Grant Agreement N° 215483

Title:	*Business Transaction Language*
Authors:	*Tilburg, UCBL, Lero, UoC, USTUTT*
Editor:	*Rafiqul Haque (Tilburg)*
Reviewers:	*Manuel Carro (UPM)*
	Eoin Whelan (LERO)
Identifier:	*CD-JRA-2.1.3*
Type:	*Deliverable*
Version:	*1*
Date:	*16 December 2009*
Status:	*Final*
Class:	*External*

Management Summary

Application integration remains one of the core drivers of innovation in service engineering. Application integration serves as a means of developing service-enabled applications based on strategic technology capable of creating and successfully executing end-to-end business processes. The trend will be to move from relatively stable, organization-specific applications to integrated, dynamic, high-value ones where process interactions and trends are examined closely to understand more accurately application needs and dynamics. Such collaborative, complex end-to-end process interactions give rise to the concept of Service Networks (SNs) (see PO-JRA-2.1.1 & PO-JRA-2.1.2).

This deliverable targets the concept of a business transaction and explores how transactional processes and process fragments fit in the context of a running scenario which deals with end-to-end processes in a service network that possess transaction properties. Conventional (ACID) and unconventional (application-based) types of atomicity are introduced, including contract, payment and delivery atomicity, in the frame of a business transaction model. The transaction model provides a comprehensive set of concepts and several standard primitives and conventions that can be utilized to develop complex Service-Based Applications (SBAs) involving transactional process fragments.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Grant Agreement N° 215483

The main purpose of the deliverable is to present the design aspects of the initial version of the Business Transaction Language (BTL) that realizes the business transaction model and that is used to specify the elements of a business transaction. The initial BTL specification presented in this deliverable defines a transaction model and mechanisms for transactional interoperability between end-to-end service constellations in SNs and provides a means to define and enforce among other things transactional Quality of Services (QoSs) and SLA stipulations into SBAs. The objective of BTL is to provide the environment to build robust and successful mission-critical SBAs, using a fusion of concepts from application integration, transaction-based and business process management technologies.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 215483 (S-Cube).

Members of the S-Cube consortium:

University of Duisburg-Essen (Coordinator)	Germany
Tilburg University	Netherlands
City University London	U.K.
Consiglio Nazionale delle Ricerche	Italy
Center for Scientific and Technological Research	Italy
The French National Institute for Research in Computer Science and Control	France
Lero - The Irish Software Engineering Research Centre	Ireland
Politecnico di Milano	Italy
MTA SZTAKI – Computer and Automation Research Institute	Hungary
Vienna University of Technology	Austria
Université Claude Bernard Lyon	France
University of Crete	Greece
Universidad Politécnica de Madrid	Spain
University of Stuttgart	Germany
University of Hamburg	Germany
Vrije Universiteit Amsterdam	Netherlands

Published S-Cube documents

All public S-Cube deliverables are available from the S-Cube Web Portal at the following URL:
http://www.s-cube-network.eu/results/deliverables/

The S-Cube Deliverable Series

Vision and Objectives of S-Cube

The Software Services and Systems Network (S-Cube) will establish a unified, multidisciplinary, vibrant research community which will enable Europe to lead the software-services revolution, helping shape the software-service based Internet which is the backbone of our future interactive society.

By integrating diverse research communities, S-Cube intends to achieve world-wide scientific excellence in a field that is critical for European competitiveness. S-Cube will accomplish its aims by meeting the following objectives:

- Re-aligning, re-shaping and integrating research agendas of key European players from diverse research areas and by synthesizing and integrating diversified knowledge, thereby establishing a long-lasting foundation for steering research and for achieving innovation at the highest level.
- Inaugurating a Europe-wide common program of education and training for researchers and industry thereby creating a common culture that will have a profound impact on the future of the field.
- Establishing a pro-active mobility plan to enable cross-fertilisation and thereby fostering the integration of research communities and the establishment of a common software services research culture.
- Establishing trust relationships with industry via European Technology Platforms (specifically NESSI) to achieve a catalytic effect in shaping European research, strengthening industrial competitiveness and addressing main societal challenges.
- Defining a broader research vision and perspective that will shape the software-service based Internet of the future and will accelerate economic growth and improve the living conditions of European citizens.

S-Cube will produce an integrated research community of international reputation and acclaim that will help define the future shape of the field of software services which is of critical for European competitiveness. S-Cube will provide service engineering methodologies which facilitate the development, deployment and adjustment of sophisticated hybrid service-based systems that cannot be addressed with today's limited software engineering approaches. S-Cube will further introduce an advanced training program for researchers and practitioners. Finally, S-Cube intends to bring strategic added value to European industry by using industry best-practice models and by implementing research results into pilot business cases and prototype systems.

S-Cube materials are available from URL: http://www.s-cube-network.eu/

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
B2B	business-to-business
BPEL	Business Process Execution Language
BPEL4Chor	Business Process Execution Language for Choreography
BPM	Business Process Management
BPMN	Business Process Modelling Notation
BT	Business Transaction
BTL	Business Transaction Language
BTM	Business Transaction Management
CSCW	Computer Supported Collaborative Work
CSP	Communicating Sequential Processes
ebXML	Electronic Business using eXtensible Markup Language
EPC	Event Process Chain
FOLTL	First Order Linear Temporal Logic
KPI	Key Performance Indicator
LTL	Linear Temporal Logic
Open EDI	Open Electronic Data Interchange
PA	Process Algebra
QoS	Quality of Service
SBA	Service-Based Application
SLA	Service Level Agreement
SN	Service Network
SOC	Service Oriented Computing
StAC	Structured Activity Compensation
UMM	UN/CEFACT Modelling Methodology

UN/CEFACT	United Nations Centre for Trade facilitation and Electronic Business
WS	Web service
WS-Coordination	Web Services Coordination
WS-Policy	Web Services Policy
WS-Transaction	Web Services Transaction
XML	Extensible Markup Language

1 Introduction

Application integration remains one of the core drivers of innovation in service engineering. Application integration serves as a means of developing service-enabled applications based on strategic technology capable of creating and successfully executing end-to-end business processes. The trend will be to move from relatively stable, organization-specific applications to more dynamic, high-value ones where business process and SBA interactions and trends are examined closely to understand more accurately application needs and dynamics. Such collaborative, complex end-to-end service interactions give rise to the concept of SNs (see PO-JRA-2.1.1 & PO-JRA-2.1.2).

Currently, SBAs concentrate on composing software services into processes but do not explicitly correlate critical business activities and events, QoS requirements, and application (business) data, such as delivery dates, shipment deadlines and pricing, in one process with related activities, events, QoS and business data in other processes in an end-to-end process constellation. This implies that application management information and procedures are deeply buried in SBA code, which severely hinders maintenance and adaptation, both of which are essential for SNs. Such hardwiring means that any change or update to the application management logic already fabricated within an application requires programmatic changes to the SBA itself. This renders the potential reuse, customization, and monitoring of application management capabilities impossible. This also introduces intrinsic discontinuities between end-to-end business processes as information flows may be disrupted. For instance, a possible decoupling of payment information in payment and invoicing business processes from the ordering and delivery information of goods and services in order management and shipment business processes increases risks, could violate data integrity and contractual agreements, and may introduce discrepancies between the various information sources, which underlie these processes. Fixing this problem requires expensive and time consuming manual reconciliation. The principal activities required to sustain SBAs that collectively enact end-to-end processes, as outlined by [Yang2000], include the "collection, management, analysis, and interpretation of the various business activities and data to make more intelligent and effective transaction-related decisions".

SBAs that support end-to-end processes in SNs typically involve well-defined standard processes such as payment processing, shipping and tracking, determining new product offerings, granting/extending credit, managing market risk and so on. These standard processes apply to a variety of application scenarios. Although such standard processes or process fragments may drive transactional applications between SN partners, they are completely external to current Web services transaction mechanisms and are only expressed as part of application logic. Indeed, there is a need for explicitly managing fine grained tenets of SBAs such as business data, correlated events, operations, process fragments, local and aggregated QoSs and associated Key Performance Indicators (KPIs), and so on, to guarantee a continuous and cohesive information flow, correlation of end-to-end process properties, termination, and accuracy of interacting business processes that is driven by application control (integration) logic.

The above considerations give rise to a multi-modal transaction processing scheme by enabling reliable business transactions that span from front-end SBAs to back-end system-level transaction support and beyond to organizations that are part of a SN. Thus, the need for application-level management technologies that can ensure highly reliable application (not only system)-level transactions and end-user performance via rapid problem diagnosis and resolution - while at the same time support change and capacity planning - is paramount. In particular, we argue in favour of the need to provide rich features and QoS similar to that offered by transaction processing monitors but at the level of application management and application control logic.

Business Transaction Management (BTM) views everything from an application perspective. In the world of BTM, an application is considered as a collection of business transactions and events, each triggering actions on the application and corresponding on the infrastructure-level, which is handled

by transaction monitors using WS-standards such as WS-Transaction and WS-Coordination. The goal is to track every business transaction in an end-to-end process and correlate it to the information collected from the infrastructure so that solving problems and planning is done efficiently and holistically. It should be possible to have the ability to "stitch together" the individual business transaction data points into a map of the transaction topology and to monitor the metrics of each transaction for Service Level Agreement (SLA) compliance. Service analytics, e.g., simulation scenarios, and monitoring can then be applied to the transaction data to proactively manage services and accurately pinpoint problems. Such an end-to-end view enables to quickly isolate and troubleshoot the root cause of application bottlenecks, e.g., failure of an order due to the unavailability of just-in time production alternatives, potential performance problems, and tune proactively.

Effective Business Transaction Management can be achieved by introducing fine grained application management techniques applied to various tenets (granules) of SBAs ranging from business data, e.g., delivery times, quantities, prices, discounts, etc., to business processes of transactional nature, e.g., payment, delivery, etc. This facilitates potential reuse, customization, of application granules, expressed in terms of processes and process fragments, as well as monitoring of applications. The philosophy of this work package is that this information and integration logic should be carved out, isolated and made visible to facilitate the design of transactional processes or process fragments that could be combined into end-to-end processes in SBAs (see Figure 1).

Loosely speaking, a transactional process fragment is a transactional sub-process that is realized within an end-to-end process, while meeting granular process properties (a list of granular process properties is shown in Figure 1). Figure-1 shows that SBAs need to cross-correlate granular process properties that span across specific processes in an end-to-end process formation. Clearly, granular process properties go far beyond conventional application properties that are considered in traditional system transaction models, and include: operational level agreements (SLA/SLO), underpinning contracts, policies, rules and QoS thresholds for services at the application-level. Some of these process properties may be designated as being transactional in nature as they can be used to drive a service composition, e.g., end-to-end SLAs. In this way, granular process criteria can be used to drive and manage the composition of end-to-end processes at the application level. In particular, end-to-end processes exhibit transactional characteristics to deliver on undertakings or commitments that govern their formation and execution at the SBA-level. In other words, an entire end-to-end process or parts of it may fail if some transactional process properties, e.g., non-conformance to SLAs or aggregate mean-value KPIs, are violated.

End-to-end processes exhibit transactional characteristics that can be supported by an appropriate transaction infrastructure that employs Web services standards, such as WS–Transaction, WS-Coordination, WS-Policy (WS-Security), and BPEL (see Figure-1). Currently, this quartet of Web services standards is used to implement Web service transactions.

The transaction management infrastructure (see bottom layer in Figure 1) could for example be based on an open source implementation framework provided by JBoss Transactions (http://www.jboss.org) which supports the latest Web services transactions standards, providing all of the components necessary to build interoperable, reliable, multi-party, Web services-based applications. In such environments there is a clear need for advanced SBAs to coordinate multiple services and processes into a multi-step business transaction. The focus of this deliverable is to devise an initial version of the BTL and we will concentrate on the transaction management infrastructure in PO-JRA-2.1.4.

The suggested approach requires that several service operations or processes attain transactional process properties reflecting SBA semantics, which are to be treated as a single logical (atomic) unit of work that can be performed as part of a business transaction that can under certain conditions succeed or fail (alternatives can also be tried out). For example, consider a SN populated by a manufacturer and various suppliers; the manufacturer develops SBAs to automate the order and delivery business functions with its suppliers as part of a business transaction (named Provider 1-3 in

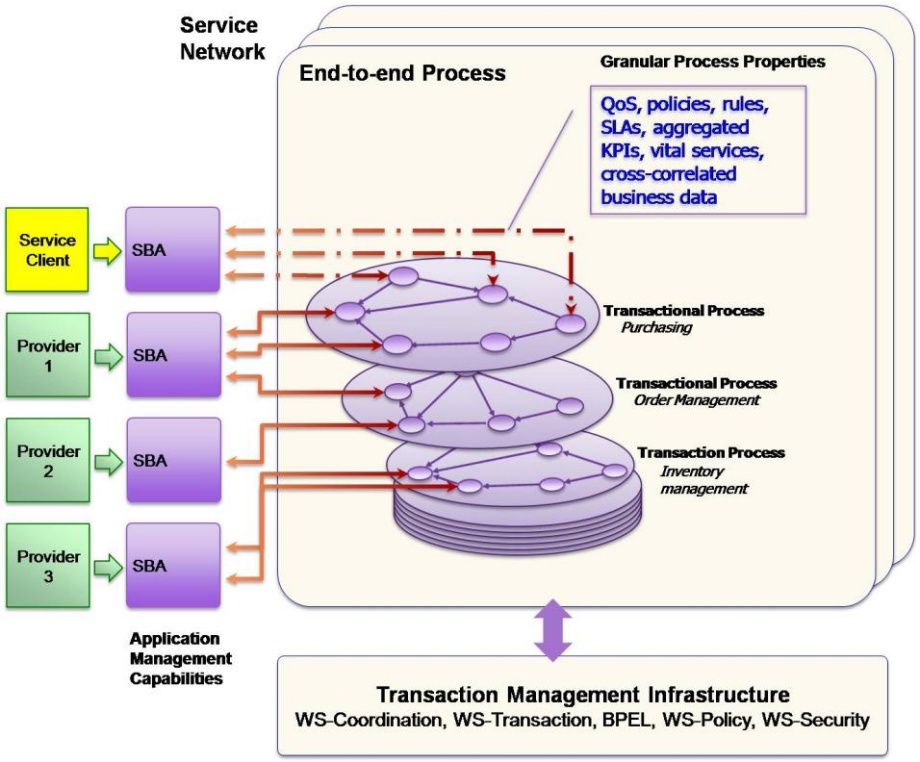

Figure 1 Service networks, end-to-end processes and transactional process fragments.

Figure 1). The transaction between the manufacturer (the client in the same figure) and its suppliers may only be considered as successful once all products are available in stock, delivered to their final destination, which could take days or even weeks after the placement of the order, and payment has ensued.

Some participating services in a SBA may be *vital* to a successful outcome of an end-to-end process. For example, a successful order for goods is a prerequisite, a strong requirement, for the entire end-to-end process to succeed followed by shipping and payment of goods. Given this vital precondition, there are several viable (successful and useful) outcomes. One *viable option* could be as shipping and insurance. Within each viable combination, there may be a further subdivision: *competitive (alternative) selection*. There might be several prices and types of shipping insurance available from which a selection must be made. These correspond to different instances of insurance services and insurance prices offered by diverse service providers.

The above rationale directs towards a business transaction driven service composition and eventual service selection. Such business transactions differ from conventional atomic (database-like) transactions by the ability to interact with a pool of potential composable services and participants (providers) at *run-time* and ponder different outcomes, before making a process-specific decision on a subset of those services and associated providers. Such process-specific decisions are based on granular process constructs (e.g., SLA mandates, cross-correlated business operations and data, policies, mean-time KPIs, and so on).

