Developing A Three-Dimensional Transaction Model for Supporting Atomicity Spheres

Willem-Jan van den Heuvel and Sergei Artyshchev
InfoLab, Tilburg University
PO Box 90153, 5000 LE Tilburg,
The Netherlands
Email: wjheuvel@uvt.nl and sergei@uvt.nl

Summary
Web Services (WSs) are touted as a new promising paradigm for developing and deploying distributed, Internet-based business applications. Recently, we have witnessed the emergence of web-service description and request languages as well as composition protocols. Transaction management is crucial for implementing reliable and robust web-based business transactions with cohesive web-services. However, existing service composition protocols tend to provide minimal support for transaction management. This support basically mimics traditional database mechanisms for ensuring ACID-properties of web-services, apparently in an arbitrary way. Ergo, sound theoretical underpinnings for incorporating transactional acidity in web-service composition languages is currently lacking.
This paper outlines a theoretical three-dimensional framework for ACID criteria of e-business transactions. This framework serves as the foundation of a transaction model that relies on "traditional" atomicity spheres.

1. Introduction

Web Services (WSs) are considered to be a contemporary paradigm for the development of distributed, Internet-based and platform-agnostic business applications [Yang2000]. Their prime appeal to the business community is in their capability to facilitate the interaction between complex heterogeneous federated systems using standard descriptions and middleware architectures. This property greatly facilitates their ability to engage into business transactions with other business partners that can last as short as a single business transaction or long as the product cycle; a critical requirement for progressive companies.
Last years, we have witnessed many efforts that were dedicated to standardize web-service interfaces for easy discovery and composition. From a conceptual perspective, web-service interfaces extend "traditional" object or component interfaces by extending the list of signatures that define their functionality with a list of non-functional properties for security, message exchange sequence, compatibility with standards, etc. Some of those standardization initiatives address web-service interfaces (e.g., WSDL, ebXML registry), while others emphasize the web-service composition (e.g., XLANG, wfXML, WSFL). In this paper, we focus at the latter category of web-service composition languages, and investigate ways to enrich them with some essential transactional properties.
Web-service flow, or choreography, languages serve to develop and enact business transactions by defining their execution flow. Business transactions can be defined as interactions between several business partners that are aimed towards achieving a predefined goal. Generally speaking, business transactions encompass the following three phases: the pre-transaction phase, the transaction phase, and lastly, the post-transaction phase. During the pre-transaction phase, prospective requesting parties identify and evaluate their requirements as well as alternative service-providers to fulfill them. The service-providers on the other hand offer their products and services to the prospective requesters by means of some standard repository, e.g. a UDDI registry. Next, the prospective service-requesters and –providers negotiate the mutual terms and conditions, and materialize them into contracts. At the end of this phase, the architecture for future transactions is established including the roles of the involved parties and the transaction context. The transaction phase encompasses the actual execution of the business transaction, possibly according to the agreements that were laid in a contract. Typically, the business transactions involve at least one paying/receiving and one goods/service-providing party. During the post-transaction phase, both the buyer and the seller check upon completion of the transaction by referencing to the contract. In addition, this phase can be used to fine-tune the performance/security of web-services as well as pre-initiation of new transactions.

Web-services can be composed and deployed dynamically, which means that a priori it is not know with which other web-services they will collaborate. In other words, they are should be both cohesive in the sense that they represent a coherent glob of business logic, and loosely coupled to satisfy

changing requirements on the fly. This implies that it is virtually impossible from a practical standpoint to statically define web-service execution paths, like in the case of "traditional" workflow systems. Rather, web-services are invoked asynchronously and independently, while generally relying on a peer-to-peer model for ensuring real-time sharing of services and removing the requirement of one fixed centralized component [Kalogeraki2001].
During execution they are considered as a holistic entity whose internal logic is to be either fully completed (committed) or aborted. To ensure this type of execution the requirement for a set of transaction properties to be adhered to is imposed. The original (database) transaction properties adhere to the generally accepted ACID criteria. Unfortunately, it is not always possible to enforce those properties in their classic (database) form due to the inherent characteristics of business transactions [tygard1989]. This paper primarily emphasizes extended atomicity properties for web-service choreographies, and, reflects on some of their implementation aspects.

This paper is organized as follows. In the following section, we will present an overview of state-of-the-art representational vehicles to represent business transactions as a set of collaborating web-services. Section 3 then discusses and organizes extensions to the traditional ACID properties that stem from the field of databases, so they adhere to the "higher-level" requirements of business transactions. Based on this framework, we subsequently develop a transactional model, encompassing two stratified spheres of control; again, by relying on and extension of proven transactional mechanisms from the field of databases and workflow systems. In section 5, these transactional models are exemplified by extending an existing web-service composition language, named WSFL. Section 6 concludes this paper with the conclusions, and enumerates some important issues future research.

