Flexible Business Transaction Composition in Service-Oriented Environments

Benedikt Kratz\textsuperscript{1} Ting Wang\textsuperscript{2}, Jochem Vonk\textsuperscript{2}, and Paul Grefen\textsuperscript{2}

B.Kratz@uvt.nl \{t.wang, j.vonk, p.w.p.j.grefen\}@tm.tue.nl

\textsuperscript{1}Tilburg University, Infolab
the Netherlands

\textsuperscript{2}Eindhoven University of Technology
Department of Technology Management
the Netherlands

\textsuperscript{*}The research reported in this paper has been conducted as part of the eXecution of Transactional Contracted Electronic Services (XTC) project (No. 612.063.305) funded by the Dutch Organization for Scientific Research (NWO).
Abstract

Reliability and clear semantics in the presence of errors are of key importance for automated execution of business processes. Consequently, transaction support is a major ingredient for business process automation in a Service-Oriented Computing (SOC) environment. As automated business processes become more complex, different parts of business processes have different transaction requirements. This implies that no single transaction model can accommodate for all possible transactional requirements. Therefore, we propose the Business Transaction Framework (BTF), catering for flexible composition of a variety of transaction models. Composition is supported in both a hierarchical and a choreographed way and is governed by composition rules, including inherent transaction constraints and application domain specific rules. Transaction models are represented by Abstract Transactional Constructs, which can be parameterized to tune the desired transactional behavior. Multiple composed transactions can be associated to a business process type, such that precise transaction semantics can be selected per process instance. The BTF is supported by a three-level architecture, laying the basis for flexible, transactional intra- and inter-organizational business process execution on a web service based platform.

Keywords: Business Transactional Framework; Advanced Transaction Models; Transaction Management; Inter-Organisational Transactions, Web Service Transactions
## Contents

1 Introduction ................................................. 4

2 Background and General Approach .......................... 6

3 Business Transaction Framework Conceptual Model .... 8
   3.1 Abstract Transactional Constructs ................... 8
      3.1.1 The ATC Concept ................................ 8
      3.1.2 The ATC Hierarchy ............................... 9
   3.2 Abstract Composite Business Transactions .......... 12
   3.3 Composite Business Transactions .................... 13

4 Business Transaction Framework Architecture .......... 15
   4.1 Architecture ........................................... 15
   4.2 Architecture Qualities ............................... 17

5 Example ..................................................... 20

6 Related Work ............................................... 23

7 Conclusion ................................................... 25

References ..................................................... 26
1 Introduction

Business processes are inherently complex and therefore impose multiple requirements on transaction support to allow a reliable execution. Until now, solutions to solve this problem have been geared towards the use of one or a limited combination of transaction models, see e.g., [GMS87, Elm92, GPS99, CCF+04a, CCF+04b].

However, in intra- and inter-organizational business transactions, various transaction models and methods of their integration are required by possibly different participants to allow a seamless integration of internal and cross-organizational systems in business processes. The Business Transaction Framework (BTF) that we present in this paper fills the void left by the previously mentioned solutions by offering a comprehensive conceptual and architectural model which allows the composition of various established transaction models into an overall business transaction in a sound way.

Transaction models are represented by Abstract Transactional Constructs (ATCs). Together the ATCs form a hierarchy with three main transactional classes. The ATC hierarchy resembles the inheritance/abstraction mechanism in object oriented approaches, as for example provided by the Swing package in the Java programming language [HC05], in which components can be plugged together to offer the desired functionality (as a GUI in the Swing case). Providing the desired transactional semantics for a business process requires a suitable combination of ATCs from the ATC hierarchy, in which certain parts of the process are transactionally covered by one ATC (or more than one ATC in the case where ATCs are nested). ATCs can be combined in a choreographed and/or hierarchical way. A specific combination of ATCs is called an Abstract Composite Business Transaction (ACBT). During the creation of an ACBT, the run-time behavior can be further specified by parameterizing the involved ATCs. The creation of an ACBT itself is governed by composition rules, which, e.g., restrict the way ATCs can be combined. Through the creation of multiple ACBTs that would fit a certain business process, the transactional behavior in the execution of that business process can be changed by choosing a different ACBT. This provides for a highly
flexible transaction support mechanism for business process in general, i.e., for intra- and/or inter-organizational business processes.