In a SN environment, transactions are complex involving multiple parties, spanning many organisations, and can have a long duration. More specifically, they are automated long-running propositions involving negotiations, commitments, contracts, shipping and logistics, tracking, varied payment instruments, and exception handling. Business transactions are very dynamic in nature. Parts of a business transaction may be retracted, alternatives may be tried out and optional activities may fail without influencing the transaction. Vital activities of a transaction behave as conventional short-lived ACID transactions and need to commit in order for the overall transaction to successfully

commit its results. In addition, data - for example, a customer account number or invoice - must be passed between the individual actions that make up the business transaction and some application control logic is required, to "glue" the actions together and make them succeed or fail (individually or collectively) depending on the requirements of the SBA. For example, there must be logic to deal with the conditional invocation of actions and with failures. Performance of all these tasks requires the infusion of advanced and unconventional transactional properties onto the services paradigm.

To achieve some of the above stated objectives business transactions rely on and extend transactional workflow technology [Sheth1993], [Alonso1996]. Like a general workflow, a transactional workflow consists of tasks that satisfy a set of coordination constraints. In particular, a transactional workflow involves coordinated execution activities which can have logical decision points that determine which branch of the flow of a work item may take at run-time in the event of alternative paths. Transactional workflow supports extensions to traditional database ACIDity to ensure workflow correctness and reliability, and to accommodate additional application functionality (e.g., to foster collaborative activities) and improve throughput (e.g., by minimizing transaction blocking) [Georg1995]. Unfortunately however, transactional workflows have typically resulted into highly application specific solutions in which application and coordination code are highly intertwined. In addition, transactional workflows are limited in that they preserve consistency and correctness of concurrent workflow executions by adhering to *application* correctness criteria such as temporal, serialization, visibility and cooperation dependencies, while largely neglecting granular process constructs that determine *process* correctness and consistency. Using the BT approach the logical decisional points in a workflow could be guided by either SLA or QoS. This means the execution paths in a transactional workflow are explicitly controlled and enforced by granular process specific criteria and standard transactional processes (payment, delivery, etc).

In this frame of mind, this deliverable sets out to define the concept of “business transaction” and requirements as well as the high-level design principles for a Business Transaction Language (defined as an extended XML vocabulary) whilst outlining the fundamental properties involved. Through integrative efforts, bringing knowledge from various disciplines including SOC, CSCW, BPM and Software Engineering, this deliverable defines how we should view a business transaction, which incorporates the process-level approach with the more conventional applications-level view and sketches the constructs for an initial transactional language. In particular, the main purpose of the deliverable is to present the design aspects of the initial version of the Business Transaction Language that realizes the business transaction model and that is used to specify the elements of a business transaction. The initial BTL specification presented in this deliverable defines a transaction model and mechanisms for transactional interoperability between end-to-end service constellations in SNs and provides a means to define and enforce among other things transactional Quality of Services (QoSs) and SLA stipulations into SBAs.

The deliverable is organised as follows. We first describe the essential characteristics of business transactions, leading to our own definition of the concept. Based on the essential characteristics of business transactions we then provide a list of requirements for defining a BTL.

Process fragments are an essential part of business transactions as they can comprise building blocks for creating and defining transactions. Thus, we then describe how the process fragments are related to a business transaction. Subsequently, we introduce a detailed scenario that serves for explaining the various concepts that we introduced in the earlier sections. The graphical representation of the scenario is provided using the Business Process Modelling Notation (BPMN). Following this we introduce a conceptual business transaction model, which describes the various essential business transaction properties and constructs. The business transaction model is then used as a basis to develop and illustrate some initial core BTL elements. Thereafter, we explore the formal underpinnings of the business transaction model to ascertain the internal validity of the BTL. Lastly, we summarize the key findings of this deliverable and outline directions for future work.

The relationship of this deliverable with work performed in other workpackages such as WP-JRA-2.2, WP-JRA-2.3, and WP-JRA-1.3 is highlighted in section 3 of this deliverable.

2 Essential Characteristics of Business Transactions

As organisations and technology continue to evolve, our understandings of the concept of a "business transaction" also undertake a multitude of changes, influenced from the classical schools of thought, which we argue now need modifications due to continuous technological developments. Therefore, it is critical that we formulate a deeper understanding to what a business transaction actually constitutes.

Today's organisational structure is heavily influenced by advanced SBAs, which execute well-defined business processes. [Papazoglou2006] explains that although service applications execute well-defined business functions, which may drive transactional applications, they are normally external to current Web service (WS) mechanisms.

The unprecedented growth in SBAs in a short period of time has emphasised the need to understand the mechanisms and underlying theorise related to the business transaction concept. Understanding the logic of business processes requires us to re-examine what is meant by a business transaction.

In this vein, the goal of this section of the deliverable is to help achieving an understanding of what is a business transaction, what do business transactions achieve, and how do they compare with conventional transaction procedures.

As research works and technological developments broaden the scope of transaction management, we will align transactional developments with our developments to improve functionality and performance within a SN environment.

2.1 Business Transaction Overview

Transactions are mainly associated with the business domain as they represent both tangible and intangible items (goods, money, information, and service). In recent years, the focus within computer science was on the automation of business transactions (i.e. process, execute and coordinate). In addition, the transaction model has undergone some significant changes through the introduction of business and information technological influences.

Nowadays, most business-to-business (B2B) collaborative applications require transactional support, while presenting many difficulties incorporating transactional properties such as Atomicity, Consistency, Isolation, and Durability (ACID). The ACID model is comprised of these four fundamental properties, which are considered the "building blocks" for transaction models. Although extremely reliable, classic ACID transactions are not suitable for loosely coupled environments such as WS based systems and transactions, which rely on long running processes. Strict atomicity and isolation is not appropriate to a loosely coupled world of autonomous trading partners, where security and inventory control may issue flexible atomicity and preventing hard locking of local databases. The major issue is the isolation of a database transaction. This property requires resource locking that is impractical in the services world. It would also preclude clients from participating in a business process, which is a strong requirement for SBAs. Sometimes, in a loosely coupled or long running activity, it may be desirable to cancel a work unit without affecting the remainder. In such cases, the strict atomic property of transactions needs to be flexible. This is very similar to the properties of nested transactions where the work performed within the scope of a nested transaction is provisional and sub-transaction failure does not affect the enclosing transaction.

The concept of the business transaction is heavily documented throughout various bodies of literature, where many authors share similar meanings and others argue its meaning within various contexts. The increase in organisations adopting a service-networked approach challenges our traditional understandings of the business transaction paradigm. In the following we summarise most of the important work and definitions for reasons of completeness.

The emergence of e-markets has created opportunities for organisations to combine capabilities and configure business transactions to integrate roles and relationships across partnering networks. The relationships of these networks play a fundamental role in the architecture of transactions. [Amit2001], capture the essence of the change which business transactions experienced in recent years due to "the unprecedented reach, connectivity, and low-cost information processing power, open entirely new possibilities for value creation through the structuring of transactions in novel ways".

[Kratz2004] also makes reference to the 'relationship' factor within a transaction. According to [Kratz2004], a business relationship is "any distributed state maintained by two or more parties, which is subject to some contractual constraints previously agreed to by those parties". [Kratz2004], then describes a business transaction as "a consistent change in the state of a business relationship. Each party in a business transaction holds its own application state corresponding to the business relationship with other parties in that transaction".

[Kambil1997], also adopts an operational view and states that all business transactions should present "significant information processing and communication to reduce uncertainties for buyers and sellers", i.e. quality, commitment, and protocols in plane for resolution over conflicts. Reducing uncertainties within a transaction heavily influences its outcomes.

[Hofman1994] simply defines a transaction as a "sequence of messages" which suggests that a transaction is triggered through the exchange of messages within a business management system (i.e. the initiator).

[Papazoglou2006] reports that a business transaction is defined as "a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties". This implies that there is an atomic or 'all or nothing' approach to meet defined objectives, and upon failure the transaction is rolled back.

According to [Clark2001], a business transaction is a "set of business information and business signal exchanges amongst two commercial partners that must occur in an agreed format, sequence and time period. If any of the agreements are violated then the transaction is terminated and all business information and business signal exchanges must be discarded". Thus, the document flow structure (time, format, and sequence) which exists between parties is important. [Kim2002], states that a business transaction consists of "one or two predefined business document flows and additional business signals". [Aissi2002] suggests that document flow is important and defines a business transaction as "an atomic unit of work between trading partners. Each business transaction has one requesting (incoming) document and an optional responding (outgoing) document".

However, the flow of these documents often only indicates the pattern in which a transaction relationship exists, for example [Kim2002] and [Aissi2002] do not propose that a transaction provides any business gain. [Papazoglou2006], includes the business value factors and states that business transactions are driven by "economic needs and their objective is accomplished only when the agreed upon conclusion among trading parties is reached, e.g., payment in exchange for goods or services".

One of the most important evolutionary factors of the traditional transaction model has been the transition from the single level transaction structure to the multi-level structures. Business processes

now interact across and between organisations to create a SN. Therefore, within a typical business transaction there must be at least two parties involved, i.e. a supplier who has a product or service to sell, and a customer who buys this product or service in exchange at a cost. A business model should therefore explicitly describe the collaborative interoperable business processes that are required to fulfil a business transaction.

The formation of a business transaction evolves to encapsulate a more networked and collective effort to reach predefined agreements, practices, procedures and outcomes. To add to this effort, [Papazoglou2003b] introduces a business transaction model encompassing business principles and models transactions with QoS characteristics, which highlights the need to describe the collaboration aspects' of business processes. Managing the complex transactions is extremely difficult, and these services are managed through the 'negotiation and enforcement of service level agreements' [Papazoglou2006].

As the definitions outline a number of key factors above, it may be useful to summarise these in Table-1.

Author	***Definition***	***Key Words***
[Hofman1994]	"sequence of messages"	Message, sequence
[Kambil1997]	"significant information processing and communication to reduce uncertainties for buyers and sellers"	Information processing, reduce uncertainties
[Clark2001]	"set of business information and business signal exchanges amongst two commercial partners that must occur in an agreed format, sequence and time period. If any of the agreements are violated then the transaction is terminated and all business information and business signal exchanges must be discarded"	Information and business signals, exchange, format, sequence, violation, termination
[Aissi2002]	"an atomic unit of work between trading partners. Each business transaction has one requesting (incoming) document and an optional responding (outgoing) document".	Atomic unit, request, respond
[Kim2002]	"one or two predefined business document flows and additional business signals"	Predefines flows
[Kratz2004]	"a consistent change in the state of a business relationship. Each party in a business transaction holds its own application state corresponding to the business relationship with other parties in that transaction"	Distributed relationship, change, transaction state
[Papazoglou2006]	"a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties"	Interaction, parties, accomplish, objectives, long periods, negotiation, conclusion

Table 1 Summary of business transaction definitions

From the above it is evident that the concept of business transactions has adopted several interpretations. It is therefore important to attempt to tie in these meanings to develop a more holistic vision of what constitutes a business transaction.

Within end-to-end processes in a SBA, complex information is exchanged for example, expected service, financial and contractual. [Medjahed2003] draw our attention to the concept of scalability of transaction-based systems which need to grow to support the relationships within these transactions, especially in the case where organisations are increasing the level of negotiating and interaction in

transactions with other organisations to provide some form of business solution. This places greater emphasis on the choreography of business transitions (specifies business states and transitions between business states). [Kim2002] explains that "the purpose of choreography is to order and sequence business transaction activity and/or collaboration activity within a binary collaboration, or across binary collaborations within a multiparty collaboration". In that respect, a business transaction describes the mission, behaviour, action, sequence, correlations of collaborative interactions with an objective of securing a business relationship to request or supply a product or service under predefined conditions.

We critically analyzed and assessed the previous business transaction definitions and draw a list of requirements based on a most recent works in the area of SN. In addition, we have reformulated the definition of a business transaction as follows:

"*A series of collaborative activities that explicitly enforces the achievement of an agreed-upon business objective in end-to-end processes. This objective is subject to service-level agreements that govern the choreographed/orchestrated behaviour, non-functional and timing requirements, correlated exchange of information, and control flow of composed services*".

A shared business objective extends over a possibly long period of time and is terminated successfully only upon recognition of the agreed conclusions, e.g., stipulated QoS, compliance to business and regulations, etc, between the interacting parties. A transaction usually outlines the liabilities of each party in the event where the intended actions are not carried out (e.g., promised services not rendered, services rendered but payment not issued). If a business transaction completes successfully then each participant will have made consistent state changes, which, in aggregate, reflect the desired outcome of the multi-party business interaction.

3 Requirements of a Business Transaction Language

Business Transaction Management (BTM) views everything from an application perspective. In the world of business transaction management, an application is considered as a collection of business transactions and events, each triggering actions on the application and corresponding on the infrastructure-level, which is handled by transaction monitors using WS-standards such as WS-Transaction and WS-Coordination. The goal is to track every business transaction in an end-to-end process and correlate to the information collected from the infrastructure so, solving problems and planning is done efficiently and holistically. It should be possible to have the ability to "stitch together" the individual business transaction data points into a map of the transaction topology and to monitor the metrics of each transaction for SLA compliance.

A BTL plays a pivotal role in BTM. The core requirement for a BTL is the ability to describe the granular transactional process properties of end-to-end processes, such as business commitments, mutual obligations and agreed upon KPIs, in a standard form that can be consumed by tools for business transaction implementation and monitoring. In this section, we will refine this all-encompassing BTL requirement.

Table 2 summarises the behavioural characteristics and key factors, which differ within a transaction and need to be expressed as part of a BTL. As summarised in Table 2, there are several characteristics and key factors, which distinguish certain stages within a business transaction. These stages, including transaction activities, business service dimensions and their implications for SNs, are captured in Table -3.

Characteristics	*Key Factors*
Generic	Who is involved
	What is being transacted
	Destination of payment and delivery
	Transaction time frame
	Permissible operations
Distinguishing	Links to other transactions
	Receipt and acknowledgment
Advanced	Ability to support reversibility (compensatible) and repaired (contingency) transactions
	Ability to reconcile transactions with other transactions
	Ability to specify contractual agreements, liabilities, and dispute solution policies

Table 2 Behavioural characteristics and key factors of transactions (Adapted from [Papazoglou2003b]).

Business Transaction	**Transaction Activities**	**Main Business Service Dimensions**		**Applications in Service networks**
Stage 1: Information Gathering	Source and product identification & potential supplier listed	Marketing Service	Information Service	Product information, response to inquiries
Stage 2: Negotiation	Evaluate supplier & analyze supplier quote		Communications Service	
Stage 3: Contract Fulfilment	Order, Payment, Delivery, Operations management	Logistics Service	Order and Payment Service	Order process, invoice, lead-time transportation, tracking shipment, advance ship notice, inventory transparency, response to inventory variety
			Delivery Service	
			Inventory Service	
		Operations Service	Product Service	Product quality, feature offering, cost, production scheduling, cycle time, capacity
			Production Service	
Stage 4: Collaboration	Collaboration planning, collaboration product development, co-location	Collaboration Service	Product Collaboration	Planning coordinator, geographical location, alternate delivery channel
			Information Collaboration	Updated information, joined planning and forecasting

Table 3 Business transaction stages and dimensions

As Table 3 (partly based on Open Electronic Data Interchange (Open EDI) [OpenEDI1995]) above outlines in order to deliver a business transaction, there are business processes, transaction activities, dimensions, and behaviour which need to be agreed upon, at various phases upon entering a

transaction agreement. The last column (far right) summarises the key tasks required at each stage within a transaction occurring in a SN.