2. Related work

Recently, several standardization initiatives are under development that stem from three perspectives, the web-service paradigm, e.g., UDDI (http://www.uddi.org), B2B standardization efforts, e.g., RosettaNet (http://www.rosettanet.org) and ebXML (http://www.ebxml.org), the business modeling domain, e.g., BPMI's BPML, and lastly, the workflow community (wfXML, http://www.wfmc.org). In particular, several flow-languages from both a web-service and a business perspective have been proposed that ensure to some extend the transactional characteristics for e-business interactions. In the following, we will briefly cover the most prevalent ones.
XLANG constitutes an orchestration language that extends WSDL, and has been implemented as a native for Microsoft's BizTalk server [BizTalk,2001], [Tatte2001]. XLANG encompasses a proprietary business process language that includes mechanisms for supporting long-running transactions. In particular, business processes dynamics are composed as a control flow that invokes the WSDL interfaces of interacting business partners. In addition, XLANG allows transactions to be composed using the standard construct of sequential and parallel execution. The notion of compensating transactions is addressed in this language by incorporating compensation blocks that are executed in case a fault occurs in an inner transaction.
Another important orchestration language initiative that is currently under development is IBM's Web Service Flow Language (WSFL) [Leymann2001]. Similar to XLANG, WSFL is designed on top of WSDL. However, its focus is broader in the sense that it not only targets external (visible) collaboration between business partners, but also formalizes internal processes. Service flow composition is implemented by allowing recursive specification of both the control and the data flow between (aggregated) web-services. Its support for long-running transactions is rather limited in comparison to the other proposals as of its full reliance on the external TP monitors (CICS) standard. IBM considers transactional properties of web-service assemblies for supporting business processes in the Web Service transactions (WSt) project [mikalsen2002]. It is expected that both the XLANG and the WSFL standard will converge into a joint World Wide Web Consortium (W3C) submission.
The third standardization initiative we wish to discuss here involves the Business Transaction Protocol (BTP, see [OASIS2002]) that is currently under development in OASIS, and evolved from the BEA BTP. The BTP implements a multi-level transaction model that is centered on a modified 2 phase-commit protocol. This protocol standard allows for combining atomic units of work into higher-order ("cohesive") web-service assemblies, while preserving the autonomy of participating organizations.
The Web Service Choreography Interface (WSCI) standard is an initiative from BEA Systems, Intalio, SAP, and Sun Microsystems. This standard models the external behavior of business processes in the context of an inter-organizational business transaction, and again constitutes an extension to WSDL. In particular, this protocol language is capable of describing the control flow between a web-service and interacting web-services by specifying them as a set of logical expressions. At the transactional level,

WSCI deals with atomic transactions, which treat a collection of activities as a single unit of work, as well as open nested transactions. By relaxing requirements for isolation, the nested open transactions give sufficient support for atomic and cohesive transactions. In addition, the open nested transactions encompass mechanisms for compensating both activities as well as sub-transactions.
The last web-service combination protocol we reflect on here is Business Process Modeling Language (BPML, [BPMI2001]) that has been proposed by the BPMI consortium. This standard aims at designing and publishing cross-organizational and evolving business processes, and is complementary to WSCI as it also can be applied to develop internal business process models. It includes a conceptual modeling language, which permits processes to be specified independently from any implementation language, middleware platform or programming model. It provides support for two transaction models, viz. the coordinated and the extended transaction model. The first transaction model is developed using two-phase commit protocol and guarantees both transaction atomicity and isolation. The extended transactions model extends the scope of the coordinated model by allowing nested and open transaction while relaxing the isolation property.

3. Extensions to the classical ACID Properties for e-Business Transactions

3.1 Differentiating Features of E-Business Transactions

As already mentioned, ACID properties originate from classic database transactions [Haeder1983], [Gray1993]. While considering business processes, several characteristics clearly differentiate it from classic model and impose additional requirements for their execution. These requirements include the following aspects:

- **Granularity (Cohesion)** – almost all of e-business transactions are complex (cohesive) in nature with multiple execution steps and distributed physical location of resources. This feature creates orchestration issues including conversation controller role definition, resource capacity planning, and lastly, data message format compatibility.
- **Coupling** – traditional transaction semantics are geared towards supporting fixed and short-term, integrated database transactions with a predictable end-state output. These semantics are hardly applicable for long-running and multi-dimensional business transactions that are loosely-coupled in the sense that there is a limited number of dependencies (both data and control) between them. It is generally believed that this will results in a considerable increase of both the flexibility and reusability (see below) of web-services as they make no assumptions on the implementation language of the participating business partners.
- **Duration/Longevity** – most of e-business transaction encompass long-running processes in nature. The exact duration of each sub-transaction sometimes can mostly not be predicted in advance. Consequently, web-service protocols should pay attention to the duration issues and business deadlines.
- **Reversibility** – unlike database transaction most of e-business transactions are inherently irreversible by simple rolling-back. This creates several supplementary functional requirements for web-service protocols, e.g. sophisticated compensation mechanisms including both compensation and contingency operators, rather then simple conventional undo/redo, roll forward and roll back capabilities.
- **Recoverability –** sub-transactions might produce results, which might be valuable due to the complexity of computation (or its duration) or because of reusability. Partial results shouldn't be lost in case of sub- or whole transaction failure. Logging, save points, and context security are among mechanisms to achieve that.
- **Reliability** – many of WS do not possess property of high availability. There should be mechanisms to guarantee that the execution time for each sub-transaction will be within expected timeframe and that for each sub-transaction compensating mechanisms exist.
- **Concurrency** – concurrency might be much higher in the e-business domain as compared to the traditional transactions due to the mentioned above characteristics. Consequently, the property of isolation is in most cases relaxed and the extended characteristic called *selective isolation* might be introduced. Selective isolation allows that *some* data elements are not locked during transaction execution.
- **Synchronization** – due to federated and volatile nature of e-business transaction execution it is virtually impossible to achieve synchronization across domains. Despite some protocols might

enforce response (confirmation) to every message sent it is rational to assume e-business transaction to be either totally asynchronous or deferred synchronous.

- **Reusability –** designates the ability to (re)combine web-services into cohesive transaction-enabling business applications. Atomic spheres are considered as an effective vehicle for demarcating of transactional boundaries within business process flows. By encapsulating fragments of business processes into cohesive, reusable and transactional constructs, it is expected that the degree of reusability can be enhanced.

