1
S2
Pattern 1: TRANSACTION ISLANDS
Pattern 2: TRANSACTION BRIDGE
Pattern 3: STRATIFIED STILTS

Fig. 3. Transaction context sharing options between process activities and service operations

These patterns comprise of three types of *primitives* that correspond to the architectural layers from Figure 3: (1) Process Activity Transactionality (PAT) primitives for the *process activities* in the SCL. (2) Communications Transactionality (CT) primitives modeling the capabilities of the *communications infrastructure* (invocation protocol, component technology). (3) Service provider Transactionality (ST) primitives stating the capability and willingness of *service providers* to join a transaction.

These primitive types are conceptual, platform-independent abstractions of concepts for example found in today's BPEL/SCA technology, and can be viewed as design time statements of architectural intent. Figure 4 illustrates the primitives:

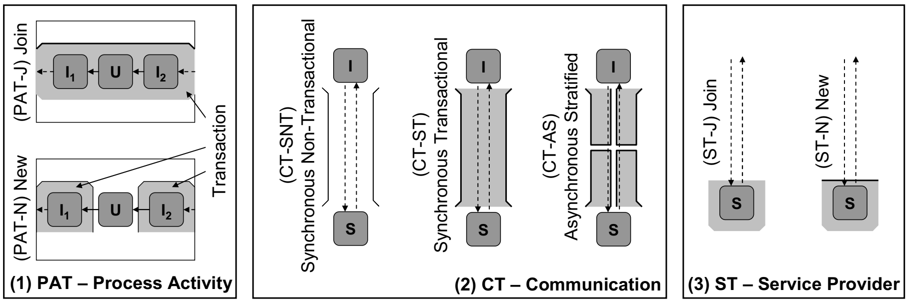

(PAT-J) Join
(PAT-N) New
Transaction
(1) PAT – Process Activity
(CT-SNT) Synchronous Non-Transactional
(CT-ST) Synchronous Transactional
(CT-AS) Asynchronous Stratified
(2) CT – Communication
(ST-J) Join
(ST-N) New
(3) ST – Service Provider

Fig. 4. Conceptual primitives as pattern building blocks (notation same as in Figure 3)

To elaborate upon the defining characteristics of the patterns and the primitives, we now present them in a format commonly used in the design patterns literature.

Intent. All patterns and primitives share the objectives motivated in Sections 2 and 3: To protect enterprise resources against integrity and correctness threats that may occur during concurrent process execution, e.g., when multiple processes and activities in the SCL invoke distributed services via a SOA communication infrastructure.

Pattern 1. Decoupled TRANSACTION ISLANDS (PAT-J+CT-SNT+ST-N in Figure 3)

Problem. How to isolate SCL process activities from service operation execution?

Solution. Do not propagate the transaction context from the SCL to the service.

Example. In the CRM case study, this pattern is applicable for Determine Wireless Customer Status. This analysis-level task is realized as a process activity that invokes a read-only operation which in this example should execute non-transactionally.

Forces and consequences. If a service operation fails, the process navigation in the SCL is not affected, and vice versa. If a service works with shared enterprise resources, its operations must be idempotent, as they may be executed more than once due to the transactional process navigation in the SCL. In many cases, the service provider must offer compensation operations, and higher-level coordination of the compensation activities is required (e.g., via business transactions; various models have been proposed). In practice, this pattern is often selected as a default choice.

Pattern 2. Tightly coupled TRANSACTION BRIDGE (PAT-J+CT-ST+ST-J shown in Figure 3); MULTIPLE BRIDGES variant (PAT-N+CT-ST+ST-J).

Problem. How to couple process activity execution in the SCL and service operation execution from a system transaction management perspective?

Solution. Configure process activities, communications infrastructure, and service providers in such a way that the SCL transaction context is propagated to the service.

Example. In the CRM case study, this pattern addresses the all-or-nothing requirements stated for Retrieve/Upgrade Wireless Customer Profile (co-located services).

Forces and consequences. Process activities and the service operations invoked by them execute in the same transaction. As a result, several service operations can also participate in the same transaction. Therefore, a natural limit for their response times exists ("tens of seconds to seconds at most" [8]). If a service-internal processing error occurs, previous transactional work, which can include process navigation in the SCL and the invocation of other services, has to be rolled back. This pattern meets resource protection needs well on the system level, but often is not applicable, e.g., when processes and operations run for days or months. Hence, a common variation of this pattern is to split an SCL process up into several *atomic spheres* [8], creating MULTIPLE BRIDGES for selected process activity/service operation pairs. Executing the process activities in a small number of transactions (single TRANSACTION BRIDGE) reduces the computational overhead for process navigation; splitting the process up into several atomic spheres (MULTIPLE BRIDGES) increases data currency.