While the information gathering and negotiation stages fall outside the scope of this deliverable (instead we refer to JRA-1.3 end-to-end quality provision and SLA conformance), we will concentrate in this deliverable on language requirements emerging from stages 3 and 4.

Based on an extensive literature survey and associated comparative analysis of existing business transaction models –notably Open EDI, the UN/CEFACT Modelling Methodology (UMM) [UMM2009] and electronic business using eXtensible Markup Language (ebXML) [ebXML2001] – in connection with the S-Cube reference architecture, we have distilled the following key transaction language requirements for stages 3 and 4:

Req 1: *Expressing collaborative activities that explicitly enforce the achievement of an agreed-upon business objective in end-to-end processes*: The BTL is pervasive in that it will be defined over end-to-end processes involving orchestrated and/or choreographed services that exhibit transactional properties.

Transactional properties may be expressed by combining existing transactional services or process fragments and associating them with application-level characteristics as well as contractual agreements in order to develop SBAs for SNs. Some of this work will be performed jointly with research activities in WP-JRA -2.2.

Req 2: *Expressing an on-demand delivery model for SBAs*: Business transactions are required to furnish an "on-demand" delivery model in which end-users may specify their preferences, e.g., desirable QoS, mandatory regulations, etc, as regards an end-to-end process. This implies that services are tentatively (re-) selected from a pool of service providers on the fly. Services and transactional process fragments can then be tailored, composed and then deployed over a variety of platforms. Some of this work will be performed jointly with research activities in WP-JRA -2.3.

Req 3: *Facilitating reusability and extensibility:* The BTL will impart constructs that define reusable and extensible transactional process fragments. Some of this work will be performed jointly with research activities in WP-JRA -2.2.

Req 4: *Expressing conventional atomic actions*: The transaction language needs to cater for conventional atomicity, as in some circumstances, service operations or transactional process fragments in an end-to-end process have to be strictly atomic. Assuming, for instance, that a client application decides to invoke one or more operations from a particular process fragment such as order confirmation, or inventory check, it is highly likely for the client application to expect these operations to succeed or fail as a unit. We can thus view the set of operations used by the client in each process fragment as constituting an atomic unit of work (viz. atomic action).

Req 5: *Expressing application-level atomicity criteria:* In addition to req-4, the language should be able to express and associate application-level atomicity (described in section 6.1.2) criteria. For instance, we may be able to express that a transaction is a payment-aware. This means that if payment is not made within a pre-specified period then the transaction fails. Similarly, transactions could be made QoS, or SLA-aware and succeed or fail depending whether QoS criteria or SLA terms are met.

Req 6: *Expressing long duration nested-activities*: Long-duration (business) activities could be expressed as aggregations of several atomic actions and may exhibit the characteristics and behaviour of open nested transactions and transactional workflows. The atomic actions forming a particular long-duration business activity do not necessarily need to have a common outcome. Under application control (business logic), some of these may be performed (confirmed), while others may fail or raise exceptions such as time outs or failure. To exemplify a long-duration business activity, consider a

slight variation of the order processing scenario where a manufacturer asks one of its suppliers to provide it with valuable and fragile piece of equipment.

Now consider that one atomic action arranges for the purchase of this product, while a second arranges for its insurance, and a third one for its transportation. If the client application is not risk-averse (due to excessive costs), then even if the insurance operation (atomic action) votes to cancel, the client might still confirm the transaction and get the item shipped uninsured. Most likely, however, the client application would probably retry to obtain insurance for the item. Once the client discovers a new insurer, it can try again to complete the long-duration business activity with all the necessary atomic actions voting to confirm on the basis of the particular coordination protocol used.

Req 7: *Expressing and enforcing policies*: This helps SNs achieve the global control of end-to-end processes by enforcing policy consistently across the runtime environment, without requiring applications to be recoded and deployed. This involves constructs to define policies and SLA thresholds based for example on transaction averages. It also involves constructs to define and enforce policies. Some of this work will be performed jointly with research activities in WP-JRA -1.3.

Req 8: *Expressing and enforcing QoS and compliance criteria*: Activities in business transactions will be able to express compliance with regulations, SLA terms and QoS characteristics in an end-to-end fashion. The BTL will therefore be equipped with constructs to define controls and counter measures. Some of this work will be performed jointly with research activities in WP-JRA -1.3. In addition to the above characteristics the BTM system that is built around the BTL should be able to provide the following functionalities, provide application management solutions, enable the maintenance of application availability, ensure the right performance and service levels, and identify and resolve problems.

Req 10: *Incident management*: The language needs to be endowed with constructs to feed the transaction management infrastructure with information on business transaction events and SLA violations (in addition to current component based events) that is useful for repairing a process anomaly or problem, thus reducing the mean time to repair. This includes constructs to define stalled transactions, missing steps, faults, and application exceptions, as well as other issues such as incorrect data values, boundary conditions, and so on.

Req 11: *Business transaction monitoring*: Traditional transaction monitors are able to monitor only system-related activities and performance. Another capability of the BTM system is the ability to compute and monitor KPIs using rules that trigger automated alerts or process actions when they move outside their target range. Process owners can then respond instantly to events that affect the bottom line. This thus implies that the BTL will contain constructs and operators that will define exactly how processes and services will be monitored (e.g., through logging or actively checking), and how process-level KPIs will be mapped down to SBA-level SLAs and QoS (see Deliverable JRA-2.1.2).

Req-12: *End-to-end visibility and optimization*: The purpose is to provide visibility into the service interactions within the scope ("context") of the business transaction and make process performance visible to process owners and also to provide a platform for problem escalation and remediation. BTM system should provide the ability to measure performance across organizational and system boundaries and detect exceptions in service interactions. As processes run, the BTM system should be able to continuously capture the snapshots of process instance data and aggregate them in a meaningful manner.

The above list considers the most important requirements for a useful BTL that offers a flexible and extendable structure. In the initial version of the language we intend to address requirements 1-8. The remaining requirements will be addressed in close collaboration with the other work packages and lay a form foundation for the forthcoming WP-2.2 deliverables.

The requirements described above are essential in determining the characteristics and functionality of the BTL that is described in section-7 of this document.

4 Process Fragments and Business Transactions

While the exact demarcations of process fragments are still unclear, there exists some consensus that it can be defined as connected process elements that bear a high level of potential reusability in various usage contexts [Ma2008]. Process fragments can be defined at any level of granularity ranging from decision points or joins to full-fledged sub-processes. Such process fragments designate sliced models or code according to some predefined criteria. For example, a process can be dissected into process fragments or a sub-process by dissecting information flows from a BPMN model. Likewise, BPEL code may be fragmented based on the Single-Entry-Single-Exit criterion where each fragment has exactly one incoming and one outgoing edge. Note that this definition of process fragments is orthogonal to the one used in situational method engineering; there a process fragment refers to a reusable design process lifecycle [Harmsen1997].

The ability to define process fragments at various levels of granularity given various fragmentation criteria makes process fragments extremely powerful and flexible building blocks for SNs. The key philosophy behind process fragments is that they can be wired into a process, while offering a complexity reducing mechanism, leveraging reuse and boosting flexibility.

Transactional process fragments denote a special type of process fragments that exhibit conventional and application-level atomicity. In particular, transactional process fragments render common business processes that are generic, industry neutral and re-usable and can be used to develop business transactions in a multiplicity of end-to-end processes. In this way we remove excess complexity from the business transaction, allowing common business processes such as ordering, distribution and payment to be expressed in a form analogous to abstract data types and rationalizing them across an end-to-end process. The basic structure of these transactional process fragments (which could be also appropriately extended and customized) provides a standard definition for building business solutions. Transactional process fragments not only help to streamline and rationalize common business practices across an end-to-end process, they also help to enforce participant commitments.

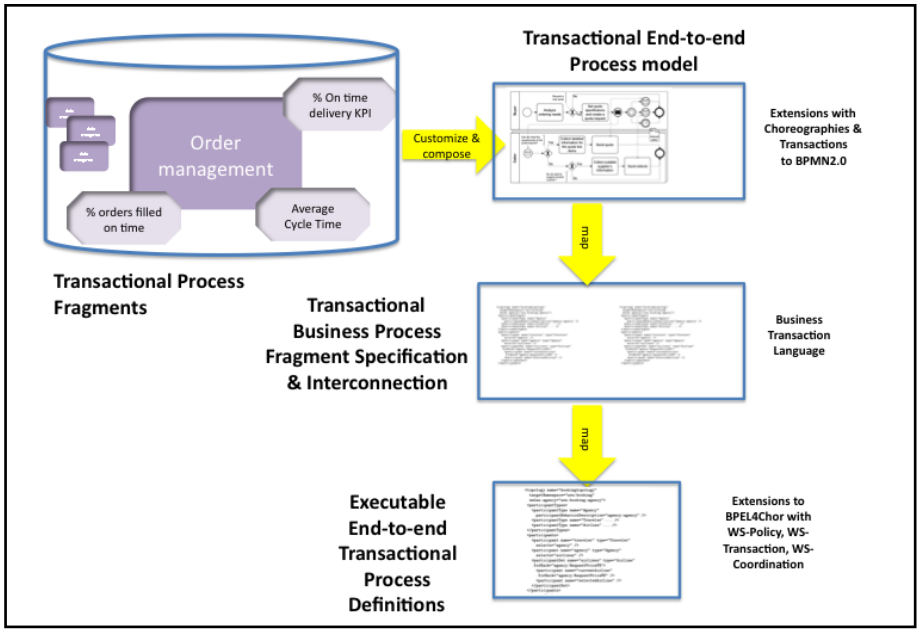

Figure 2 Modelling and enacting business transactions

Parameterization plays a pivotal role in customization of transaction process fragments: the parameterization process allows setting parameters of transactional process fragments so that they satisfy granular process properties. For example, delivery notes can be processed in an order management transactional process fragment without any reference to a preceding order (option 1), referencing an individual sales order (option 2), referring to a delivery-due list (option 3), or from a stock transport order (option 4). In particular, with customization we imply that the parameters/variants of granular features (processes, operations and/or events) in a transactional process fragment are set according to specific end-to-end process requirements. More specifically, we may distinguish between the following types of parameters [Czarnecki2000]:

- Dimension whose all sub-features represent a variant
- Dimension with optional features
- Extension point
- Extension point with optional features
- Extension point with OR-features

Our initial approach towards leveraging parameterization is inspired by [Rosemann2007], in which extensions to classical Event Process Chains (EPCs) are proposed that allow for configurable functions (that may be included, skipped or conditionally skipped) and connectors (Joins and Splits).
For this purpose, parameterization will be extended to accommodate configuration of transactional granular process properties such as SLAs, QoS, policies and rules.

Once customised, transactional process fragments can be aggregated into end-to-end processes and then may be implemented as transactional orchestrations or choreographies. This is firstly achieved at the modelling level–using an extension to BPMN-2.0 that accommodates transactional choreographies not only involving data- and control flow, but also granular process properties such as SLAs, QoS and policies (see Figure 2). This figure represents a high-level view of how business transactions can be modelled, defined and executed.

Next, transactional BPMN models are transformed to BTL and afterwards are mapped into executable transactional end-to-end processes possibly using a combination of BPEL4Chor, WS-transaction, WS-Coordination and WS-Policy.

5 Motivating Scenario

The following scenario deals with an integrated logistics process involving a customer, suppliers and a logistics service provider. This logistics model consists of forecast notification, forecast acceptance, inventory reporting, shipment receipt, request and fulfil demand, consumption and invoice notification activities. In particular, this scenario is part of the automotive supply chain case study proposed in JRA-2.1 vision paper, illustrating the use of simple business transactions and associated event monitoring.

Figure 3 depicts the flow of information between the interacting nodes (business partners) in a very simple SN involving three parties (car manufacturers, part suppliers and logistics providers). Service interactions are governed by a simplified SLA. This figure shows the sequential ordering of the interaction events between the business partners in terms of message exchanges. The message "Notify of Forecast", contains car part demand information (planning demand), is sent by a forecast owner (the car manufacturer) to a forecast recipient (the supplier).

The message “Forecast Acceptance" sent back from the supplier acknowledges that the demand forecast has been accepted. Next, the message "Distribute Inventory Report" is performed by an inventory information provider to report the status of the inventory to an inventory user. The inventory

report can include any car part held in inventory. The message "Advance Shipment Notification" allows a shipper to notify a receiver that a shipment has been assigned. This notification is often a part of the shipment process. Message "Shipment Receipt" is performed by a consignee to report the status of a received shipment to another interested party, such as another consignee, a transport service provider, a third-party logistics firm, or a shipper.

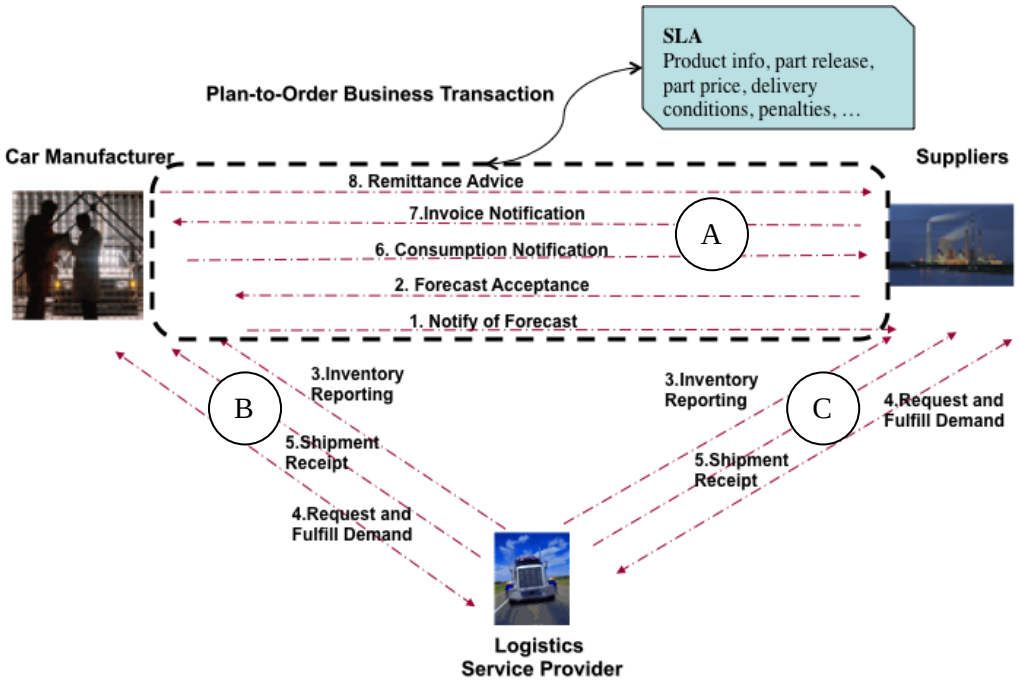

Figure 3 The Integrated Logistics Scenario

Receipt of a shipment is reported after it has been delivered by a carrier and inspected by receiving personnel. The customer then issues an "Invoice Notification" to communicate car part consumption to the supplier, allowing the supplier to trigger invoicing for the consumed material. The message "Invoice Notification" enables a provider to invoice another party, such as a buyer, for goods or services performed.

Finally, the message "Notify of Remittance Advice" enables a payer to send remittance advice to a payee (in this case the supplier), which indicates which payables are scheduled for payment.

Message exchanges are bundled together in three separate message exchanges (A, B and C). As shown in figure 3, the message exchanges 1,2,6,7 and 8 (named A) can be bundled together in the form of a business-aware transaction, which is governed by a simplified SLA between the two trading partners, i.e. the car manufacturer and the supplier.