3.2 Atomicity Requirements for E-Business Transactions

In recent work ([Papazoglou2002], [Schuldt2002], [Yang2000]), research has been conducted on the atomicity criteria of e-business transactions. In the following we will summarize some of the research results, structure them according to a simple atomicity taxonomy, and elaborate on them:

A. Infrastructure Atomicity Requirements

The infrastructure atomicity requirements pertain to those atomicity requirements which are needed to establish a reliable transaction environment. These requirements constitute elementary "building blocks" that are deployed during both the transactional and post-transactional phases that follow this initializing phase. We can broadly identify the following four levels of atomicity requirements for the pre-transactional phase:

1. Service Request Atomicity (A_{sr}): means that 1) A WS is published in an UDDI registry, and 2) by publishing the WS, it guarantees that to its best knowledge and effort it is capable or complete the transaction for the requester as an atomic unit of work. Service request atomicity is common for both e-marketplaces and supply chain transactions. Considering the B2B scenario we can further refine this property in the following two basic dimensions:

1.1. Exclusive Service Request Atomicity (A_{sre}) (`<Exclusive ={Yes/No}>`) - indicates that the requestor will exclusively utilize a requested WS. In practice, this property might inflict a substantial simplification of the transaction execution mechanism due to concurrency elimination. Consequently, a business transaction adheres to this atomicity property, if the message returned in response to a transaction initiation request contains exclusive property confirmation, otherwise the transaction will fail. From an implementation perspective, imposition of exclusive service atomicity might substantially simplify concurrency control because requestor and responder restrict their interaction only to their mutual transaction. It is assumed that *Exclusive* is set on the some criteria (for example, based on the access priorities or bids received), which might be specified as a WSDL service component.

1.2 Recursive Service Request Atomicity (A_{srr}) (`<Recursive = {Yes/No}>`) – upon completion of a web-service, it might automatically be restarted until it meets certain predefined criteria (for example, in case of re-initiation of parts request order due to the inventory depletion).

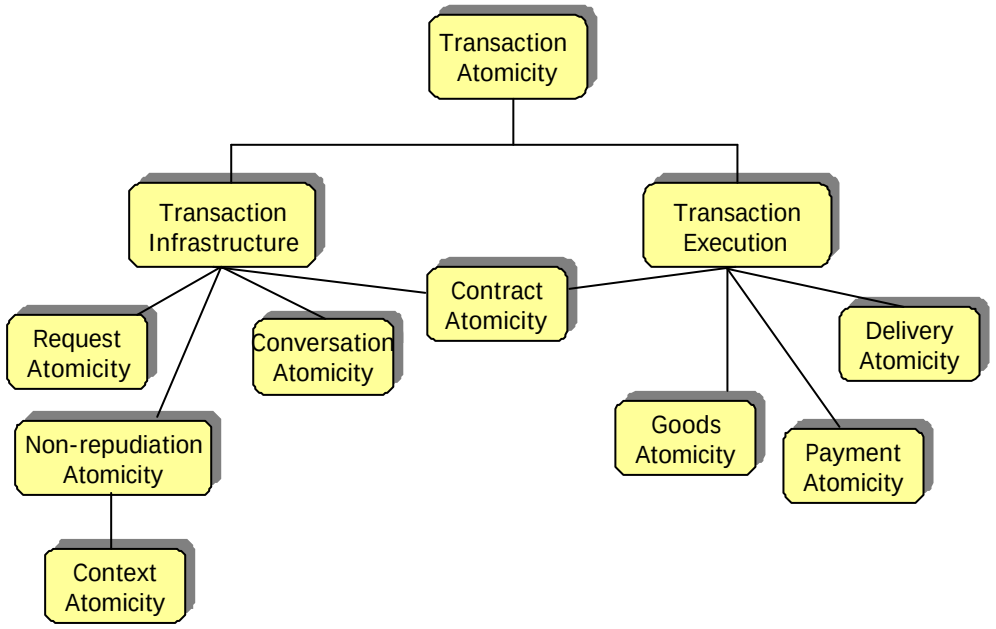


Figure 1 Taxonomy of Atomicity Criteria for Business Transactions

2. Conversation Atomicity (A_{cv}) (<ConversationController={Conversation controller name}>; <isInterruptablebyInitiator = {Yes/No}>; isInterruptablebyResponder = {Yes/No}>)-set of tags defining transaction properties towards contract execution. The *IsInterruptableby* tag explicitly specifies what party can interrupt document exchange. The conversation controller structures, monitors and controls the conversation between trading partners, and could in fact be one of the TP monitors of transacting WSs.

3. Non-Repudiation Atomicity (A_{nr}) (<isNonRepudiationRequired={Yes/No}>; <NonRepudiationMechanism={Non repudiation mechanism name/reference}> -specifies the non-repudiation provisions. Non-repudiation procedures might be specified in the contract, in this case the reference might contain contract reference or auxiliary non-repudiation mechanism (such as back-up or central authority system log). Mostly, this category of atomicity is guaranteed by an independent third-party.

4. Context Atomicity (A_{c})) (<ContextAtomic={Yes/No}> - this additional criteria refers to the run-time functional expression of non-repudiation atomicity. The context of a transaction is atomic if there are mechanisms implemented to ensure its correctness through the whole execution of the transaction.

B. Execution Atomicity Requirements

During the transaction phase, goods and/or services are transacted, possibly according to the agreements that are laid down in a contract that was negotiated during the pre-transaction phase. We distinguish between the following three fundamental levels of atomicity during this phase:

1. Payment Atomicity (A_{p}) (<PaymentProtocol = {payment protocol name/reference}>)**:** indicates the obligation for service requesters to compensate for the provided service. To ensure this level of atomicity the reliable/secure payment mechanism should be referenced for each transaction and utilized within its scope.