Pattern 3. Loosely coupled STRATIFIED STILTS (PAT-J+CT-AS+ST-J in Figure 3)

Problem. How to realize asynchronous, queued transaction processing in SOA?

Solution. Use message queuing as SOA communication infrastructure.

Example. In the CRM case study, this pattern must be applied for Reconcile Profile Upgrade, as the wireline CRM systems only provide messaging interfaces (e.g., JMS); additional compensation logic is required. In Figure 3, I_1 and S_1 use *stratified transactions* (as defined in [8]) during service invocation; on the contrary, service S_2 reads the request message and sends the response message within a single transaction.

Forces and consequences. Services do not have to respond immediately; the delivery of the messages is guaranteed by the communications infrastructure. If the execution of the service operation fails, the process may not get an immediate response; additional error handling is required, often involving compensation logic. This pattern often is the only choice in process-enabled SOA, e.g., when integrating legacy systems.

PAT primitives. As Figure 4 shows, Process Activity Transactionality (PAT) defines two primitives for the SCL, transaction context sharing or *Join (J),* and transaction context separation or *New (N).* If set to PAT-J, a process activity executes in the same transaction context as the adjacent activities in the same process; it *joins* an existing context. As a consequence, the process activity's work might be rolled back if any other process activity or service operation that participates in the same transaction fails. With *PAT-N*, a process activity is executed in a *new* transaction context. Both PAT-J and PAT-N are valid choices in all three composite patterns; PAT-J is shown in Figure 3 and commonly used in practice. In TRANSACTION BRIDGE, PAT-N models the MULTIPLE BRIDGES variant. Deciding for PAT-N is justified if two process activities should be independent from each other from a business requirement point of view. Furthermore, some process models contain loops that are too complex to fit into a single, short-lived system transaction (e.g., due to retries, refinement/completion cycles, and service provider limitations).

CT primitives. We model three Communication Transactionality (CT) primitives, *Synchronous Non-Transactional* (CT-SNT), *Synchronous Transactional* (CT-ST), and *Asynchronous Stratified* (CT-AS). CT-SNT is used in the TRANSACTION ISLANDS pattern. It represents a synchronous service invocation from the process activity without propagation of the transaction context. As a consequence, the activity waits until the call to the service returns. Once the service has been called, there is no possibility to influence the work the service conducts. For example, the CT-SNT service invocation may cause the transaction to exceed the maximum duration configured in the SCL, which may result in a transaction timeout and a subsequent rollback. With CT-SNT, undoing the work of the service can not be included in this rollback.

CT-ST is required to build a TRANSACTION BRIDGE. It models a synchronous service invocation with transactional context propagation. As a consequence, the process activity waits until the call to the service returns; a rollback may occur after the service execution has completed (the service participates in the SCL coordination).

CT-AS is part of the STRATIFIED STILTS pattern. It represents an asynchronous service invocation without transaction context propagation. In CT-AS, long-running services can be invoked without loosing transactional behavior, as the process navigation is part of a *stratified transaction* [8]. At least three transactions are involved in the invocation of a long-running service: the request message is sent in a first transaction; in a second transaction, the message is received by the service provider and the response message is sent; in a third transaction, the process activity receives the response from the service. As shown in Figure 3, depending on the service implementation, the second transaction (provider side) may be split up into two transactions: receive the message and commit, and later on, send the response in a new transaction. Such stratification details are described further in [8].

ST primitives. Two choices and corresponding primitives exist for the Service Provider Transactionality (ST): *join* an incoming transaction (ST-J) or create a *new* one (ST-N). ST-J is used in TRANSACTION BRIDGE, ST-N in TRANSACTION ISLANDS. In ST-J, the service provider participates in the transaction of the caller (if a transaction exists). As a consequence, process activity execution in the SCL and the invoked service operation influence each other, e.g., when causing a rollback. In ST-N, the service provider does not participate in the incoming transaction. As a consequence, if the transaction in which the process activity runs is rolled back and the activity is retried later (e.g., due to process engine-specific error handling procedures), the service may operate on enterprise resources that have been modified in the meantime.

4.2 Sample Mapping of Primitives to BPEL/SCA Technology and Engine

As step (c) of our method, we now map the three PAT, CT, and ST primitives to BPEL and SCA and other technology platforms. We expect that BPEL engines provide settings that allow configuring the transactional behavior at least for *invoke activities*. Services are invoked via protocols such as SOAP/HTTP, IIOP and JMS, which differ in their support for transaction context propagation and (a)synchrony. The transactional behavior of SCA components is defined by SCA *qualifiers*.

Qualifiers specify the behavior desired from the point of view of the service consumer (SCA reference and SCA import) and the service provider (SCA interface, SCA implementation).