As a whole, our global business transaction scenario consists of three binary business transactions, i.e. transaction A between the car manufacturer and the supplier, B between the car manufacturer and the logistics service provider, and C between the logistics service provider and the supplier. The SLA for each transaction contains a set of policy constraints, such as temporal and spatial constraints, penalties, some business regulatory rules, etc, that must be fulfilled during the transaction execution. It also prescribes the recovery strategies for the business transaction in case any of the message exchanges fail, for instance, either this business transaction must be compensated by some means, e.g.,issuing another forecast or invoice, or the entire transaction fails. Besides, the SLA also drives the business-aware transaction with other agreements on the KPIs of the transaction, which enables the performance monitoring and measurement of the execution. To sum up, a business transaction between two trading partners is driven by the conditions specified in the agreed-upon SLA.

Some sample conditions in the SLA that should be monitored and enforced by the transaction management system during the transaction execution are:

- Has the supplier acknowledged the order?
- Has the supplier and logistics service provider committed to a ship date?
- Will the supplier start manufacturing on-time?
- Will the order be shipped on-time?
- Does this order meet our on-time delivery goals and other KPIs?
- If the order shipped by the logistics provider does not arrive on-time, how should we proceed?
- Does it affect other partners if the logistics service provider cannot deliver the order? How to compensate this problem?

Figure 4 provides more details of the sample integrated logistics scenario using BPMN modelling notation. In this multi-party BPMN process model, not only are the public messaging activities between the partners modelled, but also the private process activities inside each partner. The process activities and message exchanges that belong to a business transaction are marked with the same color, A with blue, B with yellow, and C with magenta. In the scenario (in Figure 4), we assume that all three partners have already agreed upon following the business protocol, i.e. the sequential ordering of business interactions, described in Figure 3.

Furthermore, they have also agreed on some particular KPIs specified in the model, e.g. the time for preparing and sending an invoice by the supplier must be equal or less than two days.

A process activity can be vital or non-vital for its business transaction. If the vital one fails, its transaction will fail, e.g. the activity "Prepare Invoice" in transaction A is vital for its transaction, since without the invoice, the payment between the trading partners in A cannot be conducted. Process activities in the same or different business transactions can have weakly or strongly consistent atomicity relationships. "Weakly" means that either both the two activities succeed or if one of them fails and is compensated/recovered then the other one will also fail or be compensated/recovered in the near future. "Strongly consistent-atomic" enhances the "weakly consistent-atomic" relationship by requiring that if both activities fail, they must result into a consistent (predefined) state (see section 8.1.2 which explains formalization of consistent-atomicity). For instance, the activities "Send Invoice" in transaction B and "Receive Invoice" in transaction C are strongly consistent-atomic.

Traditional transaction monitoring mechanisms are able to monitor only system-related activities and performance. However, it is important to understand that business-aware transactions correlate application-level events and situations with the supporting infrastructure. For example if the manufacturer requests a change in the order, can we accept the change in connection with agreed-upon KPIs? Alternatively, if an infrastructure-level change has been made, we can assess its impact on the application (SBA) level. More importantly, we can deduce if the processes are still working to plan, if there are any bottlenecks and where they appear, if there was a process improvement or worsening, and so on. Correlating lower level activities, e.g., from the service composition or the infrastructure level, with higher level business events in the form of transactions, provides opportunities to continuously monitor and measure the lifecycle of a transaction, while providing data and events to trigger and populate controls, as well as time-based data for on-time measurement.

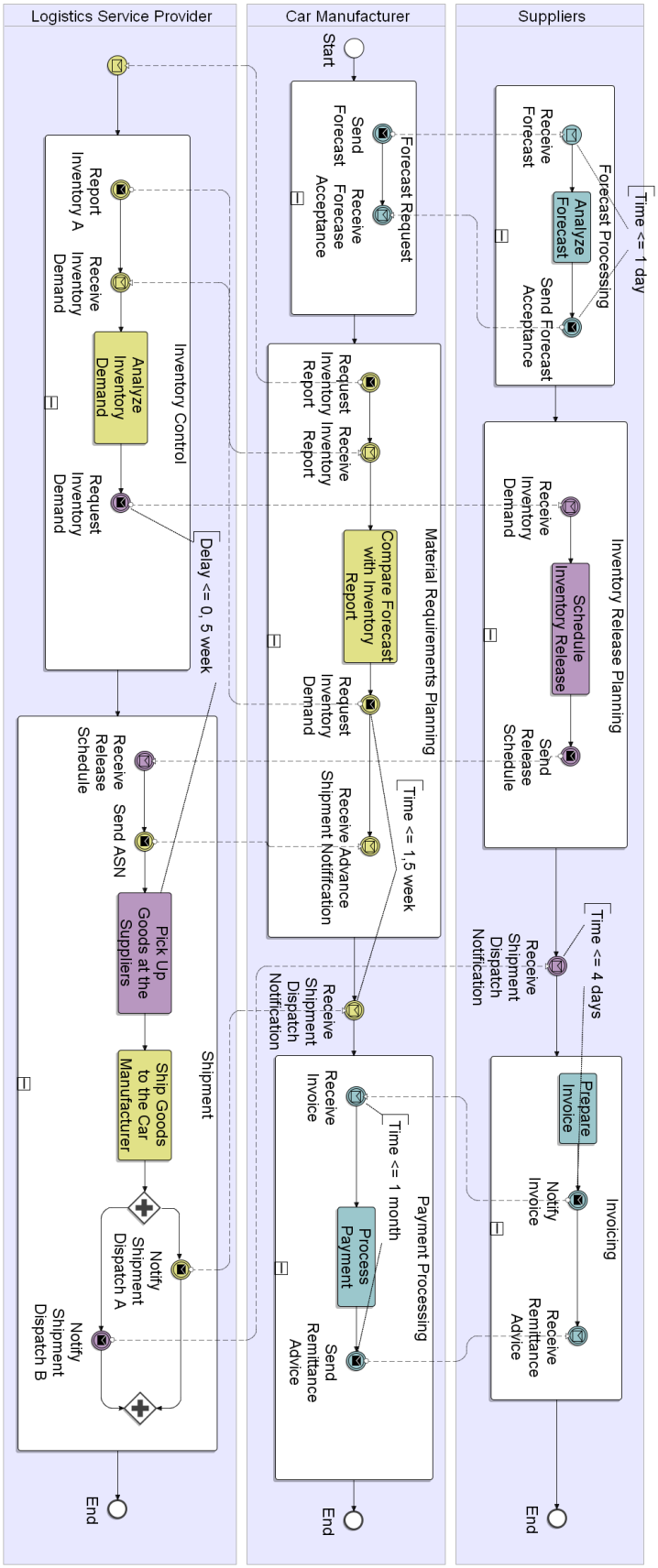

Figure 4 BPMN Model of the Integrated Logistics Scenario

6 Business Transaction Model

From the definition of a business transaction, it has been shown that it is a mission-critical task that spans across the organizational boundaries and encompasses different types of business concepts to achieve business awareness, such as SLA awareness. To provide a more in-depth understanding of a business transaction, this section first scrutinizes the business concepts that significantly pertain to a business transaction and proposes a business transaction model on the basis of which the transaction language described in section-7 can be developed. Section 6.1 describes high-level concepts of business transaction that are closely related and substantial to end-to-end business processes. This section concentrates on rationalizing their indispensability in a BTL. Then, an advanced model of business transaction is presented is section 6.2 that captures these business transaction concepts.

6.1 High-Level Concepts of Business Transaction

A collaborative business environment involves multiple partners (or organizations) that foster the indispensability of mutual obligations. The mutual obligations can be presented by an SLA that explicitly defines the common requirements and policies that have to be satisfied by the committed trading partners in a SN. Business transaction is deemed as one of the main drivers of an end-to-end process that involves multiple trading partners. Hence, any transaction mechanism that is used to enforce the business transaction in an end-to-end process needs to be cognizant of SLA concepts. These SLA concepts are described in this section including application-level atomicity that governs the transaction process. Besides, the *Business Collaboration* concept is discussed to present a brief overview of business transaction in a collaborative business environment.

6.1.1 Business Collaborations and Business Transactions

One key requirement for enabling cross-enterprise business process automation is the ability to describe the collaboration aspects of the business processes, such as commitments and exchanges of monetary resources in a standard form that can be used by applications and consumed by tools for business process implementation and monitoring [Papazoglou2003a]. Business collaboration captures the information and message exchange requirements between trading partners. Additionally, trading partners may have multiple interactions in an end-to-end process. The sequence of these interactions is captured by a *business protocol*. A business protocol identifies and captures all behavioural aspects that have cross enterprise business significance [Papazoglou2006]. Behavioural aspects may include the messaging behaviours between trading partners, which help the participants understand and plan their interactions that conform to the agreed-upon business protocol. We have incorporated a set of important business related artefacts (e.g., business protocol) in our business transaction model.

6.1.2 Application-Level Atomicity

The governing factors (e.g. SLAs) of a business transaction foster the atomic behaviour. This type of atomicity is called application-level atomicity or non-conventional atomicity [Papazoglou2003a] and consists of a set of application related atomicity criteria. Each criterion is treated as a single individual logical unit of work that determines a set of vital or viable outcomes for a business transaction. The outcomes of a business transaction may involve non-critical partial failure, or selection among contending service offerings rather than the conventional strict atomicity (all or nothing).

Application-level atomicity can be deemed as the criteria for checking consistency and correctness of a business transaction against predefined standard operations. According to [Papazoglou2003a], there may be different types of application-level atomicity that are discussed in the following:

- *Contractual Atomicity:* Business transactions are generally governed by contracts and update accounts. Contractual atomicity that can be considered as one of the most significant SLA concepts, includes messaging sequence (or interactions), QoS parameters (e.g., time to perform), and security parameters (e.g., non-repudiation). Electronic contracts define both the legal terms and conditions and technical specification that a trading partner must implement to put an electronic trading relationship into effect. As an example, if a *contract* enforces an obligation to acknowledge a *purchase order* within specified time frame, seller has to send an acknowledgement to buyer. Otherwise, the transaction should be aborted and failed. Consequently, the contract will be null and void. A business transaction is completed successfully only when the contractual provisions are satisfied.

- *Operational Level Atomicity:* Business transactions usually involve the actual delivery of purchased items (tangible and non-tangible) [Papazoglou2003a]. This type of atomicity has been well defined by [Tygar1998] and refined by [Papazoglou2003a]. The operational atomicity is decomposed into *Payment Atomicity*, *Goods Atomicity*, and *Delivery Atomicity*. This research does not explain *Goods Atomicity* but emphasizes on payment atomicity and delivery atomicity since *Goods Atomicity* can be realized by *Delivery Atomicity*.

  - *Payment Atomicity*: Payment atomic protocol affects the monetary transaction from one party to another. It is the basic level of atomicity that each operation level business transaction should satisfy. This type of atomicity has greater influence on the entire transaction process because if the payment process fails, all the cohorts must have to be failed. Notably, a payment atomic transaction can be contract atomic.

  - *Delivery Atomicity*: Delivery is typically the last phase of an end-to-end process chain. The purpose of this type of atomicity is to ensure that the right goods are delivered to the buyer. The delivery atomic protocol also guarantees the quality of the specified products is maintained and they are delivered within the specified. A business transaction cannot be completed successfully unless a *Delivered* notification arrives from the buyer, since the failure of delivery may cause failure of all the (sub-) transactions that participated in the transaction process.

Our business transaction model adapts the application-level atomic criteria to achieve a consistent and business-aware transaction mechanism since application-level atomicity contains higher business significance involving contract, constraints, and also the operations. The association of business transaction with these business aspects is shown in the business transaction model in the next section. Noticeably, these atomicities (contract and operational level atomicity) are not shown explicitly in the business transaction model instead they are represented by contractual primitive and operational primitive.

6.2 Overview of Business Transaction Model

An important requirement in making end-to-end process automation happen is the ability to describe the collaboration aspects of the processes, such as commitments and mutual obligations, in a standard form that can be consumed by tools for business process implementation and monitoring. This gives rise to the concept of a ***business transaction model*** that provides a comprehensive set of concepts and several standard primitives and conventions that can be utilized to develop complex Service Based Applications (SBAs) involving transactional process fragments (see Section 1).

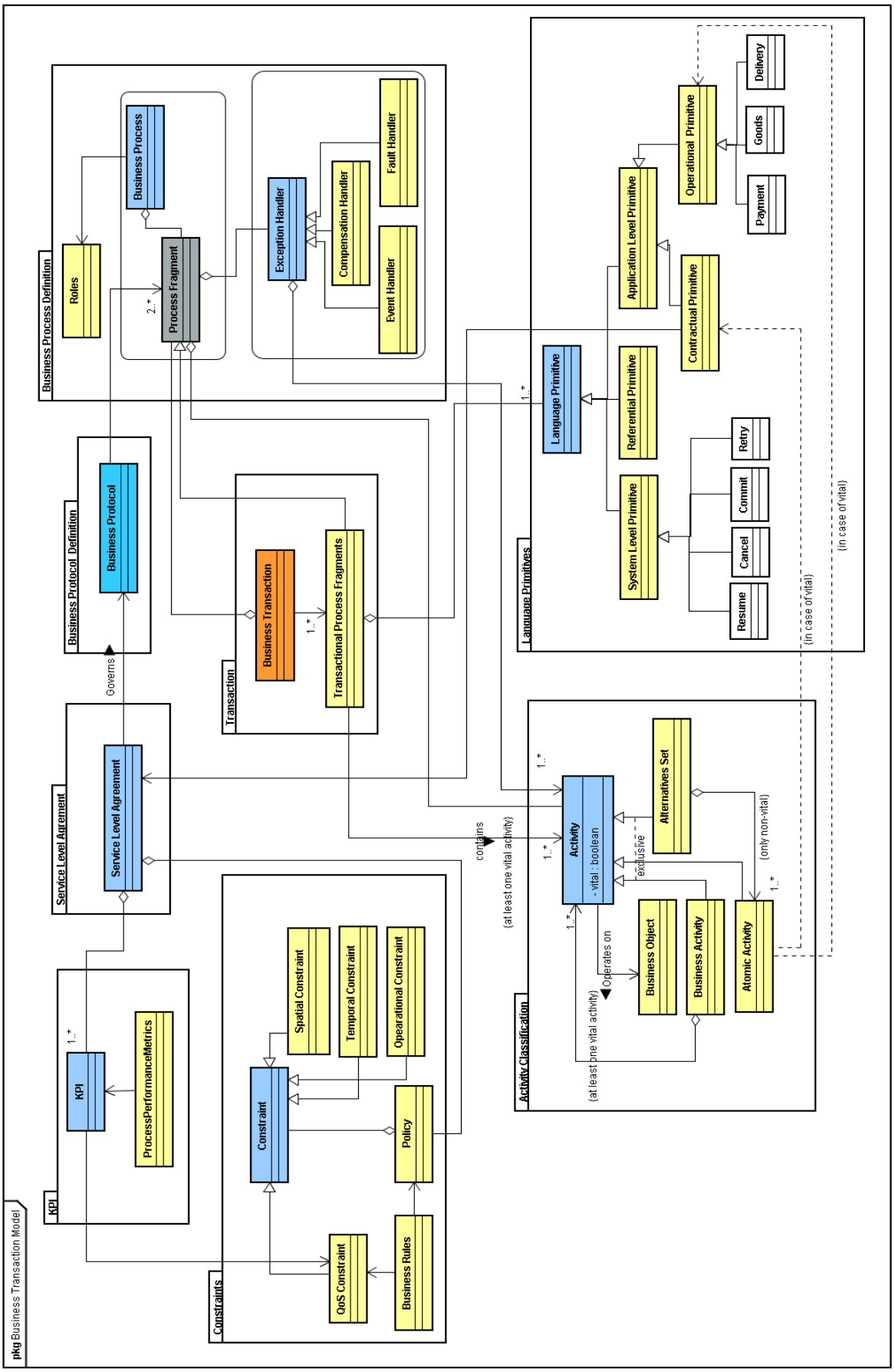

Figure 5 The Business Transaction Model in UML

Central to the business transaction model is the notion of a business transaction (see section 2.1 for the definition). Business transactions cover many domains of activity that businesses engage in, such as request for quote, supply chain execution, purchasing, manufacturing, and so on. The purpose of a business transaction is to facilitate specifying common (and standard) business activities and operations that allow expressing business operational semantics and associated message exchanges as well as the rules that govern them. The combination of all these primitives enforces SN partners to achieve a common semantic understanding of the business transaction and the implications of all messages exchanged.

