Coherence problem between Business Rules and Business Processes

Mario Lezoche

Roma Tre University
Ph.D. in Computer Science and Engineering

Coherence problem between Business Rules and Business Processes

Mario Lezoche

Coherence problem between Business Rules and Business Processes

A thesis presented by
Mario Lezoche
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

March 2008

Committee:
Prof. Michele Missikoff

Reviewers:
Prof. Leonardo Tininini
Prof. Selmin Nurcan

To my Mum.

Abstract

Business Process (BP) transformation is a key aspect of BP lifecycle. There are several reasons that may cause BP modifications. Among these, particularly important are the changes of the enterprise organization and operation strategies, which can be captured by business rules (BRs). This work focus on a BP-based organization that is regulated by a set of BRs: such BPs and BRs need to be globally consistent (and have to be maintained consistent after any changes). In this thesis is presented an ontological approach capable of representing BRs and BPs in a coherent way. Then, the objective is identifying all processes in the BP repository that are (or have become) inconsistent with the BRs and thus need to be changed to reestablish the overall consistency.

Acknowledgments

I would like to thank professor Michele Missikoff who guided me through the comprehension of the research way. My family and my friends who always supported my efforts in this period.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Business process (BP) modeling is opening a new phase in the development of enterprise software applications (ESA), thanks to the recent proposal of the Model Driven Architecture (MDA) approach, promoted by OMG [Sol00]. In essence, MDA proposes 3 levels of BP models:

1. computational independent models (CIM), where business experts provide a first (informal) description of a BP (e.g. by using EPC [dA99]);

2. platform independent models (PIM), where business experts work together with IT experts to build a procedural specification of the process in a rigorous way (e.g., by using BPMN [Whi04]), but leaving out low level technical details;

3. platform specific models (PSM), developed by IT experts, where all the technical details are introduced to achieve a complete specification, (e.g., by using BPEL4WS [IIIa]), ready to be executed by a suitable engine (e.g., ActiveBPEL1) [**?**], invoking the required services.

The OMG-MDA proposal has been mainly conceived to support a layered development of enterprise software applications, however for what concerns BP lifecycle, and in particular BP evolution does not propose any specific approach. In a dynamic enterprise, BPs need to be periodically revised and updated. Such BP transformation may be necessary for different reasons, for instance:

- poor functional performances (e.g., a process takes too much time or fails to fully achieve what expected);

- poor non-functional performances (e.g., security and privacy are not sufficiently guaranteed);
- new company policies (requiring the update of non conformant BPs).

In this work we address the latter issue, in a context where company policies are mainly expressed by business rules (BR).

Company policies are often represented in the form of business rules (BRs.) Business experts usually tend to formulate a BR in natural language. For instance, a BR can say: "all expenses greater than 10.000 euros require the approval of the Head of Unit". Another BR may involve the way business operations are performed, for instance: "the receiving of a quotation must precede the issuing of a purchase order". The latter BR can be synthesized by the expression: A PRECEDES B.

A challenging problem is to automatically identify the BPs that violate a BR and, possibly, to make the former evolving according to the latter.

In this work we present an ontological approach to BP modelling, proposing a method to verify if, given a BR, a process is consistent with it. To this end, we introduce the BPAL (Business Process Abstract Language) ontological framework and we show how the semantics of a BP can be modeled. The modeling primitives of BPAL have been derived from Business Process Modeling Notation (BPMN). The focus is, then, on the problem of a BR formulation in the context of a BPAL ontology and how to decide if a given BP is still consistent with the BR content or it should be modified. The proposal is positioned across the CIM and PIM levels of the MDA. The idea is to enhance intuitive modelling notations, such as BPMN, by associating to them a BP ontology system. The latter will provide a formal setting and the related semantic services.
Often, BRs are specified in natural language (NL) by business people and the intended meaning may not be sufficiently precise, therefore it is necessary to preliminarily understand what the BR content is, i.e., its semantics.
The focus of the first objective of the work is, therefore, on a method aimed at supporting business experts to disambiguate BRs that may be interpreted in different ways.
Once a BR has been disambiguated and reformulated to explicitly represent its intended meaning, we can address the primary objective of the work: **BP consistency**. Here, it is necessary to first identify the BPs that violate the