2. Goods Atomicity (A_{g}) (<DeliveryMechanism={Delivery mechanism name/reference}**>):** Goods atomicity is, despite its somewhat confusing name, concerned with services and goods. Goods atomicity requires that goods (or services) are non-anonymously (identifiable) and can be delivered as an outcome of expected transaction behavior, possibly as laid down in a contract. Thus, this category of atomicity typically involves payment-, delivery- and contract atomicity.
Furthermore, goods can be tangible or intangible. In case of tangible goods the property of goods atomicity might be relaxed, because it is not always possible (in case of dispute) to retrieve exactly the same goods which were subject to transaction (undo the delivery). In this case the emphasis should be on the certified delivery atomicity property.

3. Certified Delivery Atomicity (A_{cd}) (isCertifiedDeliveryRequired={Yes/No}; <CertifiedDeliveryProtocol = {Certified Delivery Protocol Name / Reference}>)**:** this category of atomicity is concerned with proving the fact that the goods were delivered according to the mutual agreements as stated in the contract. This level of atomicity could be achieved if the appropriate delivery document can be provided to all involved parties. Depending on the goods type (tangible/intangible) and delivery mechanism implemented, the certified delivery tag references the transaction content repository (in case of intangible goods, e.g., product data) or escrow service (trusted third party). Certified delivery atomicity might be relaxed in some cases of non-tangible goods or low cost of transaction/goods.

C. Contract Atomicity (<ContractReference = {Contract}>)**:**
This category of atomicity entails a special category that serves as a "bridge" between the infrastructure and execution atomicity criteria. Contracts are used to make explicit the mutual commitments between business partners and in which future commitments to transact goods, services, payments, etc., are laid down. In the context of transaction protocols, contract composition finalizes the infrastructure (transaction context) inception, and serves as the basis for the actual transaction execution. During the transaction termination, the contract is applied to terminate the transaction context. Several contemporary protocols include contract-like agreements for governing document flows during e-business transactions, e.g., the contracts in the Business Transaction Protocol [BPMI2002].

Consequently, business transactions will encompass the contract atomicity property if they include statements that express all legally binding mutual agreements between the transacting parties. Typically, contracts will govern payment and delivery processes in business transactions. In these cases, contract atomicity thus requires payment, goods and delivery atomicity.
The referenced contract could be: a legal document stating responsibilities or each party – referenced as address of document location or pointer to its text; or, a BPSS or other metadata, defining responsibilities of each party, business processes and agreements. However, here we do not wish to elaborate on vehicles to represent contracts in the context of e-business transactions. In [weigand2002], we have worked out a formal specification language for contracts that is integrated with a business-component based programming model. Contract atomicity is obligatory for contract-governed e-business transactions. In case of e-marketplace scenario this property might be modified because of many marketplaces tend to be governed by implicit contracts.

In short, the atomicity criteria for e-business transactions might be semi-formally represented with the following equation:

$$\mathbf{A} = (\mathbf{A}_{\mathbf{infrastructure}} \text{ È } \mathbf{A}_{\mathbf{execution}}),$$

Where $\mathbf{A}_{\mathbf{infrastructure}} = \{ A_{sr}, A_{sre}, A_{srr}, A_{nr}, A_{cv}, A_{nr}, A_{ct} \}$
$\mathbf{A}_{\mathbf{execution}} = \{ A_{p}, A_{g}, A_{cd} \}$

In line with these categories of atomicity, we will introduce a transaction model that comprises both infrastructure spheres and execution spheres (see section 5).

4. A Three-Dimensional Framework of Business Transaction Atomicity Criteria

Before we present a transactional model based on the notion of atomic spheres, we will introduce a three-dimensional model of the atomicity requirements that business transactions exhibit. This framework actually serves as the "meta-model" on which the transactional model (see next section) is grounded.
As indicated in the introduction, business transactions can be organized in three phases: pre-transaction phase, transaction phase and post-transaction phase.