Our contribution to the field of Service-Oriented Computing (SOC) is twofold. First, this framework acts as an abstraction layer on top of service oriented computing concepts like context, coordination and transactions. Through this layer we abstract from intra- and inter-organizational aspects allowing our framework to cater for both aspects at the same time. Additionally this abstraction layer also integrates related concepts from fields like databases and workflows to allow highly flexible transaction support for business processes. With the framework it becomes possible to choose and to combine existing transaction models in a way most suitable (for the required transaction support) for applications. Our second contribution to the SOC field is the implementation of the BTF using Web Service technology.

The remainder of this paper is organized as follows. In Section 2 we will present the background and an overview of the BTF. Thereafter we will focus in Section 3 on the conceptual model and in Section 4 on the architectural model of the BTF. We will illustrate in Section 5 the applicability of our framework in an example. In Section 6 we will give an overview of related work and we will finish this paper in Section 7 with the conclusions.
2 Background and General Approach

A business process contains activities and tasks which are interconnected and possibly even nested. The different activities (and tasks) may impose various transactional requirements on the complete business process. Also the participants which perform the actual tasks can be quite heterogenous (e.g., traditional databases, web-services) and may require their own transaction models. In this context, the reliable execution of a business process requires many different transaction models.

However, current endeavors within the arena of transaction support for long running business transactions focus on the use of existing specifications in an ad hoc manner thereby mainly focusing on application level transactions. The aim of our research is to provide a transaction framework with a formal base which can leverage existing specifications and transaction models in this area as required. The BTF therefore contains support for a wide variety of existing transaction models.

As no single transaction model offers all required transaction semantics, combinations of transaction models are required. The need for a broad range of transaction models makes it necessary to compose an overall transaction plan for a business process, which integrates and connects different transaction models with each other in relation to business process steps in order to reliably execute the business process. It is therefore of key importance to understand the meaning of compositions of transaction models on a technology and business level.

To be able to create such a transaction plan we must first analyze existing transaction models with their properties and behavior and secondly identify which transaction models can be connected together and what the semantics of such a connection is. In this way transaction models can be related to each other in a sound way so that they become interoperable (i.e., which transaction models can be used next to or within each other) during runtime.

To accommodate the above requirements the BTF has three distinct phases, see Figure 1. The first two phases are design phases where busi-
Figure 1: Abstract overview of the BTF

Business transactions are specified using atomic building blocks of the BTF (the definition phase) and compositions thereof (the composition phase). The last phase is the execution phase and is concerned with the enactment of business transactions. The further details of the concepts used in these phases (i.e., the conceptual model) are elaborated in the next section.
3 Business Transaction Framework Conceptual Model

This section describes the concepts used in the framework and relates them to each other.

The concepts of our framework are distributed across the three phases illustrated in Figure 1. In the definition phase, the concept representing the atomic building blocks of the BTF, called Abstract Transactional Constructs (ATC), are covered. During the composition phase, ATCs are composed into compositions that are represented by the Abstract Composite Business Transactions (ACBT) concept. In the execution phase ACBTs are enacted. The concept related to an enacted ACBT is called a Composite Business Transactions (CBT).

The three distinct phases form a logical sequence in the sense that concepts in subsequent phases of this framework build upon the concepts of preceding phases.

In the following subsections we present the required concepts of each phase together with their specific properties and details, starting with the ATC concept of the definition phase.

3.1 Abstract Transactional Constructs

In this section we will first present the details of the ATC concept. Thereafter we will present a set of related ATCs grouped together into a hierarchy.

3.1.1 The ATC Concept.

An ATC is a construct that is an abstract representation of a transaction model with its specific properties and functionality comparable to a transaction model class in object orientation or to a component that represents a particular transaction model.

ATCs are (if allowed by the particular transaction model) composable horizontally, allowing to create a choreographed ordering of ATCs, and vertically, allowing to nest ATCs. The behavior of an ATC is parameterizable
through its interfaces. The static behavior of an ATC is parameterizable through the structural interface of an ATC. This parametrization of static behavior connects an ATC with other ATCs horizontally and/or vertically and allows the setting of requirements for and restrictions on these connections. The dynamic behavior of an ATC is parameterizable through its behavioral interface and allows to set the run-time behavior of a particular transaction.