The business transaction is initiated by a service client and brings about a consistent change in the state of a relationship between two or more network parties. A business relationship is any distributed state held by the parties, which is subject to contractual constraints agreed by those parties.

There are four key components in a business transaction model that help differentiate it from (general) message exchange patterns. These are:

- Commitment exchange;
- The party (or parties) that has the ability to make commitments;
- Business constraints and invariants that apply to the message exchanged between the interacting parties; and,
- Business objects (documents) that are operated upon by business activities (transactional operations) or by processes.

We have developed a meta-model in UML that captures the key constructs of the business transaction model and their interrelationships (see Figure 5).

Business transactions make up the core of the transaction model, and may as such incorporate a blend of transactional and non-transactional process fragments that are coordinated through *business protocols* and collectively make up an end-to-end process.

Transactional process fragments are characterized by universally acceptable *system level primitives* – such as resume, cancel, commit and retry. There are also *referential primitives* that correlate an activity with other activities using control or data flow, e.g., payment refers to an associated order. *Application level primitives* comprise of *contractual primitives* and *operational primitives*. *Contractual primitives* define mutual agreements between network parties relying on constructs such as authorizations, obligations and violations. Lastly, *operational primitives* help to enforce network partner commitments. They introduce a mandatory set of three operational level atomicity criteria that reflect the operational semantics of three standard business activities (ordering, payment, and delivery). For instance, payment atomicity affects the transfer of funds from one party to another in the transaction. This means that the transaction would fail if payment is not made within a pre-specified time period that was agreed between a supplier and a customer. Delivery atomicity, on the other hand, implies that the right goods will be delivered to a customer at the time that has been agreed.

Transactional process fragments embody one or more activities that fall apart in the following two mutually exclusive activity types: *vital* or *non-vital*. *Atomic activities* are short-lived and atomic actions which in some cases may be part of long-running business transactions, referred to as *Business Activities* (adopting the WS-Transaction terminology). *Alternatives set* is a group of alternative atomic activities and can be *vital* or *non-vital*. Alternatives set can contain only non-vital atomic activities since it is not pre-defined which alternative is executed at run-time.

Business activities usually operate on *business (document-based) objects*. These are traditionally associated with items such as purchase orders, catalogues (documents that describe products and service content to purchasing organizations), inventory reports, ship notices, bids and proposals. Such

objects may also be associated with agreements, contracts or bids. This allows business transactions to interchange everything from product information and pricing proposals to financial and legal statements.

Activities could also belong to the *Exception Handler* e.g. *Compensation Handler, Event Handler,* and *Fault Handler*. They are special kinds of activities that are performed in case of particular activity fails or repair the transactions after they were disrupted.

An *SLA* is defined as a coherent set of explicitly stated *policies* that prescribe, limits, or specifies any aspect of a business protocol that forms part of the commitment(s) mutually agreed to among the interacting parties, and stipulates agreed upon *QoS constraints* (some of which stem from *business rules*). An SLA also outlines what each party can do in the event the intended actions are not carried out (e.g., promised services not rendered, services rendered but payment not issued).

Policies may encompass one or more *constraints*. *Spatial constraints* regulate the access to protocols based on criteria such as spatial point and route. *Temporal constraints* stipulate timing restrictions such as time, date, duration and interval. The business transaction model also furnishes some standard operational constraints for governing message exchanges. The model could easily express sequencing semantics, which for instance require that "ordering" occurs first and is followed by "transport" and "delivery". "Payment" can happen either before or after the "delivery" function.

7 Initial Design of Business Transaction Language (BTL)

Our main goal in this deliverable is to design a BTL for expressing transactional properties of SBAs. The business transaction model (described in the earlier section) already covers the scope of the language.

The next phase is targeted to develop a BTL, that is, to transform our transaction model into language constructs which will become the basis of the next deliverables of the WP-JRA-2.1. This section presents some initial and elementary constructs that we have developed and captured in a transaction language named as BTL. The BTL is a declarative transaction language and its constructs are XML-based representing the transactional elements specified at design time. BTL is also planned to facilitate annotating the granular process properties for each of the activities in the process fragments that compose end-to-end processes. Noticeably, a detailed description of the run-time environment as well as run-time transformation between the BTL and equivalent executable constructs is not within the scope of this deliverable. The mapping mechanism will be developed in the upcoming deliverable. Potential mapping will most likely be from BTL to an execution language, e.g., an extended transactional BPEL. Hence, the purpose of this section is confined only to our elementary work on BTL constructs.

The business transaction model in the previous section consists of various domains (e.g. SLA, Constraints, Primitives, etc) that are related to business transactions, their elements, and associations between the elements as well as domains. In the section, we codify the model considering all these elements and relations. Figure 6 presents a simple example of BTL code snippet and is used for illustrating purposes only.

This XML snippet is an example of using the BTL constructs defined in section 6.2 to describe part of the Integrated Logistics Scenario in Figure 3. In particular, we focus on the *process fragment* "Payment Processing" at the car manufacturer, which is in fact a *transactional process fragment*.

First, the fragment uses a *Business Protocol*, which prescribes the sequence order of activities and interactions performed by the car manufacturer that must be visible for other participants. An activity can be *vital* or *non-vital*. For instance, "Process Payment" is considered to be vital since it actually

```
<BTL>

    <TransactionalProcessFragment name=”Payment Processing”>
      <BusinessProtocol>
        <Sequence>
          <Activity vital= “false”>Receive Invoice</Activity>
          <Activity vital= “true”>Process Payment</Activity>
          <Activity vital= “false”>Send Remittance Advice</Activity>
        </Sequence>
      </BusinessProtocol>
      <LanguagePrimitive>
       <applicationLevelPrimitive>
        <operationalPrimitive>
          <Activity vital= “true”>Process Payment</Activity>
          <extend>Payment</extend>
        </operational Primitive>
        <contractualPrimitive>
          <SLA>
            <Policy>
              <timeToComplete>1 month
              </timeToComplete>
               <creditMaximumAllowance>1000$
               </creditMaximumAllowance>
            </Policy>
          </SLA>
        </contractualPrimitive>
      </applicationLevelPrimitive>
     </LanguagePrimitive>
    </TransactionalProcessFragment>

</BTL>
```

Figure 6 Sample code snippet of Business Transaction Language

executes the payment, while “Receive Invoice” and “Send Remittance Advice” are non-vital since it may be no problem at all for the supplier to resend an invoice a few times more and to check the payment without the remittance advice.

Second, the *transactional process fragment* may be driven by many *Language Primitive*, which are in this case the *operational primitive* and *contractual primitive* that are the subsets of *application level primitive*.

The *operational primitive* indicates that in order to realize the *vital activity* “Process Payment”, we will extend the existing predefined operational primitive “*payment*”. The *contractual primitive* prescribes the agreed-upon *SLA* between the participants, which comprises of a number of *constraints* embedded in a *policy* definition. In this case there are two policy constraints, one prescribes that the completion time for this fragment must be within a month, and the other one allows the payment using credit card only up to 1000$.

8 Formal Underpinnings of Business Transactions

In section 6, the conceptual design of business transaction has been outlined, and in Section 7 an initial business transactional language (BTL) has been introduced based on XML and a semi-formal meta-model to capture and represent various transactional properties that the end-to-end process model (BPEL program) must satisfy. In order to be able to automatically verify the adherence between the two specifications (business process specifications on one side and transactional properties on the other side), both specifications should be based on a formal foundation of a logical language to pave the way for the application of future automatic analysis and reasoning techniques. In addition, and more importantly in the context of this deliverable, formalization is needed to ascertain the construct and internal validity of the BTL. However, both BPEL and BTL are not grounded on a formal model. BPEL and BTL should be automatically mapped to formal models to enable the automatic verification between the two specifications.

Process Algebra (PA) is a formal description technique to specify software systems, particularly those constituted of concurrent and communicating components. Many PA languages have been proposed each with different levels of capabilities, such as CCS, ACP, CSP, π-calculus, LOTOS, or Timed CSP [ref]. On one side, we propose to formally represent end-to-end business process model using a process algebraic language.

On the other hand, various transactional properties can be formally specified using a family of languages called temporal logics, such as Propositional Linear Temporal Logic (PLTL) [Emerson1990]. Consequently, advanced and sophisticated automated verification tools associated with PA can be utilized to ensure the conformance between the end-to-end business process model and the relevant transactional properties. Many model checking methods are available to check the validity of such temporal logic expressions capturing the relevant transactional propertied over a system formally represented using a process algebraic model [Hoare1985].

In addition, the automated verification tools associated with PA can also be utilized to check that two systems at different levels of abstraction are equivalent. This is particularly important when refining from the global view of the system to the local one.

It is important to mention that, the formal foundation is targeted to check the conformance in design-time, because although the proposed business transaction language is aimed at covering both design-time as well as run-time utilization, this deliverable is particularly focused on design-time.

There exist already a few proposals dealing with advanced business transactions over the last decade [Kratz2008] but the ideas of "vitality" and the "consistent-atomicity" have not been formalized even in the most "business-oriented" transaction works, e.g. [Papazoglou2006] or [Vonk2003] and [Vonk2008]. Furthermore, the global oriented way to specify contract leads to a systemic problem, i.e., global view vs. local view in the enforcement of contract because of partial knowledge of roles. In [Hantry2009] we try to give a formal semantics to vitality, consistent-atomicity and operational atomicity in a global model, using both declarative temporal logic [Rychkova2008] over transactional statement as well as StAC [Ray2006] to model the imperative version of our definition. A proposition to refine from global to local view would be using B method [Emerson1990] refinement with StAC, and we could enhance the model with roles and role refinement by building on the ideas from [Rychkova2008], that has not really been formalized.

This section is organized as follows: section 8.1 presents the formal definition of vitality and atomic consistency that are expressed using PLTL. The vitality and consistent-atomicity are significant to design a transaction layout for SBAs in SNs. The induction properties of business transaction are discussed in section 8.2. Section 8.3 describes a preliminary formal framework for business transactions and lastly, section 8.4 explains formal challenges and future works.

8.1 Formal Semantics of Business Transactions Concepts

From the business transaction model (explained in section 6.2), it is clear that activities can be different types such as vital activity. We concentrate on defining vital activity since vitality of an activity can enforce a business transaction to be atomic in nature. The vitality concept is tightly coupled with the concept consistent-atomicity which is also formally defined in this section. The atomicity is in fact, application-level atomicity (explained in section 6.1.2) that is represented as contractual and application level primitives in the business transaction language.

Traditionally, in advanced transaction models [Emerson1990], transactional statements can be begun, executing, aborted, and committed. The statement "aborted" means that, the transaction execution has failed either because of internal failure, or external interruptions.

In the following discussion, for the sake of convenience, we will use the intuitive notion of "undo" for different transaction effects. A transaction is undone when it is either rolled back (backward) in case it has been aborted or compensated in case it has been committed already.

Taking [Ray2006] into consideration, many transactional dependencies can be summarized by the composition of rules.

Consider T and T' are two sub-transactions. The rules to exhibit transactional dependencies between them can be summarized as follows:

1. After process has terminated, in the history of transactional statement, if T (transaction) has committed (resp. begun, aborted, executed) then T' has committed (resp. begun, aborted, executed). Note that, there is a possibility that T has not committed in this case.
2. After process has terminated, in the history of transactional statement, T has committed (resp. begun, aborted, executed) after T' has committed (resp. begun, aborted, executed). Unlike the previous rule, T must have committed in this case.

Both the above rules and their combinations can be formally expressed using well known program specification language LTL [Emerson1990] as argued in [Ray2006]. LTL is an intuitive and language. It is a logic used to formally specify temporal properties of software or hardware designs. In LTL each state has one possible future and can be represented using linear state sequences, which corresponds to describing the behaviour of a single execution of a system.The primitive temporal operators of LTL are F, G,U,X,A. The language is built by induction starting either from Boolean formula, or first order logic formula.

Consider we model transactional statement using finite number of Boolean variables. In this case, the system can be described as a state transition automaton, in which each state is labeled with an interpretation over these Boolean variables. This automaton is named as Kripke structure. Given that a possible execution path P produced by the automaton (a sequence of states such that for any two ordered neighbor states there exists a corresponding transition) with P= (S1,S2,…..) and a and b are two formulas in LTL, then

- If 'a' is a Boolean formula, then 'a' is true for P if only if the interpretation of S1 is a model for 'a'.
- formula F(a) is true for path P if and only if 'a' is true at a time point in the future of P i.e. 'a' is true for a suffix path (Sk,S(k+1)…..) of P.
- G(a) is true for path P if and only if 'a' is always true at any time point in the future of P i.e. 'a' is true for any suffix path (Sk,S(k+1)…..) of P,

- U(a,b) is true for path P if and only if 'a' is continuously true from the current time point until that there exists a time point in the future while 'b' is true, i.e. there exists k such that 'a' is true for any suffix path (Si,S(i+1)……) of P with i < k and b is true for suffix path (Sk,…) of P.
- X(a) is true for P means 'a' is true at the next time point i.e. 'a' is true for the suffix path (S2,….) of P
- Once the LTL formula 'a' has been built, it is usually assessed from one state over all the possible execution paths starting from that state i.e. A(a) is true at a given state S if and only if 'a' is true for any possible execution paths P starting from S : (S,…..).

The FOLTL [Emerson1990] is the first order logic version of LTL. It is inductively built with temporal operators, Boolean operators and existential quantifier over variables from the first order logic formula, starting from first order logic formula (for more detailed see [Emerson1990]).

In an end-to-end process, application-level atomicity (e.g. contractual atomicity) enforces various public constraints in particular, business rules, contract, or external regulatory laws. The vitality and business consistent atomicity notions contribute significantly to express these constraints. We will show how LTL can be used expressing vitality and application level consistent-atomicity.

8.1.1 Formal Definition of Vitality

An end-to-end process may consist of several vital activities (it contains at least one vital activity). To exemplify, we consider a two vital activities may originate from two different process fragments. They are as follows:

- "Checkorder" is vital for ordering
- "Certificate delivery" is vital for shipment

There is a need to provide a formal semantics to the vitality. We start with revising different possible business rules to enumerate different possible transaction behaviors of activities.