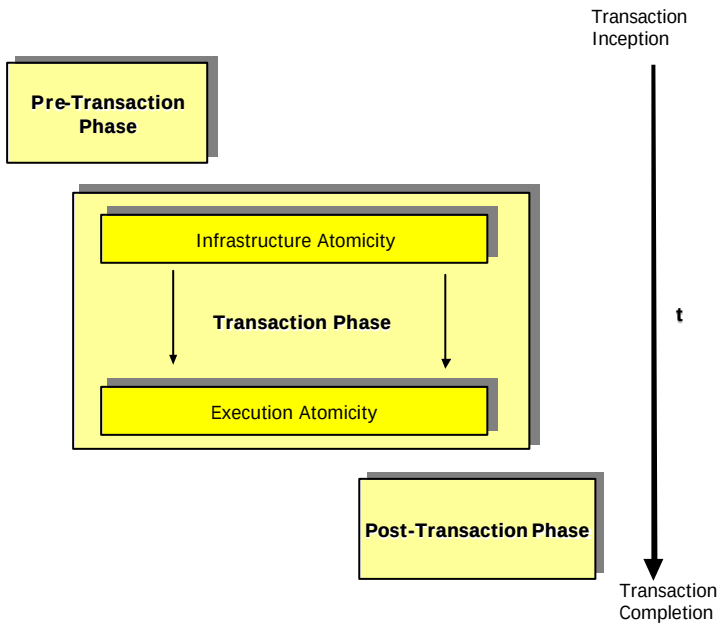


Figure 2 Atomicity Criteria Implementation within Transaction Dynamics

Not all atomicity criteria that were presented in Section-3, are meaningful during all phases of business transactions. This in fact depends on the *transaction state* of a business transaction. Figure-2 illustrates three prototypical phases of business transactions, and projects the two categories of atomicity criteria (infrastructure- and execution atomicity) in the transaction phase.
Both the pre- and the post-transaction phase are hard to formalize into (standard) business protocols as they involve many transaction context dependent activities, e.g., the searching/matching process for special product suppliers is hard to standardize. Hence, we argue that during these phases it is not meaningful to require that they comply with ACID criteria.
The transaction phase on the other hand typically involves usage of standard business protocols, possibly according to a (implicit) business contract. It is meaningful to perceive the atomicity criteria that are applicable during the transaction phase from two interrelated perspectives: the structural and the dynamic (or behavioral) perspective.
The structural perspective encompasses the stable transaction (sub)phase independent foundation of atomicity criteria. The atomicity criteria that make up this perspective serve as a prerequisite for combining them into "higher-order" atomicity criteria during the actual execution of the business transaction.
The behavioral perspective of atomicity criteria is reflected in the transaction execution flow for a business transaction. This view is built on top of the structural perspective, and specifies minimal atomicity criteria to which the transacting partners have to adhere.
Both views can be thought of as they mirror the infrastructure (static perspective) and the execution (dynamic perspective) atomicity criteria that were depicted in Figure-1.

Orthogonal on the static and dynamic perspective on business transaction, we can distinguish between declarative and functional definition of business transactions. The declarative definition of business transactions describes the main properties of them, without addressing any implementation (functional or procedural) elements. In fact, the declarative definition can be considered as a specification of meta-data that provides additional, non-functional information about the actual business transaction. The declarative definition perceives a business transaction as a holistic, black box entity without being concerned with its internal workings. The functional perspective on the other hand, is concerned with the actual implementation of the transactions, and describes their control flow, mostly as an execution

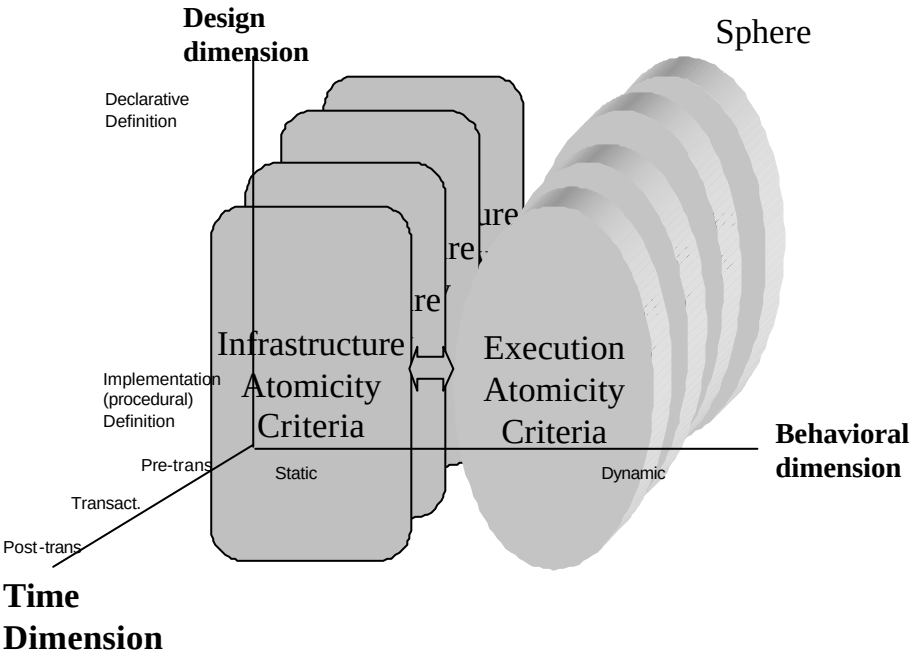


Figure 3 Fusing the design, behavioural and time dimension

flow. Thus the functional definition actually breaks up the black box, and transforms it into a white-box definition.

As demonstrated in Figure-3, the infrastructure atomicity criteria and the associated execution atomicity criteria, can be instantiated each time again for a new business transaction. We say that the

actual execution of the business transaction according to some atomicity criteria, actually dynamically creates an "*implementation spike*" whose boundaries dynamically expand/truncate at transaction time.

5. Spheres of Atomicity Model

After having defined the atomicity taxonomy and three-dimensional model for transactions, we can how define **infrastructure** and **execution** spheres (see Figure-3). Similar to ACID properties, the notion of (atomic) spheres stems from the domain of traditional database transactions [Davies1978]. Atomic spheres basically constitute a collection of actions that can be represented at various levels of granularity, but always looks from an external perspective. Spheres are atomic in the sense that a sphere (including all sub-spheres) either totally commits or aborts. This mechanism is particularly useful for enacting global transactions that encompass various related sub-transactions. Here, we tried to extend this mechanism to the level of business transactions, comprising in fact a set of interactions.

Spheres of atomicity have the following two elementary characteristics: behavior and persistence. Sphere behavior basically evolves around two patterns: expansion and truncation. Expansion occurs while goods or services are transacted between several parties – for every additional party involved the atomicity property is expanded to influence interaction between them. Truncation takes place when the interaction between certain parties is concluded (sub-transaction committed or aborted) so further interaction will not be performed. Persistence of atomic spheres is required to record the process dependencies of all steps during the business transaction as well as to indicate the set of atomicity criteria that are applicable to it.