(1) The PAT primitive from Figure 4 does not have a direct BPEL realization; typically, BPEL engine vendors add proprietary support for it. Furthermore, additional standardization work is underway; for example, the BPEL for Java (BPEL4J) specification introduces *ACID scopes* [2]. The exact semantics are BPEL engine-specific. For example, during a rollback an engine may let the entire process fail, request resolution by a human operator, or retry one or more activities at a later point in time (potentially with a different transactional scope). While this is engine-specific behavior outside of the scope of the BPEL specification, the process modeler must be aware of it when selecting between PAT-J and PAT-N. (2) CT-SNT as a synchronous invocation not propagating the transactional context maps to SOAP/HTTP or IIOP as transport protocol. CT-ST maps to SOAP/HTTP with WS-AtomicTransaction support or to IIOP. CT-AS can be implemented with JMS; however, no standardized WSDL bindings exist at present. CT also determines the SCA qualifiers on reference, import, and interface level, e.g., `SuspendTx` and `JoinTx`. (3) ST can be mapped to the SCA qualifier `Transaction` on component implementation level.

Table 1 maps the three conceptual patterns from Section 4.1 to CT and ST primitives and corresponding SCA qualifiers. At the time of writing, these qualifiers resided in a non-standard namespace [7], not yet in one of the emerging SCA standards [9]. The full mapping reference can be provided.

Table 1. Mapping of conceptual patterns to primitives and SCA qualifiers

Primitive Qualifiers / **Patterns**	**CT** **SCA reference** (BPEL process as component invoking others)	**CT** **SCA import** (reference to external service)	**CT** **SCA interface** (service provider component)	**ST** **SCA implementation** (service provider component)
TRANSACTION ISLANDS	CT-SNT `DeliverAsyncAt=n/a` `SuspendTx=true`	CT-SNT `JoinTx =false`	CT-SNT `JoinTx =false`	ST-N (or ST-J) `Transaction =local|global|any`
TRANSACTION BRIDGE	CT-ST `DeliverAsyncAt=n/a` `SuspendTx=false`	CT-ST `JoinTx =true`	CT-ST `JoinTx =true`	ST-J `Transaction =global`
STRATIFIED STILTS	CT-AS `DeliverAsyncAt =commit` `SuspendTx=false`	CT-AS `JoinTx =n/a`	CT-AS `JoinTx =n/a`	ST-J `Transaction =global`

Mapping to IBM WebSphere Process Server (WPS). WPS [7] provides a BPEL engine, which exposes processes and services as SCA components; in WPS, a BPEL-based SCL connects to the underlying architectural layers via SCA. The SCA qualifiers from Table 1 govern the transactional context propagation and behavior. Furthermore, PAT translates into a proprietary invoke activity configuration attribute called `transactionalBehavior` which can be set to `requiresOwn` (PAT-N) and `participates` (PAT-J). Two additional vendor-specific values exist, which we did not model as primitives, `commitBefore` and `commitAfter` [7]. We implemented this PAT mapping in a decision injection tool prototype. The tool reads the conceptual pattern selection decision in and configures the WPS process model accordingly.

5 Related Work

Transactional workflows and business-level compensation have been studied extensively. However, existing work primarily focuses on advancing transaction middleware, runtime protocol, and programming model design. Methodological and modeling aspects for engineering transactional workflows from business requirements to conceptual design to low-level implementation details, however, are covered only insufficiently. SOA-specific challenges such as logical layering (e.g., SCL) and loose coupling are not addressed in detail. Reusable decision models or pattern catalogs do not exist.

Papazoglou and Kratz [10] propose a design approach for business transactions based on standard business functions such as payment and delivery in supply chains. Our approaches are complementary as they focus on different design decision points.

Witthawaskul and Johnson [16] use unit-of-work modeling to express transactional primitives in a Model-Driven Architecture (MDA) context; they provide sample transformers to Hibernate and J2EE (but not SOA). Our PAT and ST primitives are inspired by their platform-independent transactionAttribute (UnitOfWork stereotype).

The WS-BPEL specification [14] defines operational semantics for executable business processes, touching upon well-known transactional behavior without going into details. For instance, it provides the concept of isolated scopes in order to support exclusive access to particular resources. However, the BPEL specification does not define which coordination protocols and service component models should be used in order to comply with the specification; this is left to BPEL engine implementations.