BR, and then to propose the updates necessary to make the former consistent with the latter.
BRs can represent a large variety of enterprise directives and constraints. Here we focus on a specific case, the procedural aspect of a BP, namely the sequence of activities, and BRs that indicate the correct order in which activities should be performed. Such BRs, having the form: A PRECEDES B, have been extensively analysed by Van der Aalst [6]. Here we address the same problem, but our solution is inherently different since we address the intensional level, i.e., BP schemas, while in [**?**] the focus is on the extensional level, i.e., BP instances.
The literature reports a large number of relevant results in the areas of process modeling, rules representation and management and, in particular, results addressing the analysis of activities sequencing. However, we wish to point out that we avoided these types of approaches, such as those based on temporal algebra (see the proposals based on the Allens work [All83]) or temporal logic [D.M91], and its variations (such as Event Calculus [RM86] and Situation Calculus [PF99]). Instead, here we approached the analysis of BP and the sequencing of their activities by using the notion of precedence.
The application of the mentioned methodologies resulted in a tool whose function is of controlling the coherence between BR and BP.
The rest of the work is organized as follows. The next chapter introduces the Business Process Modeling concept. Chapter 3 illustrates what are business rules and how they are used. Chapter 4 introduces BPAL to represent a BP and its specification. Chapter 5 shows a case study. Chapter 6 elaborates Business Rules impact onto Business Process Schema. Chapter 7 presents the developed application to control the coherence between business rule and business process. Finally Chapter 8 presents the conclusions and the appendices contain deepening on BPAL and the application code.

Chapter 2

Business Process Modeling

2.1 What is a Business Process

A model is a set of propositions or statements expressing relationship among constructs.

A business process, or business method, is a model composed by a collection of related, structured activities or tasks that produce a specific service or product (serve a particular goal) for a particular customer or customers.

There are three types of business processes:

1. Management processes, the processes that govern the operation of a system. Typical management processes include "Corporate Governance" and "Strategic Management".

2. Operational processes, processes that constitute the core business and create the primary value stream. Typical operational processes are Purchasing, Manufacturing, Marketing, and Sales.

3. Supporting processes, which support the core processes. Examples include Accounting, Recruitment, Technical support.

A business process begins with a customers need and ends with a customers need fulfillment. Process oriented organizations break down the barriers of structural departments and try to avoid functional silos.

A business process can be decomposed into several sub-processes, which have their own attributes, but also contribute to achieving the goal of the super-

process. The analysis of business processes typically includes the mapping of processes and sub-processes down to activity level.

Business Processes are designed to add value for the customer and should not include unnecessary activities. The outcome of a well designed business process is increased effectiveness (value for the customer) and increased efficiency (less costs for the company).

Business Processes can be modeled through a large number of methods and techniques. For instance, the Business Process Modeling Notation is a Business Process Modeling technique that can be used for drawing business processes in a workflow.

2.2 Business Process Modeling Languages

Today there is a renewed interest in business processes (BP), especially from a technological point of view. After a long time since the notion of a business process or a workflow was introduced [Coab], the recent advent of Service Oriented Architectures [OAS] gave a new momentum to this modelling practice. However, there is still a great difference between what is seen as BP modelling in the business world and in the IT world. In the former, processes are described mainly for human-to- human communication, for decision making in production processes, administrative processes, to understand their impact on the enterprise organization. In the IT world, processes are seen as a form of high level programming language (see for instance BPEL [**?**] [IIIb]), conceived to achieve a better use of web services (and, more generally, e-services), i.e., they represent an executable form of the application logics, as part of a complex software artefact. While the IT view of a BP is based on a prescriptive, executable form, aiming at a running software application (based on the service-oriented paradigm), the business view of a BP remains at an intuitive level, with models independent from the underlying technology. For its deep nature, a business model is highly intuitive, easy to be read and understood by humans, but at the same time exhibits a degree of ambiguity, incompleteness, bearing often internal contradictions. The dichotomy between business and IT approaches, that represents a serious problem in enterprise automation, has been addressed by the Model-Driven Architecture (MDA) approach proposed by the OMG. The MDA approach is structured in three main levels, with a progression of modelling activities that go from a business perspective (CIM: Computational Independent Modeling) to an application design perspective (PIM: Platform Independent Modeling) and, finally, reaches the implementa-

tion perspective (PSM: Platform Specific Modeling.) The MDA appears today as one of the most relevant proposals aimed at bridging the business world with the IT world. With its progressive modeling framework, it addresses one key problem laying in the different nature of business models and IT models. However, MDA is still in an early stage and currently it mainly represents a general reference framework with a limited actual impact. In parallel, there are intense research activities aimed at producing specific methods and tools, capable of providing a smooth and coherent evolution of models from the business perspective to the technical one. To improve the picture, one idea is to move upstream the moment where formal methods are introduced. However, it is difficult to impose new, formal languages to business people. Therefore, the solution is to inject formal semantics in existing business modelling methods. In this way, business people continue to use their modelling methods, having in parallel a formal support to such established solutions. Semantic annotations make use of ontologies (see OPAL [FD07].) However, current semantic technologies [GA04] are particularly suited to model static structures, e.g., the information part of a business model. The semantic enrichment of the behavioural part is still an open research issue, despite existing encouraging results [Coaa], [PR04], [DRF05]. In the literature, several languages for BP have been proposed. Such languages can be sketchily gathered in three large groups.