An ATC contains two types of property-sets:

- Static properties (i.e., class variables) that are valid for all instances of a particular ATC and which are thus specific for a transaction model. These properties cover the static properties (e.g., the transaction delimiters) and the dynamic properties (i.e., the behavior) of a transaction model (e.g., the transaction protocol with its states and order of executions).

- Transaction specific properties (i.e., instance variables) that can be set using the parameters of the structural and behavior interfaces. For the former, properties that refer to other ATCs or that specify the use of particular transaction model property (e.g., use a Saga savepoint in this ATC) can be set. For the latter, the run-time properties for a particular transaction (e.g., the current state of a transaction) can be set.

### 3.1.2 The ATC Hierarchy.

With the above definition of an ATC we can now classify various existing transaction models and relate them into a tree structure using abstraction of common and specialization of specific properties of transaction models, introducing thereby a relationship between transaction models based on the abstraction relation. In this way we strive to achieve the above mentioned interoperability between and reusability of transaction models together with the definition of rules that guide the composition of ATCs.

An initial hierarchy is presented in Figure 2. The most abstract transaction model in our model is the empty transaction model called TxM (i.e.,
transaction model), which defines common properties of a transaction model like the name, structural couplings and transaction management events, see e.g. [CR90, CR92].

Figure 2: The ATC hierarchy

Below the most abstract model we can identify and classify three main groups of transaction models:

- The flat transaction model \((FTxM)\)
- The choreographed transaction model \((CTxM)\)
- The nested transaction model \((NTxM)\)

The flat transaction model is the oldest, most simple and most widely used transaction model. It adheres to the ACID properties and has extensions like the two phase commit protocol (2PC) in case of distributed flat transactions. Examples of the flat transaction model from SOC are for example WS Atomic Transactions \((WS AT)\) [CCF+04a] and WS-CAF’s ACID Transactions \((Tx ACID)\) [BCH+03].

The choreographed transaction models are used in environments that require long-lived transactions. In such cases, a choreographed transaction decomposes a long running transaction into small, (sequentially-executing) sub-transactions. This model can be further specialized for example into the
saga or extended saga transaction models. The extended saga transaction model can be decomposed further into workflow transaction models like the \emph{WIDE Global Transaction Support} (\emph{WGTxM}) \cite{VGBA99} and \emph{X-Transaction} (\emph{XTxM}) \cite{VG03}. Also the \emph{Long Running Action} (\emph{Tx LRA}) \cite{BCH++03} model from WS-CAF fits in this category.

The nested transaction models adopt a top-down method to decompose a complex transaction into child transactions according to the application semantics. Nested transactions overcome the shortcomings of flat transactions by permitting parts of a transaction to fail without necessarily aborting the entire transaction. This model can be further classified into specializations like the open nested and the closed nested transaction model, which can again be further specialized into, for example, the \emph{WIDE Local Transactions model} (\emph{WLTxM}) \cite{BGVA98}.

Examples of open-nested transaction models from SOC are for example \emph{WS Business Activity} (\emph{WS BA}) \cite{CCF++04b} and the \emph{Business Process Transaction Model} (\emph{Tx BA}) from WS-CAF \cite{BCH++03}.

All existing transaction models can be classified into these three main categories of transaction models. We see recent and new developments of transaction models in the realm of workflows, web services and grid-computing as special types of the three main classes of transaction models.

Adhering to the principles of OMG’s model driven architecture (MDA) \cite{MM03} approach, our framework also provides support for platform specific transaction model implementations, for example Oracle (\emph{ORA}) or Sybase (\emph{SB}). Figure 2 shows a clear distinction between platform specific and platform independent models. Platform specific models are leaves below the dotted line that are ready to use and have a system which implements the higher level transaction model. Consider as an example a business transaction that intends to use a flat transaction model without specifying the concrete platform specific implementation which should execute this transaction. The BTF can decide then at run-time the concrete platform specific model. This implies that each ATC has a mapping to a concrete platform specific transaction model.