In accordance with the taxonomy for atomicity criteria that was presented in Section-3, we can now define two categories of transaction spheres:

Infrastructure sphere: includes service request, conversation, and non-repudiation atomicity. Those type of atomicity are sufficient to initiate the transaction properly, and establish the contract between parties or identify that further interaction wouldn't be performed for some reason.
Execution sphere: encloses transaction execution (goods, services and money involved). Contract, payment, goods, and certified delivery atomicity should be preserved in order to make sure that the payment is done, goods are exactly as initially declared, and goods or services were actually delivered.

6. Extensions to WSFL for support Extended ACID Properties

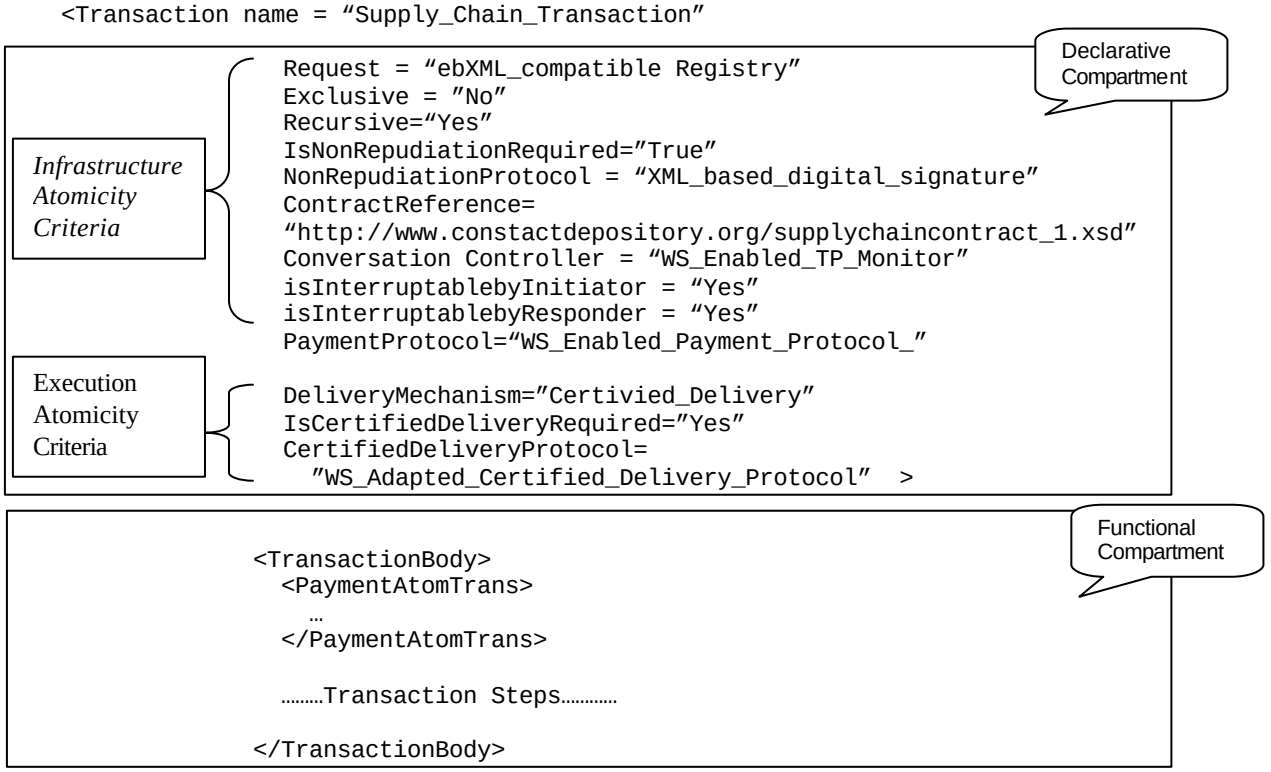
Now we have defined the basics of the transaction model for atomic infrastructure and execution spheres, we will introduce some simple extension to orchestration languages. Here, we have chosen to extend the WSFL language, but in principle, any other language with the same features could have been used here (see Section-2).

6.1. Extending WSFL

Transaction properties can be described using a declarative or a functional approach. The declarative approach results in the declaration of the *desired* transaction properties by explicit reference. In fact, the declarative approach defines *meta-data* about the web-service orchestration. The meta-data can be used to represent a broad range of issues, e.g., regarding the atomicity criteria. In addition, the meta-data can be applied to demarcate the transactional (atomic) fragments of a business transaction (<Transaction>...</Transaction>).
The declaration approach can be formalized with a special compartment in a WSFL definition that consists of a set of meta-data tags (e.g., using XML tags). Each tag represents a criterion for either infrastructure or execution atomicity, e.g. non-repudiation atomicity and payment atomicity. Transactional demarcations on their turn, can be made explicit by using transaction start and end tags (<Transaction>…</Transaction>).

The following simplified excerpt demonstrates this extension to WSFL :

```
<Transaction name = "Supply_Chain_Transaction"
```




```
</Transaction>
```

The second compartment of the WSFL excerpt formalizes the functional approach for describing web-service orchestrations. In particular, the functional approach defines the sequence of tasks that has to be executed, as well as external resources and exception handling to achieve the desired (declared) outcome. This is basically achieved by deploying an extended version of the internal flow language, that in fact is a lightweight version of a workflow definition language.
Due to space limitations, we will not elaborate the transaction body, e.g., by defining an XML schema or DTD for the extended version of the WSFL, here. It includes classical constructs for parallelism and concurrent execution, e.g. forks and joins. The transaction body extends WSFL by introducing operators for payment/goods/certified_delivery/contract transactions (e.g., <PaymentAtomTrans>…. </PaymentAtomTrans>), compensation, redo/undo, roll back and roll forward. The compensation construct (<Compensation>...</Compensation>) defines the compensation for the execution step of the transaction. In some cases it might be possible to restart the step disregarding the results it produced before the failure occurs. In this case only *Restart* should be located between tags. Roll forward (<RollForward> …</RollForward>) denominates a step necessary to perform in order to continue execution despite failure. For example, it might be possible to try booking another flight, or even a train instead of airplane. In this case the call for alternative stem must be placed between tags. The redo operator (<Redo>...</Redo>) can be used to indicate that a certain operation should be retried until a condition is met, or until time-out. The undo construct (<Undo>...</Undo>) basically constitutes the execution of one semantically opposite operators, e.g., insert/delete.