First, if "ordering" commits then "Checkorder" must have been committed. Second, if "Checkorder" is aborted, then ordering must be aborted in a near future. Third, if the ordering is undone, then "Checkorder" may have to be aborted or not, compensated or not, or deviated or not in the history of the transaction (whatever the order). Thus, "Checkorder" might or might not be aborted/compensated. On the contrary, this third point constitutes another case for the second business rule. Indeed, it is clear that "certificate delivery" activity has to be aborted when shipment is aborted, and compensated if shipment is compensated. Thus, we believe this third constraint implies a distinction between weak vitality and strong vitality. We provide the formal definitions for these vitality concepts in the followings:

- ***Weak vitality***

  Let's assume T' is a sub-transaction of T. T' is weakly vital for T if and only if:

  - for any execution, at any time, if T has been committed , then T' must have also been committed
    - $AG\ (commit - T \Rightarrow commit - T')$
  - for any execution, at any time, if T' has been aborted then T is aborted in a close future
    - $AG\ (aborted - T' \Rightarrow F\ (abort - T))$

- for any execution, at any time, if T ' has been compensated then T is aborted or compensated in a close future.
  - $AG\ (Compensated - T' \Rightarrow F\ (aborted - T \vee Compensated - T))$

- ***Strong vitality***

Now assuming that T' is a sub-transaction of T, T' is strongly vital for T if and only if:

- T' is weakly vital for T
- for any execution, at any time, if T aborted then T' is aborted or compensated in a close future.
  - $AG\ (aborted - T \Rightarrow F(abort - T' \vee Compensated - T'))$
- for any execution, at any time, if T has been compensated then T' is aborted or compensated in a close future
  - $AG\ (Compensated - T \Rightarrow F\ (aborted - T' \vee Compensated - T'))$

Considering the above definitions of vitality, we can summarize that "Check order" is weakly vital for "ordering", but "certificate delivery" seems to be strongly vital for "shipment".

8.1.2 Formal Definition of Consistent-Atomicity

We already described the application-level atomicity (see section 6.1.2) that is represented as *consistent-atomicity* in this section to explain the formalization of the concept conveniently. The title consistent-atomicity is considered to formally define the application-level atomicity encompassing consistency that has been introduced to cover the business concepts. The consistency can be ensured once the application atomicity property is satisfied (e.g. the transactions are said to be consistent when it satisfies contractual atomicity such as business rules). Intuitively, if a specific condition cannot be ensured by two (or a set of) transactions, then these two (or more) transactions have to be undone.

Note that in contrast to classical transactional notions such as ACID [Gray1993], SAGAS [Molina1987] , or nested transactions [Eliot1985] , this new kind of atomicity may cross-cut the structure of the normal execution flow. Furthermore, two transactions can be atomic even though none of them is a sibling of the other, i.e. they do not have the same parent transaction.

Conceptually, the parent transaction is switched by "reference" and "cross-cutting" constraints. Similarly, to a parent transaction, a constraint could abort or commit. A typical example is the contract in an end-to-end system. The followings are examples of constraints:

- If payment and shipment appears in the same business process, then shipment must be performed after payment.

- Payment atomicity [Papazoglou2006]: An operational payment can be decomposed into two sub-transactions "change account of role A" and "change account of the role B". At any time, both roles check their accounts. If the both of them eventually find the same corresponding change, then transactional payment statements can continue. Otherwise "change account" has to be failed.

Payment and shipment are linked only by the first constraint and independent from the normal execution structure. However, note that there is no constraint on the business states corresponding to the failed transactional states. Thus, we distinguish two types of consistent atomicity namely, *Weak-*

consistent atomicity and *Strong-consistent atomicity* described in the following. We use the FOLTL specification in order to express conditions.

- ***Weak-consistent atomicity***

  Assume T and T' are two transactions, and Cond is a first order Logical Formula of business rule constraints using over Global history variables, then T and T' are weakly-consistent atomic if and only if

  - for any path, at any time, if T and T' have committed then Cond must be true
    $AG\ (commit - T\ \wedge\ commit\ T' \Rightarrow Cond)$
  - for any path, at any time, if cond is false then both T and T' will be failed, i.e. aborted and eventually compensated, in a close future

  $AG\ (\neg\ (cond) \Rightarrow F\ ([abort\text{-}T \vee Compensated\ (T)]\ \wedge\ [abort - T' \vee\ Compensated - T']))$

- ***Strong-consistent atomicity***

  Assume T and T' are two transactions, and cond1, cond2 are two Logical formulae of business rule constraints assessed over the Global history variable. T and T' are strongly-consistent atomic if and only if

  - T and T' are weakly-consistent atomic
  - Once they have failed together then cond2 must be true.
    $([abort - T \vee Compensated - T]\ \wedge\ [abort\text{-}\ T' \vee\ Compensated - T']) \Rightarrow cond2$

Regarding the first constraint in our example, payment and shipment are weakly-consistent atomic with the condition "shipment is after payment". If any execution fails because of this condition, then both must be undone no matter what their transactional statements are. However, there is no condition regarding the terminated failed point. It poses few questions in particular, must the payment be undone as a simple payback or, following the penalties described in the contract, B will require a penalty from the violator A while paying back? On the contrary, in the second constraints, both "change account" are strongly consistent atomic, because the constraint enforces the condition to be fulfilled on the terminated failed point.

The following formulae describe (formalize) such a constraint:

- $Cond1 \Leftrightarrow Account_A.commit(Payment) = Account_A.starting(Payment) + Price\ \wedge$
  $Account_B.commit(Payment) = Account_B.starting(Payment) - Price$
- $Cond2 \Leftrightarrow Account_A.aborted(Payment) = Account_A.starting(Payment)\ \wedge$
  $Account_B.aborted(Payment) = Account_B.starting(Payment)$

Where, *Account.current.A* and similar names and *Price* stand for variables in the first order logic. Names labeled by *starting* are historical variables corresponding to the value on starting payment. Given some vitality and consistent-atomic dependencies, and similarly to the inference principle illustrated in business rules engine as Drools [Browne2009] or Web sphere business event [IBM2001], we argue that some other dependencies can also be inferred. Following [Ray2006], these kinds of dependencies stem from the temporal logic properties, and are critical at design time. Then, some shortcuts could also be provided while analyzing the temporal dependencies. We explain these dependencies in the next.

8.2 Induction Properties of Transactional Dependencies

8.2.1 Vitality is Transitive

Theorem 1: Assume T' is a weakly vital sub-transaction of T, and T" is a weakly vital sub-transaction of T', then using the LTL semantics we can easily show that T" is weakly vital sub-transaction of T. It means that the notion of weak vitality is transitive. The same properties also hold for strong vitality.

If the signature activity (non-repudiation) is weakly vital for the certificate delivery activity and certificate delivery activity is strongly vital for shipment then we can deduce that signature activity is at least weakly vital for shipment.

8.2.2 Consistent-Atomicity Propagation

Assuming that T'1 and T'2 are sub-transactions of T1 and T2 respectively, and T'1 and T'2 are weakly-consistent atomic transactions (with cond) then,

Theorem 2: If T'2 is weakly vital for T2 then T'1 and T2 are weakly-consistent atomic.

Proof: This theorem is proved using LTL properties. The proof of this theorem provided in Appendix (See Appendix A for Weak Vitality and Appendix B for Weak-Consistent Atomicity).

Corollary 2.1: If T'1 and T'2 are weakly vital for T1 and T2 respectively, T1 and T2 are weakly consistent atomic.

Proof: By theorem 2, since T'2 is weakly vital for T2, then T'1 and T2 are weakly consistent atomic. Again by the theorem 2, since T'1 is weakly vital for T1 then T1and T2 are weakly consistent atomic.

An example of these properties is invoice notification acknowledgement must be sent less than two hours after the invoice notification activity has completed.

Since invoice notification is at least weakly vital for payment activity, the payment and invoice notification acknowledgements are weakly consistent atomic (because they are linked by a constraint from a business rule). This means that as soon as the temporal condition fails, the payment must be aborted.

Theorem 3: Assuming that T'1 and T'2 are sub-transactions of T1 and T2 respectively, and T'1 and T'2 are strongly-consistent atomic transactions (with cond1 and cond2), if T'2 is strongly vital for T2 then T1' and T2 are strongly-consistent atomics (with cond1 and cond2).

Proof: Like Theorem 2, this theorem can also be proved using LTL properties. The proof of this theorem is not discussed, because the techniques to proof both the theorems (theorem 2 and theorem 3) are identical. Thus, we have provided the proof of theorem 2 in Appendix that also provides clear understanding of this theorem (See Appendix A and B for reference).

Corollary 3.1: If T'1 and T'2 are strongly vital for T1 and T2 respectively, then T1 and T2 are strongly-consistent atomic.

A sample propagation of strong consistent atomicity is explained in the following. We consider the following conditions:

- goods atomicity [Papazoglou2006] property enforces a strong consistent atomicity between two activities namely, "change account activity" and "loading activity"

- the “change account activity” is strongly vital for “payment”
- the “loading activity” is strongly vital for “shipment”

In case of the above conditions, the “payment” and “shipment” strongly inherit the good atomicity property. More explicitly, if payment is undone because of an independent or external event (e.g. cancel the order), then shipment must be undone too, and the activities “change account” and “loading activity” must be consistently failed.

8.3 Preliminary Formal Framework for BTs

The contractual primitive (explained in section 6.2) involves SLA that is composed of policies which denote a set of constraints (business transaction model demonstrates the relations between constraint, policy and SLA). Business rule is an important constraint type that can be declarative as well as imperative. The declarative rules concentrate on supporting the definition of independent rules and ensure that the rules specify *what* should happen, in contrast, imperative rules ensure that the rules specify *how* should happen[Taylor2007] . One of the main problems with the declarative rules is that it is not always possible to exactly enforce them during the execution. Two critical reasons are the global and **omniscient** declaration and instability of transactional statements.

The former reason is concerned with the public contract principle that enables abstracting from several local and orchestrated points of view, in order to get a global and (public) omniscient view. For instance, a role may require that when another role receives a message then a third role starts an activity. However, it is still unclear whether the message level could enable this execution enforcement.

The latter reason “instability of transaction statements” appears when a future condition has to be enforced without any further future execution details, e.g. the business rule "After business process has been terminated, if T has been committed, then T' must have been committed". One of its sub-conditions is "if T has been committed then in the future T' will be committed too”. However, no execution enforcement can satisfy this sub-condition since the future transactional statement of T' cannot be controlled by anyone.

Thus, it is only possible to implicitly restrict such allowed cases with executable (imperative) business rules by leveraging some declarative business rules such as "T must not commit until T' has committed". There is often a gap between declarative global specifications and imperative and local specifications. We propose a preliminary model for imperative business transactions that could deal with these issues. We propose to use StAC to model transactional aspects and B method to deal with the refinement of specifications from the global to the local and more detailed point of views. The latter proposal is inspired by the work in [Rychkova2008].

8.3.1 StAC: A Formal Language for Advanced BTs

Many techniques dealing with advanced business transactions have been proposed in the literature. [Butler2004] provided the StAC language for modeling business processes that supports a compensation mechanism. The authors defined the compensation as an action used to recover from a failure or deal with a change within the plan, especially when a rollback of the process is not possible. StAC is similar to a process algebraic language such as CSP presented in [Hoare1985] but has additional operators dealing with compensation and exception handling.

There exist other frameworks too (e.g. BizTalk [Satish2009], BPEL4WS [Curbera2003], BPBeans [Chessell2001]) but they are not at a formal level. Hence, we do not consider them in this section.

The CSP Language

Communicating Sequential Processes (CSP) [Hoare1985] is a formal language for interaction pattern descriptions in concurrent systems. Many researchers have used the CSP language to build a long-time transaction model. This language offers many operators which are able to define transactions and especially the compensation transactions. Using CSP language for Business Process Modelling, a process can be an atomic action, a sequential composition of two processes (P1; P2), a parallel composition of two processes (P1 || P2), a choice between two processes (P1 □ P2), a normal termination (SKIP), a process throwing an interrupt (THROW), a process leading to an interrupt (YIELD), a process handling an interrupt (P1 ▷ P2) or a block of transactions ([P1 P2]).

To summarize, the CSP operators are illustrated in Figure 7

.

P, Q :: =	*A*	(atomic action)
	P ; Q	(sequential composition)
	P □ Q	(choice)
	P ‖ Q	(parallel composition)
	SKIP	(normal termination)
	THROW	(throw an interrupt)
	YIELD	(yield to an interrupt)
	P ▷ Q	(interrupt handler)
	[PP]	(transaction block)

Figure 7 The CSP operator [Hoare1985]

The StAC Language

Structured Activity Compensation (StAC) [Ferreira2000] is a business process modelling language that also includes operators for modelling compensation processes. StAC provides a precise interpretation for the compensation mechanisms that are integrated into parallel executions, process hierarchies and exceptions, as illustrated in Figure 8. In addition to compensations and exceptions, StAC also supports several operators to combine sequential and concurrent (parallel) processes, which allow for coordinating basic activities. Moreover, it provides specific operators to compensate activities as if they have never existed.

Process :: =	*A*	(activity level)
	skip	(skip)
	b & P	(condition)
	call (N)	(call named process)
	P \ S	(hiding)
	P ; Q	(sequence)
	$P \underset{X}{\|\|} Q$	(parallel)
	P□ Q	(external choice)
	P⊓ Q	(internal choice)
	P{Q}R	(attempt block)
	⊙	(early termination of attempt)
	P ÷ Q	(compensation pair)
	☒	(reverse)
	☑	(accept)
	[*P*]	(compensation scoping)

Figure 8 StAC operator [Butler2004]

For brevity reason we focus only on the compensation operator. This operator is represented by the symbol "÷". The primitive of compensation pair is written "P÷Q", where P is the primary process and Q is its compensation. When (After) P runs (completes) successfully, Q is stored and will eventually be invoked in case any failure (or abortion) occurs in the whole transaction.

For example, in an online book ordering process, several compensation pairs can be defined and can be invoked in case of a book ordering failure. The choice of a book can be modelled by an operation "AddBook" that adds the book in the basket and decreases the number of available books in the database. This operation can be compensated by the operation "ReturnBook" that cleans the basket and increases the number of the available books in the database.

Bookstore = $\|\, c \in C.\ Client(c)$
Client(c) = ***Arrive****(c) ; ChooseBooks(c) ; ((****Quit****(c) ;* $\boxtimes$*)* $\square$ *Pay(c)) ;* $\odot$ *;* ***Exit*** *(c)*
ChooseBooks(c) = ***Checkout*** *(c)* $\square$ *(ChoseBooks(c) ; ChooseBooks(c))*
ChhoseBooks(c) = $\square$ *b* $\in$ *B [(****AddBook****(c,b)* ÷ ***ReturnBook****(c,b)); **overBudget**(c)* $\rightarrow$ $\boxtimes$*]*
Pay(c) = ***ProcessCard****(c) ;* $\neg$ ***accepted****(c)* $\rightarrow$ $\boxtimes$

Figure 9 Example of StAC [Butler2004]

StACi: an operational semantics for StAC

StACi is a variant of StAC. In StACi, a process can have several simultaneous compensation tasks associated with it. It can also decide which task the compensation activities are attached to. Each individual compensation task can be reversed or accepted. In this approach, the scoping of compensation is hierarchical and each scope has a single implicit compensation task.

Moreover, to distinguish different compensation tasks, the operators that deal with compensation tasks are indexed by the compensation task indexes to which they apply.

Process :: =	A	(activity level)
	$skip$	(skip)
	$b \,\&\, P$	(conditional)
	$call\ (N)$	(named process call)
	$P \setminus S$	(hiding)
	$P\ ;\ Q$	(sequence)
	$P \underset{X}{\|} Q$	(parallel)
	$P \sqcap Q$	(internal choice)
	$P \square Q$	(external choice)
	$\odot$	(early termination of attempt)
	$P\{Q\}_v R$	(attempt block)
	$early$	(prematurely terminated process)
	$[P]_v$	(protection block)
	$New\ (i).P_i$	(create new compensation task)
	$\boxtimes_i$	(indexed reverse)
	$\boxdot_i$	(indexed accept)
	$\uparrow_i P$	(push).
	$J \triangleright I$	(merge)

Figure 10 StACi operators [Butler2004]

Several StACi operators stem from the StAC language. The specificity of StACi concerns only the compensation operators and early termination.