A second distinction between the transaction models is indicated by the
difference between white and grey boxes. Models with grey boxes are specific enough to be executed (and which are thus usable at least in theory and which may or may not have an implementation) and white boxes represent ATCs that are only abstract and super-type like, i.e., \textit{non-executable}.

3.2 Abstract Composite Business Transactions

In this section we introduce the ACBT concept of our framework, which is used during the composition phase of the BTF.

In this phase several inputs are combined together to design an abstract blue print which reflects the transactional requirements of a business process. The inputs of this phase are:

- A business process specification,
- the collection of ATCs,
- the set of ATC composition rules and
- an optional contract in the case of inter-organizational business process, which possibly puts constraints on the above inputs.

The business process specification (e.g. in BPEL4WS [ACD+03]) is used as a starting point to identify the required set of transactions semantics, based on the activities and tasks of the process, which are required to execute the business process in a reliable way. As every transaction adheres to a particular transaction model, the result of the mapping from process to transactions is a transaction plan of that process consisting of the connected transaction models.

The available ATCs, i.e. the ATC hierarchy, serve as the building blocks in the composition of the transaction plan that suits the business process. The composition process is guided by composition rules\textsuperscript{1}, which specify the allowed ATC compositions, and may have also additional requirements from

\textsuperscript{1}Whilst we acknowledge the importance of the composition rules, this is a part of future research mentioned in Section 7.
contracts and requires application specific knowledge about business processes. By parameterizing the structural part of the interface of the ATCs, an ACBT is created. Through the structural interface the horizontal, indicating a sequence (i.e., horizontal composition), and vertical, indicating nesting (i.e., vertical composition), connections between the transaction models can be set. This allows our framework to map complex business processes to transaction graphs.

An ACBT in fact is a structural specification that is comparable to a workflow specification. The ACBT of a business process can be seen as a blueprint of a transaction plan. A business process can have multiple ACBTs to facilitate different situations. Some situations may require very strict transactional requirements (e.g., high profile sales), whereas others can be handled with very few transactional semantics (e.g., low frequency and turnover sales).

Every ACBT can make use of other ACBTs, either coupled together in a sequential way or by nesting. This introduces great flexibility and allows reuse of existing ACBTs to adopt to new situations. Using ACBTs as reusable transaction patterns reduces ambiguity and opens possibilities for verification and analysis like reasoning about behavior at run-time. Also computer supported ACBT creation becomes possible because of the clear, strict and consistent semantics.

3.3 Composite Business Transactions

In the execution phase of the BTF, ACBTs will be enacted. CBTs are the concepts that represent enacted ACBTs and are described in this section.

At the start of a business process execution an ACBT that is appropriate for that business process execution, i.e., it provides the required transaction semantics, is instantiated. In the case that more than one ACBT would fit a business process, the most suitable one must be chosen. Instantiating an ACBT implies that every ATC in this ACBT is parameterized through its behavioral interface with behavioral information for this particular business process instance. This means that for every running business process instance
multiple ACBT instances can exist next to each other. The parameterized ATCs are real transaction model instances (i.e., concrete transactions) that are referred to as transactional constructs (TC) in our framework. TCs are coordinated and orchestrated and are the entities that actually group work items (i.e., tasks) into transactions. The complete set of interconnected TCs which emerges from an ACBT is called a composite business transaction (CBT) in the BTF.

There are two types of TCs in a CBT. The first type are TCs that are not further nested and that actually perform tasks or interact with (remote) transaction managers and their resources as a part of a transaction. The second type of TCs performs mainly coordination tasks. This keeps the control at the TC and allows a reaction to events and even dynamic adaptation of a CBT during run time. Events are initially bound to scopes and to specific TCs of the first type in a CBT, which can however escalate certain event types to higher levels. How events are handled can be parameterized at the creation of a CBT.

If a non-executable ATC, i.e., the white boxes in the ATC hierarchy as for example shown in Figure 2, is chosen in an ACBT, it means that any executable child of that ATC (in the ATC hierarchy) is applicable.
4 Business Transaction Framework Architecture

To support the BTF conceptualized in Section 3, a highly dynamic and flexible architecture is required. The design of the architecture must be in line with the requirements from the conceptual model and the phases of the BTF and should adhere to general architectural design principles and styles. In Section 4.1 we present the BTF architecture. In Section 4.2 we elaborate on the key qualities of our architecture and its underlying design principles.