6.2 Atomicity Spheres and WSFL

As argued in the above, atomicity spheres are considered to be a powerful vehicle for ensuring the atomicity of long-lived and nested business transactions. Business transactions can be described with the proposed extended version of WSDL. In fact, we would like to argue here that this extended version conforms to a transaction model while including two main spheres, the execution and infrastructure sphere, and can be modeled along the axis of three orthogonal perspectives. The execution sphere is embedded in the infrastructure sphere that takes care of all low-level atomicity requirements. It encompasses several other transaction execution level spheres of atomicity. This transaction model is depicted in Figure-4.

In particular, Figure-4 illustrates one global execution atomicity sphere that encapsulates six tasks that are collectively aimed at the delivery of products (let's say semi-conductors) to a service requester (e.g., Philips-Conductors). Please not that the actual actors are not included in this figure. The control flow is depicted with directed arrows. Basically, the control flow is structured as follows. Firstly, the top lane in the figure is directed towards the production and delivery of goods, and takes the perspective of the service provider. The lower swimming lane in the same figure, consists of one single activity "Payment", and is modeled from the perspective of the service requester.

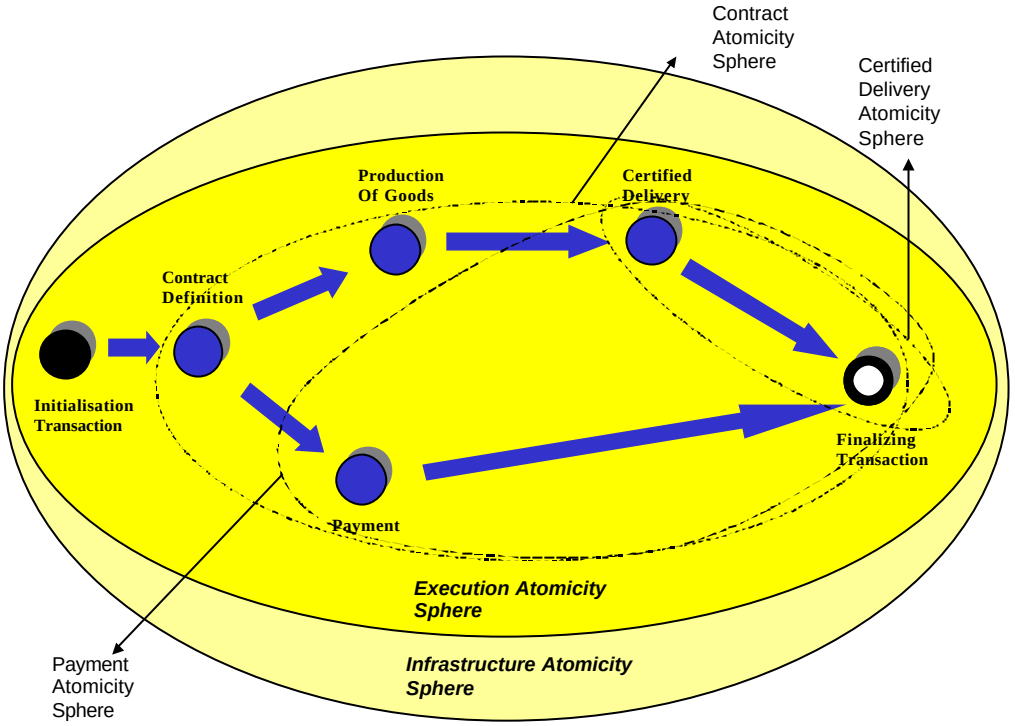


Figure 4 Transaction Model based on Atomicity Spheres

These tasks are captured in several atomicity spheres. To ensure readability of the figure, it only shows a few potential atomicity spheres, viz., the payment atomicity sphere, the certified delivery atomicity sphere and the contract atomicity sphere. The figure depicts that several tasks are actually grouped into cohesive atomic spheres, and that their surface actually demarcates the transactional boundary. All tasks within a sphere are either totally executed or skipped. One could argue that the payment atomicity sphere can actually not be considered to be a "traditional sphere" as it violates the structural condition that all control connectors that are entering the membrane of the sphere should originate from the same task [leymann2000]. However, as we did not want to violate the inherent symmetry of a business transaction, and the contract sphere could enforce deadlines, we decided to carefully allow multiple originators within this specific context.

Inspired by Leymann's definition [leymann2000] of atomic spheres, we now formally specify the relaxed notion of atomicity spheres in the context of web-services:

Definition 1 (Atomic Spheres)
Let G be an (a)cyclic web-service orchestration graph, and T the set of tasks that are connected with a set of connectors Conn. Generally speaking, and atomic sphere constitutes a subgraph, AS=G and AS?0, with the following three properties: (1) $\forall E \in AS$: $\psi(E)$ is a transactional web-service, $\psi(E)$: $x_{v \in \iota(E)} DOM(v) \rightarrow x_{v \in o(E)} DOM(v)$, (2) $\forall A \in AS$, $\forall B \notin AS$: $A \not\in B$ (as explained we have relaxed the condition regarding external predecessors), and (3) $\forall A \in AS \cap \lambda_t : \psi(A)$ committed or $\forall A \in AS \cap \lambda_t : \psi(A)$ aborted.