8.3.2 From Global Knowledge to Local Knowledge

The public contract principle enables abstracting from several local and orchestrated points of view, in order to get a global and (public) omniscient view. To the best of our knowledge, only the BPEL local language has been translated into StACi [Butler2005]. There is still no formal translation of global knowledge business transaction language in StACi. In the following, we will describe a preliminary model using StAC [Butler2005] and B method [Raymond1996]. B method is a well known formal refinement program method using theorem provided via the platform RODIN [Damchoom2008]. It seems that it fits with the refinement from a global to a local view, similarly to the work of [Rychkova2008] with Z notation. On the contrary to Z notation, B gets some special and sophisticated theorem provers [Damchoom2008]. This encourages us to apply it for our model.

Activities

The main difference between a controlled and local view and a global view of a system is the introduction of roles that are responsible for a specific activity. An activity is assigned with a name and roles that an activity is responsible to perform. A role can trigger its activity only if it knows that the guard is enabled. Thus, even if the guard of triggering may range on the history knowledge (as in certain business rules) of the whole global variables, in reality it will be assessed only on the history knowledge of the corresponding role.

Messages

Messages can be example of activities. In first step, we consider the communications are synchronous. More precisely, the only "atomic" message activity will be "message.messagename.sender.receiver", which means that the "receiver" has received the message from the "sender".

Latter in the refinement, message could be considered asynchronous especially while the granularity of time constraints is comparable to the time delaying of asynchronous message.

Normal Flow

We suppose that designers define a traditional execution structure using the XML language.

```
<sequence>
        Ordering;
        Manufacturing
        Planning
        <parallel>
                Payment
                <sequence>
                        Logistic
                        Shipment
                <endsequence>
        <endparallel>
<Endsequence>
```

Following [Butler2005], we first model this normal execution flow with StAC and B method and then leverage the event and fault handlers to model the deviated transactional executions.

NormalExecution $\widehat{=}$
Orderring ;
manufacturing ‖ planning;

payment ‖ (logistics ; shipment)

Note that activities are not marked with a role, because activity can range over several and imprecise responsible roles at this level of abstraction. In the following, as soon as there is no specification of roles, it will be interpreted as an undetermined (or an abstraction) set of responsible roles. Latter in the refinement, the role will have to be determined. An example of a detailed and refined sub-transaction is the following ordering:

Ordering ≙
Message (order command, buyer, Order Management);
Assign (business process ID, OM); $\downarrow_{ordering}$ Fail ID;
message(check order, order management, SD);
checkorder(SD);
message (check order confirmed, SD, OM) □ message (check order fail, SD, OM);
message (check customer history, OM, CRM);
message (check customer history, CRM, Bank);
Checkhistory (Bank);
Message (History ok, Bank, CRM) □ message (bad history, bank, CRM);
…

At this stage, the activities are assigned to respective responsible roles. The expressive power of StACi enables to model flexible notions of compensation for whatever case handling. In this example, the compensation process has only been added to the execution flow using the action “fail ID”. This compensative action has been memorized in the corresponding "ordering" transaction**.** In order to deal with deviated transactional flows we exploit the idea of event catcher.

Event catcher

As used in [Butler2005], an event handler is modelled using a Loop, where

Loop (P) = X where X = *skip* □ (P ; X)

This loop will allow the modelling of "catching" an event. Given a guard G, we can trigger an action A when this guard is enabled with the following notation:

Loop (G ⇒ A)

We define an event catcher by

EC (G, A) = Loop (G ⇒ A)

Then in order to model all the events we can use the parallel symbol. For instance, the model of catching “check order fail” and “customer history is bad” and then to quit the normal execution flow is the following syntax:

EC ≙ EC ((message check order fail) ε global History, ⊙) ‖ EC ((message history bad ε global History), o)

This event will interrupt the normal flow according to the StAC structure. To be noticed, the event catcher are also not assigned with responsible roles until the detailed refinement where it should be determined whether the system needs an external neutral role as coordinator, monitoring system, or

orchestrator, or if the business roles ensure (consider) themselves as some parts of the event "catching".

Fault Handler

A fault handler is just a (event) catcher of case:

> FH = $\hat{=}$ FH ((message check order fail) ε global History, message (order failed, OM, buyer)) ‖ $reverse_{ordering} \wedge$ FH ((message History is bad) ε global History, message (order failed, OM, buyer)) ‖ $reverse_{ordering}$ …..;

where, FH (.., ...) has the similar semantics to the one of EC. Again, note that some activity will have to be distributed among roles in the refinement phase later.

The complete program structure

Finally, the complete program structure would be,

> Skip {Normalflow ‖ EC} FH

meaning that, as soon as an event is caught, the system will be interrupted and then deviated following the fault handler part of the execution.

The payment atomicity

As a conclusion, we would like to explain that operational notions such as the good payment atomicity [Papazoglou2006] can simply be modelled using the B method (similarly to [Rychkova2008]). The main contribution of this approach is to provide a formal support for explaining the business transactions in a process at a high level of abstraction similar to the semi formal work provided by [Damchoom2008]. This model is "state oriented" rather than activity oriented. Thus, while annotation of activity is not enough precise or would lead to a non consensual annotation, there is a need to model the activity using only consensual "business states". We will define the meaning of an atomic payment at a high level using the business consensual items such as account or price, as described in the following sample detailed payment:

> payment $\hat{=}$ new(payment);
> message("invoice notification", financial unit, buyer);
> message(starting message, buyer, bank);
> (transaction, bank) ;
> message(transaction confirmed, bank, buyer, seller) □ transaction failed;

In order to formally elicit the notion of atomic payment, we need to publicly elicit the accounts of both the buyer and seller:

> *(transaction, bank)* $\hat{=}$ *AnyPrice.Price' Account.currrent.Buyer := Account.current.Buyer - Price* ‖ *Account.current.Seller := Account.current.Seller + Price'.*

In order to manage the case while Price and Price' can be not equal bank transfer error, we need to ensure the payment atomicity by adding a catcher in the event handler set, in order to catch the error raised when cond1 has failed :

> EC(*Account.starting.buyer +Account.starting.Seller* ≠ *Account.current.buyer + Account.current.seller*, ⊙).

In case a fault is caught during the normal execution flow, we have to handle it (using a fault handler) since we have to enforce the cond2:

> FH (*Account.starting.buyer* + *Account.starting.Seller* ≠ *Account.current.buyer* + *Account.current.Seller, Account.current.Buyer* := *Account.starting.buyer* | *Account.current.Seller* := *Account. starting. Seller)*

This will enable a rollback of the transaction using the history variables.

In this section, we provided a formal definition for the notion of consistent-atomicity and vitality, as well as introduced some induction properties. As mentioned in [Ray2006] temporal dependencies constitute major concerns in advanced transactional models, yet do not seem to have been widely applied, for instance in describing business rules.

9 Summary and Outlook

In SNs, business processes are increasingly complex and integrated both within internal corporate business functions (e.g., manufacturing, design engineering, sales and marketing, and enterprise services) and across end-to-end processes. The trend will be to move from relatively stable, organization-specific applications to more dynamic, high-value ones where business process interactions and trends are examined closely to understand more accurately application needs and dynamics. In such environments there is a clear need for advanced business applications to coordinate multiple services into a multi-step business transaction. This requires that several service operations or processes attain transactional properties reflecting business semantics, which are to be treated as a single logical (atomic) unit of work that can be performed as part of a business transaction.

The recent advent of WS technologies and open standards such as WSDL, BPEL, and WS-Policy has helped to evolve our thinking about how distributed applications can connect and work together. However, none of these core WS specifications were designed to provide mechanisms by themselves for describing how individual services can be connected to create dependable business critical solutions with the appropriate level of complexity that can guarantee absolute completion and accuracy of interacting business processes.

Indeed, there is a need for explicitly managing fine grained tenets of SBAs such as business data, events, operations, process fragments, local and aggregated QoS and associated KPIs, and so on, to guarantee a continuous and cohesive information flow, correlation of end-to-end process properties, termination and accuracy of interacting business processes that is driven by application control(-integration) logic.

The above considerations give rise to a multi-modal transaction processing scheme by enabling reliable business transactions that span from front-end SBAs to back-end system-level transaction support and beyond to organizations that are part of a SN. Thus, the need for application-level management technologies that can ensure highly reliable application (not only system)-level transactions and end-user performance via rapid problem diagnosis and resolution - while at the same time support change and capacity planning - is paramount. In particular, we argue in favour of the need to provide rich features and QoS similar to that offered by transaction processing monitors but at the level of application management and application control logic, giving rise to the concept of a business transaction.

The business transaction then becomes the framework for expressing detailed operational business semantics. Conventional approaches to business transactions, such as Open EDI, UMM and ebXML, focus only on the documents exchanged between partners, rather than coupling their application interfaces, which inevitably differ.

This deliverable targeted the concept of a business transaction and explored how process fragments, and particularly transactional process fragments, fit in the context of a running scenario which possesses transaction properties. Conventional (ACID) and unconventional (application-based) types of atomicity were introduced, including contract and delivery atomicity, in the frame of a business transaction model. The transaction model provides a comprehensive set of concepts and several standard primitives and conventions that can be utilized to develop complex SBAs involving transactional process fragments.

This main goal of this deliverable was to introduce an initial version of a BTL. In particular, the initial BTL specification presented in this deliverable defines a transaction model and mechanisms for transactional interoperability between end-to-end service constellations in SNs and provides a means to define and enforce transactional QoSs into SBAs. Both the model and language are firmly grounded on a requirements analysis, involving an in-depth literature survey while taking into account requirements stemming from other work packages in the S-Cube framework. The approach taken mimics business operational semantics and does not depend upon underlying technical protocols and implementations. Mission-critical composite applications will differ from the smaller-scale composite applications built using BPEL extensions (see JRA-2.2). The mission-critical composite applications will be built on the established foundation of SNs, enterprise systems, WS standards and platform.

The research results presented in this deliverable are core results in nature. We plan for improvements and extensions in multiple directions.

Firstly, the BTL merely addresses design-time transactional desiderata. Runtime support for business transactions, including the mapping of the BTL to specific runtime languages and standards, will be studied in the upcoming deliverable, PO-JRA-2.1.4. This thus implies that the BTL will be iteratively refined and extended to more effectively cope with coordination and recovery. Notably, this will include elaborating the mapping from transactional process fragments to service compositions addressed in JRA-2.2, using and possibly extending on widely accepted standards such as WS-Policy, WS-Security, WS-Transaction and BPEL (4Chor).

While we have introduced an initial formalization of business transactions herein to ensure internal and construct validity, we clearly need to further strengthen our formal framework and the associated BTL iteratively.

The first challenge would be to enhance the business transaction notions and study new transactional dependencies, and then compare them to the inference techniques of the business rules engines. Similarly, we plan to investigate possible existing links between declarative business transactions and imperative execution. For instance, how to produce imperative execution or to select them in a bottom up approach? The second challenge is to pursue a model from global to local view in order to adapt the B method to our case. In our view, the definition of a business activity must be formally specified based on simple and very consensual business concepts. We also plan to verify our approach with real problems of flexibility or non-consensual business notions. Finally, since some recent work has started to deal with other alternatives of the compensation mechanism as the Tentative Hold Protocol [Roberts2001], we plan to assess the compensation mechanism using a cost model of the execution of the database system, and compare this cost against the new methods. This last issue would deal with the performance of the deployed transaction system level in order to ease the tasks of the BT designers by providing an adaptive solution that is able to (semi-)automatically choose an appropriate strategy among many compensation- or locking-based ones on system level.

The above challenges will be achieved in deliverable JRA-2.1.5 that will revolve around formal verification. The external validation of the business transaction model and associated BTL with use cases and scenarios were not part of this deliverable either; instead we plan to work on that in the context of PO-JRA-2.1.6.

Appendix A: Proof of Weak Vitality

Theorem: Assume that T' is a weakly vital sub-transaction of T, and T" is a weakly vital sub-transaction of T', then using the LTL semantics we can easily show that T" is weakly vital sub-transaction of T.

Proof: We assume T' is weakly vital for T

- $AG(\bar{commit} - T \Rightarrow \bar{commit} - T')$.
- $AG(aborted - T' \Rightarrow F(abort - T))$.
- $AG(Compensated - T' \Rightarrow F(aborted - T \vee Compensated - T))$

And T'' is weakly vital for T'

- $AG(commit(T') \Rightarrow (\bar{commit}(T')))$.
- $AG(aborted(T'') \Rightarrow F(aborted(T')))$
- $AG(Compensated(T'') \Rightarrow F(aborted(T') \vee Compensated(T')))$

Then using deduction system from[Gabbay1980]*:*

$$\frac{\vdash (comT \Rightarrow comT' \wedge comT' \Rightarrow comT'') \Rightarrow (comT \Rightarrow comT'')}{\vdash G[(comT \Rightarrow comT' \wedge comT' \Rightarrow comT'') \Rightarrow (comT \Rightarrow comT'')]}$$
$$\frac{\vdash G[(comT \Rightarrow comT' \wedge comT' \Rightarrow comT'')] \Rightarrow G[comT \Rightarrow comT'']}{\vdash G[comT \Rightarrow comT'] \wedge G[comT' \Rightarrow comT''] \Rightarrow G[comT \Rightarrow comT'']}$$
$$\frac{\vdash A(G[comT \Rightarrow comT'] \wedge G[comT' \Rightarrow comT''] \Rightarrow G[comT \Rightarrow comT''])}{\vdash A(G[comT \Rightarrow comT'] \wedge G[comT' \Rightarrow comT'']) \Rightarrow AG[comT \Rightarrow comT'']}$$
$$\frac{\vdash AG[comT \Rightarrow comT'] \wedge AG[comT' \Rightarrow comT''] \Rightarrow AG[comT \Rightarrow comT'']}{AG[comT \Rightarrow comT'], AG[comT' \Rightarrow comT''] \vdash AG[comT \Rightarrow comT'']}$$

$$\frac{\vdash ((((\bar{abT'}) \Rightarrow \exists j, (abT)^{j})) \wedge (\forall k, (abT)''^{k} \Rightarrow \exists l, ((\bar{abT'})^{l+k}))) \Rightarrow ((abT'') \Rightarrow \exists z((abT)^{z}))}{\vdash ((abT' \Rightarrow F(abT')) \wedge G(abT'' \Rightarrow F(abT'))) \Rightarrow (abT'' \Rightarrow F(abT'))}$$
$$\frac{\vdash G[((abT' \Rightarrow F(abT)) \wedge G(abT'' \Rightarrow F(abT'))) \Rightarrow (abT'' \Rightarrow F(abT))]}{\vdash G[(abT' \Rightarrow F(abT)) \wedge G(abT'' \Rightarrow F(abT'))] \Rightarrow G[abT'' \Rightarrow F(abT)]}$$
$$\frac{\vdash G[abT' \Rightarrow F(abT)] \wedge GG[abT'' \Rightarrow F(abT')] \Rightarrow G[abT'' \Rightarrow F(abT)]}{\vdash A(G[abT' \Rightarrow F(abT)] \wedge G[abT'' \Rightarrow F(abT')] \Rightarrow G[abT'' \Rightarrow F(abT)])}$$
$$\frac{\vdash A(G[abT' \Rightarrow F(abT)] \wedge G[abT'' \Rightarrow F(abT')]) \Rightarrow AG[abT'' \Rightarrow F(abT)]}{\vdash AG[abT' \Rightarrow F(abT)] \wedge AG[abT'' \Rightarrow F(abT')] \Rightarrow AG[abT'' \Rightarrow F(abT)]}$$
$$\frac{}{AG[abT' \Rightarrow F(abT)], AG[abT'' \Rightarrow F(abT')] \vdash AG[abT'' \Rightarrow F(abT)]}$$