4.1 Architecture

The BTF architecture, shown in Figure 3, contains seven functional components and four types of data components (artifacts) representing the concepts explained in Section 3. As can be seen in the figure, the components are distributed across three layers. These three layers are:

- Artifact Management Layer. This layer contains the functional components that manage artifacts.
- Artifact Creation Layer. In this layer the functional components that create the needed artifacts are located.
- BTF Management Layer. This layer contains an overall control component, that manages the other functional components, but also addresses issues related to interoperability with, e.g., BTF managers of other organizations.

Next to the distribution of the components across these layers, the components are also grouped into three different structures to meet the needs imposed by the three phases.

The definition phase structure contains the components related to the ATCs. The ATC Editor is used to create, delete and modify ATCs. The ATC Library Manager manages the access to the ATCs. On request of the ATC Editor or the ACBT Composer (see below) ATCs are retrieved from the
Components in the composition phase structure are involved with the handling of ACBTs. The \textit{ACBT Composer} is used to compose ACBTs from ATCs retrieved through the ATC Library Manager. The ACBT composer thereby adheres to the composition rules specified to connect ATCs together vertically and horizontally into different ACBTs in a sound way. It uses business process specifications as the input and the contracted \textit{Quality of Service} as the service agreement. The resulting ACBTs meet the required transactional semantics of the business processes. During the ACBT composition, the \textit{ACBT Library Manager} registers, persists and maintains the newly created ACBTs in a library to ensure their reusability. The ACBT
Composer and the ACBT Library Manager together are in charge of ACBT creation by parameterizing the ATCs. As an ACBT is reusable in similar business scenarios, the ACBT Library Manager picks up specific ACBTs for the real execution according to various preferences.

The execution phase is the phase where real composite transactions are executed. Components in the execution phase structure handle CBTs. The CBT Creator, when fed with a specific ACBT, forms a CBT, thereby instantiating the run-time properties of the ATCs and creating as a result connected TCs. For every CBT a CBT Manager is created (as multiple instances of business process can run simultaneously in parallel) to handle the execution of its corresponding CBT to control and monitor its corresponding running CBT with its TCs. When the overall CBT is finished (i.e., commits), the corresponding CBT Manager disappears.

The BTF Manager is the coordinator component for all other components in the architecture. It coordinates and controls the activities of other components and communicates with the underlying systems, e.g., DBMS, WfMS, through the IT infrastructure such as the Enterprise Service Bus [Cha04]. It can also work across the organizational boundary with BTF Managers from other organizations using open communication standards like SOAP or HTTP. In general, the BTF Manager serves as the hub of the BTF and interacts with all functional components.

4.2 Architecture Qualities

In [BCK98] an overview of three types of architecture qualities is presented:

- system qualities, which include runtime observable (e.g., performance, functionality or availability) and runtime invisible qualities (e.g., modifiability, portability or reusability),

- business qualities (represent concerns in the business environment where an architecture is developed, like time-to-market or cost) and

- architecture qualities (related to the architecture itself, e.g., completeness or correctness).
Based on the above qualities, the key qualities for the BTF architecture in the context of reliable automated execution of business processes are functionality, portability, reusability, correctness, and completeness. A heterogeneous architecture style like the layered style or the independent component style [BCK98] is used in the BTF architecture presented in Section 4.1 to achieve the desired BTF architecture qualities. Within the BTF architecture, modifiability, reusability and portability is achieved by decoupling various portions of the computations into components and assigning the components to different layers. The following issues are thereby addressed, not only making the BTF architecture a solid foundation for the BTF but also letting the architecture support the key features of the BTF:

1. Separation of concerns in the BTF architecture is established through the three layered architecture and the grouping of the components of the BTF architecture into dynamic structures, distributed across the three phases presented in Section 2. Through this layering and distribution, the BTF architecture supports contract-driven intra- and inter-organizational business processes as concerns of different requirements on security, functionality, etc can be separated.

2. The BTF architecture supports extensibility to accommodate changes in the environment of the BTF. Consider the creation of new ATCs and addition to the ATC Library when there are newly developed transactional models available. Also the architecture accommodates changes in business processes by allowing the creation of new ACBTs according to the new process specification.