SOA patterns have begun to emerge over recent years. For example, Zdun and Dustar define a pattern language for process-driven SOA [18]. In enterprise application architecture literature, we find a service layer pattern and general coverage of transaction management issues, but no coverage of workflow applications [3]. Hohpe and Woolf introduce a PROCESS MANAGER mainly concerned with message routing; their TRANSACTIONAL CLIENT allows sharing a transaction context over a message queue, but does not cover forces and consequences in process-enabled SOA [5]. The Patterns for e-business initiative [6] provides top-down design guidance, but does not cover transaction management details of the EXPOSED PROCESS MANAGER. There are workflow patterns [13], transactional workflow patterns [1], and service integration patterns [11], which focus on control flow and interaction structure, but do not address system transaction or business compensation design. These patterns also do not cover SOA implementation technology details such as WSDL transport bindings or BPEL and SCA deployment settings. Even if the existing patterns do not cover transaction management design aspects in detail, our decision and pattern-centric method leverages the pattern vocabulary and given design advice as background information.

6 Summary and Outlook

In this paper, we introduced a new analysis and design method leveraging architectural decision models and patterns in support of the full lifecycle of designing transactional workflows, a particularly challenging problem in the construction of process-enabled SOA. We motivated the need for such an approach by (a) identifying recurring, reusable architectural decisions. We then (b) defined three conceptual patterns, TRANSACTION ISLANDS, TRANSACTION BRIDGE, and STRATIFIED STILTS, consisting of platform-independent primitives modeling system transactionality on (1) process activity, (2) communications infrastructure, and (3) service provider level. We (c) defined and implemented a mapping from the conceptual primitives to known technical uses in BPEL and SCA and one particular BPEL/SCA engine. Such a full-lifecycle analysis and design method allows sharing conceptual architectural knowledge across technology and platform boundaries, but also takes platform-specific aspects into account. This is required because legacy systems limitations constrain the decision making in practice, for example the transaction boundaries of existing software assets and commercial packages implementing parts of the business process.

Future work includes documenting more variations and pattern selection guidance for our three patterns. The three primitives can be mapped to more runtime platforms such as the Spring framework. To extend the method, architectural patterns for other recurring decisions, for example business-level compensation, should be documented. Finally, we plan to investigate whether our design-time patterns can be represented as runtime policies in emerging SOA runtimes, for example future versions of SCA.

References

[1] Bhiri, S., Gaaloul, K., Perrin, O., Godart, C.: Overview of Transactional Patterns: Combining Workflow Flexibility and Transactional Reliability for Composite Web Services. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, Springer, Heidelberg (2005)
[2] BPELJ: BPEL for Java, ftp://www.software.ibm.com/software/developer/library/ws-bpelj.pdf
[3] Fowler, M.: Patterns of Enterprise Application Architecture. Addison Wesley, Reading (2003)
[4] Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufman Publishers, San Francisco (1993)
[5] Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Addison Wesley, Reading (2004)
[6] IBM Patterns for e-business: Exposed Serial Process application pattern, http://www.ibm.com/developerworks/patterns/b2bi/at8-runtime.html#soa
[7] IBM WebSphere Business Modeler: Integration Developer, Process Server, http://www.ibm.com/developerworks/websphere/zones/businessintegration
[8] Leymann, F., Roller, D.: Production Workflow. Prentice Hall, Upper Saddle River (2000)
[9] Open Service Oriented Architecture, http://www.osoa.org/display/Main/Home
[10] Papazoglou, M., Kratz, B.: A Business-aware Web Services Transaction Model. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, Springer, Heidelberg (2006)
[11] Service Integration Patterns, http://sky.fit.qut.edu.au/~dumas/ServiceInteractionPatterns

[12] SOAP 1.1, http://www.w3.org/TR/2000/NOTE-SOAP-20000508
[13] v.d. Aalst, W.M.P., ter Hofstede, A.: Workflow Patterns, www.workflowpatterns.com
[14] Web Services Business Process Execution Language (BPEL), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[15] Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315
[16] Witthawaskul, W., Johnson, R.: Transaction Support Using Unit of Work Modeling in the Context of MDA. In: Proc. of EDOC 2005, IEEE Press, Los Alamitos (2005)
[17] WS-AtomicTransaction: WS-Business Activity Framework, WS-Coordination, http://www.ibm.com/developerworks/library/specification/ws-tx
[18] Zdun, U., Dustdar, S.: Model-Driven and Pattern-Based Integration of Process-Driven SOA Models, http://drops.dagstuhl.de/opus/volltexte/2006/820
[19] Zimmermann, O., Doubrovski, V., Grundler, J., Hogg, K.: Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario. In: OOPSLA 2005 Conference Companion, ACM Press, New York (2005)
[20] Zimmermann, O., Gschwind, T., Küster, J., Leymann, F., Schuster, N.: Reusable Architectural Decision Models for Enterprise Application Development. In: Overhage, S., Szyperski, C. (eds.) Proc. of QoSA 2007. LNCS, Springer, Heidelberg (2007)