$$\frac{\vdash (((\forall i, (cpT')^{i} \Rightarrow \exists j, (cpT \vee \bar{abT})^{j+i})) \wedge ((cpT'') \Rightarrow \exists l, ((cpT' \vee abT')^{l})) \wedge (\forall m, (abT')^{m} \Rightarrow \exists n, (abT)^{m+n}) \Rightarrow ((cpT'') \Rightarrow \exists z((cpT \vee abT)^{z}))}{\vdash G[cpT' \Rightarrow F(abT \vee cpT)] \wedge [cpT'' \Rightarrow F(abT' \vee cpT')] \wedge G[abT' \Rightarrow F(abT)] \Rightarrow [abT \Rightarrow F(cpT \vee abT)]}$$
$$\frac{\vdash G(G[cpT' \Rightarrow F(abT \vee cpT)] \wedge [cpT'' \Rightarrow F(abT' \vee cpT')] \wedge G[abT' \Rightarrow F(abT)] \Rightarrow [cpT'' \Rightarrow F(cpT \vee abT)])}{\vdash GG[cpT' \Rightarrow F(abT \vee cpT)] \wedge G[cpT'' \Rightarrow F(abT' \vee cpT')] \wedge GG[abT' \Rightarrow F(abT)] \Rightarrow G[cpT'' \Rightarrow F(cpT \vee abT)]}$$
$$\frac{\vdash A(G[cpT' \Rightarrow F(abT \vee cpT)] \wedge G[cpT'' \Rightarrow F(abT' \vee cpT')] \wedge G[abT' \Rightarrow F(abT)] \Rightarrow G[cpT'' \Rightarrow F(cpT \vee abT)])}{\vdash A(G[cpT' \Rightarrow F(abT \vee cpT)] \wedge G[cpT'' \Rightarrow F(abT' \vee cpT')] \wedge G[abT' \Rightarrow F(abT)]) \Rightarrow AG[cpT'' \Rightarrow F(cpT \vee abT)]}$$
$$\frac{\vdash (AG[cpT' \Rightarrow F(abT \vee cpT)] \wedge AG[cpT'' \Rightarrow F(abT' \vee cpT')] \wedge AG[abT' \Rightarrow F(abT)]) \Rightarrow AG[cpT'' \Rightarrow F(cpT \vee abT)]}{AG[cpT' \Rightarrow F(abT \vee cpT)], AG[cpT'' \Rightarrow F(abT' \vee cpT')], AG[abT' \Rightarrow F(abT)] \vdash AG[cpT'' \Rightarrow F(cpT \vee abT)]}$$

The strong vitality can be proved in an identical way.

Appendix B: Proof of Weak-Consistent Atomicity

Theorem: Assume that T'1 and T'2 are sub-transactions of T1 and T2 respectively, and T'1 and T'2 are weakly-consistent atomic transactions (with cond); if T'2 is weakly vital for T2, then T'1 and T2 are weakly-consistent atomic.

Proof: Assume the weak consistent atomicity for T'' and T'

- $AG(commit-T' \wedge commit-T'' \Rightarrow Cond)$
- $AG(\neg(cond) \Rightarrow F([abort-T' \vee Compensated(T')] \wedge [abort-T'' \vee Compensated-T'']))$

Meanwhile, consider the weak vitality of T' sub-transaction of T.

- $AG(commit-T \Rightarrow commit-T')$.
- $AG(aborted-T' \Rightarrow F(abort-T))$.
- $AG(Compensated-T' \Rightarrow F(aborted-T \vee Compensated-T))$

Then,

$$\frac{\frac{\frac{\frac{\frac{\frac{\vdash [(comT \Rightarrow comT' \wedge ((comT' \wedge comT'') \Rightarrow (cond))] \Rightarrow (comT \wedge comT'' \Rightarrow cond)}{\vdash G[[comT \Rightarrow comT' \wedge ((comT' \wedge comT'') \Rightarrow (cond))] \Rightarrow (comT \wedge comT'' \Rightarrow cond)]}}{\vdash G[comT \Rightarrow comT'] \wedge G[(comT' \wedge comT'') \Rightarrow (cond)] \Rightarrow G(comT \wedge comT'' \Rightarrow cond)}}{\vdash A(G[comT \Rightarrow comT'] \wedge G[(comT' \wedge comT'') \Rightarrow (cond)] \Rightarrow G(comT \wedge comT'' \Rightarrow cond))}}{\vdash AG[comT \Rightarrow comT'] \wedge AG[(comT' \wedge comT'') \Rightarrow (cond)] \Rightarrow AG(comT \wedge comT'' \Rightarrow cond)}}{AG[comT \Rightarrow comT'], AG[(comT' \wedge comT'') \Rightarrow (cond)] \vdash AG(comT \wedge comT'' \Rightarrow cond)}$$

$$\frac{\frac{\frac{\frac{\vdash (\forall i,(cpT')^i \Rightarrow \exists j,(abT \vee cpT)^{i+j}) \wedge (\forall k,(abT')^k \Rightarrow \exists l,(abT)^{l*k}) \wedge (\neg(cond) \Rightarrow \exists m,([abT' \vee cpT'] \wedge [abT'' \vee cpT''])^m) \Rightarrow (\neg(cond) \Rightarrow \exists n,([abT \vee cpT] \wedge [abT'' \vee cpT''])^n)}{\vdash G(cpT' \Rightarrow F(abT \vee cpT)) \wedge G(abT' \Rightarrow F(abT)) \wedge (\neg(cond) \Rightarrow F([abT' \vee cpT'] \wedge [abT'' \vee cpT''])) \Rightarrow (\neg(cond) \Rightarrow F([abT \vee Cp(T)] \wedge [abT'' \vee cpT'']))}}{\vdash G[G(cpT' \Rightarrow F(abT \vee cpT)) \wedge G(abT' \Rightarrow F(abT)) \wedge (\neg(cond) \Rightarrow F([abT' \vee cpT'] \wedge [abT'' \vee cpT''])) \Rightarrow (\neg(cond) \Rightarrow F([abT \vee Cp(T)] \wedge [abT'' \vee cpT'']))]}}{\vdash G(cpT' \Rightarrow F(abT \vee cpT)) \wedge G(abT' \Rightarrow F(abT)) \wedge G(\neg(cond) \Rightarrow F([abT' \vee cpT'] \wedge [abT'' \vee cpT''])) \Rightarrow G(\neg(cond) \Rightarrow F([abT \vee Cp(T)] \wedge [abT'' \vee cpT'']))}}{\vdash AG(cpT' \Rightarrow F(abT \vee cpT)), AG(abT' \Rightarrow F(abT)), AG(\neg(cond) \Rightarrow F([abT' \vee cpT'] \wedge AG[abT'' \vee cpT''])) \vdash AG(\neg(cond) \Rightarrow F([abT \vee cp(T)] \wedge [abT'' \vee cpT'']))}$$

The proof of strong-consistent atomicity is identical to the proof of weak-consistent atomicity

References

[Aissi2002] S. Aissi, P. Malu, and K. Srinivasan. E-Business process modelling: the next big step. IEEE Computer, Volume 35, Issue 5, pp. 55-62, 2002.

[Alonso1996] G. Alonso et al., "Advanced Transaction Models in Workflow Contexts," Proc. Int'l. Conf. Data Eng., Feb. 1996.

[Amit2001] R. Amit and C. Zott. Value Creation in eBusiness, Strategic Management Journal, Volume 22, pp. 493-520, 2001.

[Browne2009] P. Browne. *Jboss Drools Business Rule.* s.l. : Packt Publishing, 2009.

[Butler2005] Michael J. Butler, Carla Ferreira, and Ng. Muan Yong. "Precise Modelling of Compensating Business Transactions and its application to BPEL". J.UCS, Issue 5, Vol. 11, pp. 712-743, 2005.

[Butler2004] Michael J. Butler, C.A.R Hoare and Carla Ferreira. "A trace semantics for long-running transactions." In 25 Years Communicating Sequential, pp. 133–150,2004.

[Chessell2001] M. Chessell, D. Vines, C. Griffin, V. Green, and K. Warr. *Business process beans: System design and architecture document.* , Transaction Processing Design and New Technology Development Group, IBM UK Laboratories. Technical report, 2001.

[Clark2001] J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer. ebXML Business Process Specification Schema, Version 1.01, UN/CEFACT and OASIS Specification. Available at: http://www.ebxml.org/specs/ebBPSS.pdf.

[Curbera2003] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.1. *[Online] http://www-106.ibm.com/developerworks/ library/ws-bpel,*2003.

[Czarnecki2000]K. Czarnecki and U. Eisenecker. "Generative Programming: Methods, Tools and Applications", Addison-Wesley, 2000.

[Damchoom2008]K. Damchoom, M. J. Butler, and Jean-Raymond Abrial. "Modelling and Proof of a Tree-Structured File System in Event-B and Rodin." ICFEM, pp. 25-44, 2008.

[ebXML2001] ebXML Initiative. ebXML business process specification schema version 1.01. May 2001. [Online]. Available at: http://www.ebxml.orgspecsebBPSS.pdf

[Eliot1985] J. Eliot , B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing.* s.l. , MIT Press, p. 160, 1985.

[Emerson1990] A. E. Emerson. *Temporal and modal logic, Handbook of theoretical computer science.* [éd.] Jan van Leeuven. s.l. : Elsevier Sciences Publishers, pp. 995-1072. Vol. B :formal models and semantics, 1990.

[Ferreira2000] C. Ferreira, M. J. Butler. "A process compensation language. In." [éd.] LNCS. Integrated Formal Methods (IFM'2000), s.l. : Springer-Verlag, Vol. 1945, pp. 61 – 76, 2000.

[Gabbay1980] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. "On the temporal analysis of fairness". [éd.] ACM. POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 1980, pp. 163-173.

[Georg1995] Dimitrios Georgakopoulos, Mark F. Hornick, and Amit P. Sheth. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, Vol. 3, No. 2. (1995), pp. 119-153.

[Gray1993] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* s.l. : Morgan Kaufmann, 1993.

[Hantry2009] F. Hantry. *High level business queries.* university of Claude Bernard Lyon. Technical report, 2009.

[Harmsen1997] A. Harmsen, M. Ernst, and U. Twente. Situational Method Engineering. Moret Ernst & Young Management Consultants, 1997.

[Hoare1985] C.A.R Hoare. *Communicating Sequential Processes.* s.l. : Prentice-Hall, 1985.

[Hofman1994] W. J. A. Hofman. conceptual model of a Business Transaction Management System, Uitgeverij Tutein Nolthenius, PhD Thesis

[IBM2001] IBM. Web business event. *[Online] 01.ibm.com/software/integration/wbe/.*

[Kambil1997] A. Kambil and Eric van Heck. Reengineering the Dutch Flower Auctions: A Framework for Analyzing Exchange Organizations. Information Systems Research Volume 9, Number 1, March, pp. 1-19, 1997.

[Kim2002] H. Kim. Conceptual Modelling and Specification Generation for B2B Business Process based on ebXML. In: SIGMOD Record, Volume 31, 2002.

[Kopp2009] Oliver Kopp, Matthias Wieland, and Frank Leymann. "Towards Choreography Transactions", 2009.

[Kratz2008] B. Kratz and J. Vonk, P. Grefen. "A Survey on the History of Transaction Management: from Flat to Grid Transactions." Distributed and Parallel Databases, s.l. : Springer, Issue 3, Vol. 23, pp. 235-270,2008.

[Kratz2004] B. Kratz. Protocols for long running business transactions. Infolab technical report series, No. 17, Infolab, Tilburg University, 2004.

[Ma2008] Z. Ma and F. Leymann. A Lifecycle Model for Using Process Fragment in Business Process Modeling. BPDMS'08.

[Medjahed2003]B. Medjahed, B. Benatallah, A. Bouguettaya, A. Ngu, and A. Elmagarmid. Business-to-business interactions: issues and enabling technologies. VLDB J, Volume 12, Number 1, pp. 59-85, 2003.

[Molina1987] Hector Garcia-Molina and Kenneth Salem. "Sagas". SIGMOD Conference (1987), pp. pages 249-259, 1987.

[OpenEDI1995] ISO/IEC JTC1/SC30. The Open-EDI reference model. Committee Draft 14662, 1995.

[Papazoglou2006]M. P. Papazoglou and B. Kratz. A business-aware web services transaction model. In Dan, A, and Lamersdorf, W. (eds.) Proceedings of the 4th international conference on service oriented Computing (ICSOC 2006). Lecture Notes in Computer Science, Volume 4294 Springer, Berlin, pp 352-364.

[Papazoglou2003a]M. P. Papazoglou. Web Services and Business Transactions. World Wide Web: Internet and Web Information Systems, Volume 6, Number 1, pp. 49-91, 2003.

[Papazoglou2003b]M. P. Papazoglou and D. Georgeakopoulos. Service Oriented Computing. Communications of the ACM (2003) - 203.130.231.110, 2003.

[Ray2006] I. Ray and T. Xin. "Analysis of dependencies in advanced transaction models." Distributed and Parallel Databases, Issue 1, Vol. 20, pp. 5-27, 2006.

[Raymond1996] A. Jean-Raymond. *The B Book - Assigning Programs to Meanings.* s.l. : Cambridge University Press, 1996.

[Roberts2001] J. Roberts and K. Srinivasan. " Tentative Hold Protocol Part 1." White Paper, s.l. : W3C, november 2001. http://www.w3.org/TR/tenthold-1.

[Rosemann2007]M. Rosemann and W. M. van der Aalst. A configurable reference modelling language. Inf. Syst. 32, 1, March-2007.

[Rychkova2008]I. Rychkova. *Formal semantics for refinement verification of entreprise models.* Ecole polytechnique fédérale de Lausanne. PhD thesis, 2008.

[Satish2009] T. R. Satish. "BizTalk Integration Server Architecture." *Lecture Notes in Informatics, 2009.*

[Sheth1993] A. Sheth, M. Rusinkiewicz "On transactional workflows", IEEE Data Engineering Bulletin, 1993.

[Steinmetz2008]T. Steinmetz. Ein Event-Modell für WS-BPEL 2.0 und dessen Realisierung in Apache ODE. Diploma thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany (2008) (In German).

[Tygar1998] J. D. Tygar, "Atomicity in electronic commerce," *ACM-Mixed Media*, April 1998.

[Tylor2007] J. Tylor and N. Raden. *Smart (Enough)System: How to Deliver Competitive Advantage by Aotomating Hidden Decision.* Pearson Education Inc. ISBN-0-13-234-796-2. Prntice Hall, MA-2007.

[UMM2009] UN/CEFACT's Modeling Methodology (UMM): UMM Meta Model – Foundation Module Candidate for 2.0, 2009-10-01 http://umm-dev.org/umm-specification/

[Vonk2008] J. Vonk. *An Analysis of Contractual and Transactional Aspects of a Teleradiology Process*. Beta Working Papers; Eindhoven University of Technology. Vol263, 2008.

[Vonk2003] J. Vonk and Paul W. P. J.Grefen. "Cross-Organizational Transaction Support for E-Services in Virtual Enterprises." Distributed and Parallel Databases, Issue 2, Vol. 14, pp. 137-172, 2003.

[Yang2000] J. Yang and M. P. Papazoglou. Interoperation support for electronic business. Communication ACM, Volume 43, Number 6, pp. 39-47, 2000.