3. Compatibility to current standards and support of the application of the BTF in different business scenarios is achieved through the use of SOC technology for the BTF architecture implementation. In this way the BTF architecture can utilize existing applications both internally and externally of an organization. Internal IT infrastructure like standards-conformable and/or commercialized DBMSs, WFMSs and Transaction Processing Systems are leveraged by the BTF for the local
sub-transactions. External applications required for inter-organization transactions are leveraged through Web Service technology, the Enterprise Service Bus (ESB) and remote BTF managers. Future research (see Section 7) will address the details of these issues.
5 Example

To illustrate the workings and benefits of the BTF, we present a small example in this section, which covers a simple travel arrangement business process illustrated in Figure 4.

![Figure 4: The travel arrangement business process](image)

The travel arrangement process contains two subsequent activities with subtasks, sales and book, and thereafter three further tasks. Within the sales activity, the select tasks (selecting a hotel, a flight and possibly a car) can be performed in parallel, after which the cost of the selected travel package is computed. The book activity is further decomposed into three parallel tasks that are all related to the booking of the previously selected travel options. After the book activity, the invoice is sent to the customer after which the payment is checked. Finally the documents can be sent to the customer.

We assume that a library of ATCs has already been created (using the ATC Editor and ATC Library Manager) containing e.g. the ATCs presented in Figure 2. Suppose the following requirements are considered relevant for the process:

- It is important that a hotel and a flight can be booked. Preferably, a car can also be booked, but only if the hotel and flight booking succeeds.

- The information concerning the selected trip (and costs) gathered in the sales activity must be stored, so that it can become a package to
be offered to other customers.

Looking at the business process and the transactional requirements as listed above, the following transaction models can be appropriate:

- Nested Transaction Model (NTxM). This would suit the ‘Sales’ and ’Book’ subprocesses. Selection and booking of the car is considered 'non-critical'.

- WIDE Global Transaction Support (WGTxM). This transaction model, based on the SAGA transaction model [GMS87], can be used to split the long-running process into smaller, shorter-running process steps, using compensation steps in case (part of) the process needs to roll-back.

- Flat Transaction Model (FTxM). If the last activity (‘Send Docs’) fails, it should be redone until it succeeds (forward recovery). Failing this activity should have no further impact on the rest of the already completed activities. Also, the ‘Invoice’ and ‘Payment’ activities can be supported by a flat transaction model if their compensation is taken care of by the (higher level) WGTxM.

![Figure 5: Example ACBT for the travel arrangement](image)

The resulting ACBT is shown in Figure 5, which represents the transaction plan for the travel arrangement business process based on the transaction requirements as mentioned above. The order between the ATCs is indirect, illustrated by dashed arrows, meaning that the order is determined by the process execution engine, which is then followed by the transaction execution
engine. Note that implementation wise, both engines might be combined into one and the same component, see also [Gre02]).

In other situations the process might require different transaction semantics. For example when it is known a-priori that the process will not take a long time to complete, the ACBT might consist of one (big) nested transaction or one flat transaction. Assuming a scenario in the SOC environment, the organization might be consider as two separate units (Front-end and Back-end) or organizations, each of which offers its process (parts) as one or more Web Services. The entire travel arrangement process might then be specified in a BPEL4WS process specification [ACD+03], attached with suitable transaction specification like WS-Tx [CCF+04a, CCF+04b], see also Figure 6.

Figure 6: Example ACBT for the travel arrangement process in a SOC environment
6 Related Work

Research into the BTF groups together work from various areas like databases, workflows, business processes and service oriented computing and is inspired by the work found in [Pap03] and the transactional work done in the WIDE and CrossFlow projects [GVA01, VG03]. In [Pap03] the need for a BTF has been motivated together with an overview of several technologies and protocols that may support the BTF.