Likewise, atomicity spheres for e-business transactions can be defined. Here, we confine ourselves to the definition of payment atomicity:

Definition 2 (Payment Atomicity Sphere)
A payment atomicity sphere denotes a subtype of an atomic sphere, PAS = AS, that is embedded in at least one infrastructure sphere, AS_{infra}. A payment atomicity sphere can be represented as a tuple, PAS = (PAS_{decl}, PAS_{funct}), where $PAS_{decl} \in \{ AS_{infra}$ **(**A_{sr}, A_{sre}, A_{srr}, A_{nr}, A_{cv}, A_{nr}, A_{ct}) $\cup AS_{execution}(A_p)\}$. PAS_{funct} = (T,Conn,V,i,o,C,ε, Φ) with T is the set of tasks, Conn the set of connectors, V designates the set of data elements, i/o respectively the set of input and output containers, C refers to the set of conditions, ε on its turn represents the exit conditions, and lastly, Φ the assignment of join conditions.

7. Conclusions and Future Research

In this article, we have developed a three-dimensional transaction framework and model that support the notion of infrastructure and execution atomicity spheres. In order to implement the stratified transaction framework and model with the structural infrastructure sphere, and the dynamic execution sphere, we have operationalized these concepts in a contemporary web-service orchestration language, WSFL. Therefore, we suggest structuring a WSFL-like specification in two basic containers: a declarative/meta-data container and a functional/control-flow container. The declarative container essentially specifies meta-data about the control flow in terms of atomicity criteria. In the future, we would like to extend the meta-data to include more non-functional properties, starting at the remaining CID properties. The functional container on its turn basically represents the orchestration flow. It includes various operators to guarantee sphere atomicity including roll-forward and rollback mechanisms.

The results we have reported here are really core research results in nature. More research is needed in the following areas. Firstly, we intend to further formalize atomicity spheres, e.g. in terms of their content, relationship and behavior. In addition, we plan to investigate the applicability of the notion of compensation spheres in the context of e-business transactions. Compensation spheres allow both transactional and non-transactional web-services to co-exist; a realistic scenario in practice. Thirdly, more research is required into the possibility to guarantee the other CID properties. Lastly, the transactional model should be (partially) validated with an experimental prototype. This prototype should not only support service orchestration design, and its check upon acidity conformance, but also provide some run-time environments. We are designing and developing a prototypical implementation of execution spikes (e.g., payment and delivery spikes) that live within persistent infrastructure containers that monitor and enforce the low-level infrastructure atomicity criteria.

References

[BizTalk2001] Framework 2.0: Document and Message Specification, Technical Report. Microsoft Corporation, 2001

[BPMI2001] Assaf Arkin, "Business Process Modeling Language (BPML)", BPMI.org, Draft .04, 2001

[Davies1978] Davies, C.T., "Data Processing Spheres of Control", IBM Systems Journal, 17(2):179-198, 1978

[Gray1993] Gray, J. and Reuter, A., "Transaction Processing : Concepts and Techniques", Morgan Caufmann, San Francisco, 1993

[Haeder1983] Haeder , T. and Reuter, A., "Principoles of transaction-oriented database recovery", ACM Computing Surways 15(4):287-317, 1983

[Kalegeraki 2001] Kalogeraki, Vana; Pruyne, Jim; van Moorsel, Aad , “A Peer-to-Peer Architecture for Delivering E-Services”, HP Technical Report HPL-2001-181, 20010801, 2001

[Lewis2001] Lewis., P.M., Bernstein,A., Kifer, M., Databases and Transaction Processing : An Application Oriented Approach, Addison-Wesley, 2001

[Leymann2000] Leymann, F. and Roller, D., “Production Workflow: Concepts and Techniques”, Prentice-Hall, N.J., 2000

[Leymann2001] Leymann., F. (2001) WebServices Flow Language(WFSL 1.0)
IBM Software Group

[Mikalsen2002] Thomas Mikalsen, Stefan Tai, and Isabelle Rouvellou (IBM T.J. Watson Research Center, New York, USA), "Transactional Attitudes: Reliable Composition of Autonomous Web Services." In, Proceedings of the Workshop on Dependable Middleware-based Systems (WDMS 2002), part of the International Conference on Dependable Systems and Networks (DSN 2002), Washington D.C., June 2002

[OASIS2002] OASIS, Business Transaction Protocol, Version 1.0, 2002

[Papazolou2002] Papazoglou M.P. (2002) The world of e-business : web services, workflows, and business transactions. Notes from Keynote Speech, at CAISE2002, Toronto, May 2002.

[Schuldt2002] Heiko Schuldt , Gustavo Alonso , Catriel Beeri and Hans-Jörg Schek , “Atomicity and isolation for transactional processes”, In: Transactions on Database Systems, 27(1): 63-116, ACM Press, March 2002

[Tatte2001] Tatte, J. XLANG –web services for business process design
Microsoft Corporation, 2001

[Tygard1998] Tygar, J.D., Atomicity in Electronic Commerce, ACM-Mixed Media, 1998

[Yang2000] J. Yang and M.P. Papazoglou, “Interoperation Support for Electronic Business”, Communications of the ACM, Vol. 43, No. 6, pp. 39-47, June 2000

[Weigand2002] Weigand, H. and Heuvel, WJ, “Organisational Workflow Integration using Contracts”, DSS, 33:247-265, 2002

[WFMC2001] Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding Workflow Management Coalition, 2001