Over the recent years many SOC transaction models have been proposed i.e., BTP [CDF+02], WS-Tx [CCF+04a, CCF+04b] and WS-CAF [BCH+03]. All these have different capabilities which may be beneficial in particular situations. For the integration of advanced transaction models, related work can be found in the various transaction meta-models. These models have a long history in databases. The ACTA framework [CR90, CR92] facilitates the formal specification, analysis and composition of advanced database transaction models. The ASSET system [BDG+94] uses primitives from the ACTA framework to allow the definition of application specific transaction semantics in the realm of databases and workflows. In [HLR+03], the CORBA Activity Service Framework that uses an event-signalling mechanism to coordinate activities according to a particular transaction model is evaluated against various advanced transaction models.

In [HW05], a meta-model for web service transaction models is proposed. Similar to our approach, their approach aims at flexible transaction support for web services using existing transaction models. However, their sole focus is on the modeling and representation of transaction models (selecting or creating one transaction model per application as opposed to the composition of different transaction models as we propose), whereas our approach also provides an execution framework with an detailed architecture and a clear coupling between business process and transactions.

In our opinion the above approaches are limited in their applicability because of their from database technology driven development without taking into account business and workflow requirements. The BTF we propose offers a more comprehensive framework which integrates both technical and
business requirements. The set of meta-models however offers interesting abstraction principles that we will use during the construction of the ATCs and the specification of composition rules.

In [TMR05], BPEL4WS [ACD+03], Ws-Tx and WS-Policy [BBC+04] are combined into a middleware based approach to support transactional workflows. They propose to combine multiple SOC transaction models as WS-C coordination types into BPEL4WS specifications. In relation to this approach, the BTF is not restricted to the SOC domain but offers a comprehensive and structured integration of various database, workflow and SOC transaction models. Additionally the BTF supports the specification of alternative ACBTs next to each other together with their context specific execution. The BTF abstracts thereby from particular technologies using high-level concepts with mappings to existing transaction models and technologies. E.g., it should be possible to map the approach in [TMR05] to our framework.

Other approaches in which combinations of multiple transaction models are used to support process execution exist in the area of inter- and/or intra-organizational workflow transactions. For example, in the WIDE project [GPS99] a combination of an extended saga model is combined with a nested transaction model. In contrast to the work presented in this paper, the combination of transaction models is fixed, while the BTF supports any (suitable and feasible) combination of transaction models.
7 Conclusion

The Business Transaction Framework, as presented in this paper, introduces concepts that allow composition of transaction models throughout a business process. Representing transaction models into Abstract Transactional Constructs and their composition into Abstract Composite Business Transactions allows the flexible creation of reusable transaction plans to support business processes. The execution behavior thereof can be parameterized into Composite Business Transactions at run-time.

The Business Transaction Framework provides a comprehensive and clear structured solution to ensure reliability and clear semantics of automatically executed business processes. Current approaches in the area of transaction support for business processes lack the ability to truly provide such a platform-independent way of integrating the required transaction models accommodating the various activities and their resources (databases, workflows, web services) throughout the business process into a cohesive whole.

The BTF is supported by a dynamic and flexible three-level architecture, that exhibits a grouping of functional components into three structures to accommodate the three phases of the BTF.

In the service oriented computing (SOC) environment, the BTF acts as an abstraction layer on top of service oriented computing concepts like context, coordination and transactions. This way it also possible to choose and integrate the most suitable transaction models for applications in the SOC environment.

Future work concerning the BTF will first of all focus on the formal ATC specification and the mapping of existing transaction models to this specification together with the formalization of the ATC composition rules. This allows the evaluation and comparison of the various existing transaction models themselves as well as the analysis and evaluation into the possible connections and nesting of the transaction models.

From a broader perspective, integration with existing standardization efforts from the business process arena (e.g., RosettaNet, EbXML) and e-contracting together with the incorporation of unconventional atomicity cri-
teria [Pap03] will take place. As the BTF is also applicable in the SOC environment (e.g., dealing with inter-organizational business processes), standardization eases the interoperability between the involved heterogeneous organizations.

Also, a prototype will be developed to serve as a proof of concept for the Business Transaction Framework.
References


[BBC+04] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Ashok Malhotra, Anthony Nadalin, Nataraj Nagaratnam, Mark Nottingham, Hemma Prafullchandra, Claus von Riegen, Jeffrey Schlimmer (Editor), Chris Sharp, and John Shewchuk. Web Services Policy Framework (WS-Policy), September 2004.