- **Descriptive languages**. They have been conceived within the business culture, but they lack of a systematic formalization necessary to be processed by an inference engine. In this group we find diagrammatic languages, such as EPC [**?**], IDEF [idea] [ideb] UML-Activity Diagram [grob], and BPMN [Groa] [JM05]. Also UML-Activity Diagram can be listed here, even if originally conceived for other purposes. The BP defined with these languages are mainly conceived for inter-human communication and are not directly executable by a computer.

- **Procedural languages**. They are fully executable by a computer but are not intuitive enough for being used by humans, and lack of a declarative semantics, necessary to be processed by a reasoning engine. Example of these languages are BPEL [**?**] and XPDL [JM05] [WFM]. Formal languages. They are based on rigorous mathematical foundations, but are difficult to be understood and are generally not accepted by business people. In this group we find languages such as PSL [Boc05] [SC00], Pi-Calculus [Mil99], Petri-Nets [Pet77].

- **Ontology-based process languages**, such as OWL-S [Coaa], WSDL-S [W3C], WISMO [DRF05], and Meteor-S [PR04]. This group of languages have a wider scope, aiming at modeling semantically rich processes in an ontological context, and have been conceived not directly connected to the business world. They are hence not considered here.

In this section we analyse the three above categories of languages, with the intent of proposing a solution that combines the key advantages of the three above groups: intuititivity and ease of use, rigorous mathematical basis, and possibility of execution.

Descriptive Methods

As anticipated, this group gathers the BP modelling methods that are mainly conceived for inter-humans communications. Here, among the most renewed methods, we have: UML, EPC, IDEF, and BPMN. The Unified Modeling Language1 (UML) is an OMG2 standard providing the specification for a graphical, general purpose, object-oriented modeling language. It defines 13 types of diagram, classified as structural, behavioural, and interaction oriented, according to what aspects of a system or a scenario they describe. These diagram types adopt a common meta-model, MOF3, defining their abstract syntax. UML has just an informal semantics associated and therefore it presents a high risk of ambiguities and contradictions in its models. Here there is not an explicit notion of a BP, however, the behavioural diagrams, and in particular the activity diagram and the sequence diagram, can be used to model a BP. Another important method is EPC: Event-driven Process Chains, a modeling technique for business processes modeling developed in the 1990s. The basic elements of EPC are: functions, corresponding to activities to be executed; events, describing the situation before and/or after a function execution; and logical connectors (and, or, xor) used when events determine the branching of the control flow. EPC has been proposed to model BP in the context of SAP R/3 application platform, then it has spread and there are popular modelling tools, such as Visio and ARIS, that support it (the latter is based on an extensions, eEPC, that integrate also the modelling of the enterprise data and organization.) EPC is very limited in term of its modelling scope, but at the same time it is simple and therefore its learning curve is rather steep. However, since neither the syntax nor the semantics are well defined, it is not possible to formally check the model for consistency and completeness. Furthermore the lack of a formal representation can generate ambiguities, in particular, when

models are exchanged between different tools. To solve this problem there are a number of proposals for a formalization, e.g., by using Petri Nets [**?**]. The ICAM Definition method (IDEF) is a set of standardized modeling techniques, initially proposed by an initiative of the United States Air Force. Among these techniques we mention IDEF0, the function modeling method, and IDEF3, the process description capture method. IDEF0 is a method designed to model the decisions, actions, and activities of an organization or a system. It allows activities and important relations between them to be represented in a non-temporal fashion. It does not support the complete specification of a process. The IDEF3 provides a mechanism for collecting and documenting processes, by capturing precedence and causality relations between situations and events. There are two IDEF3 description modes, process flow, capturing knowledge of "how things work" in an organization, and object-state transition network, summarizing allowable transitions an object may undergo throughout a particular process. Here we have a limited scope (e.g., actors are not modeled) and a fragmentation due to the need to use more than one diagram for the same BP. The Business Process Modeling Notation (BPMN) is the most recent standard notation proposed by OMG to design business processes. The main goal of this graphical notation is to be readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementation to the business people who will manage and monitor those processes. Since it is selected BPMN as the candidate BP modelling method, it will be specifically elaborated on in the Section 2.2.

Procedural Methods

This second group gathers the BP models endowed with a precise operational semantics, having therefore an associated execution engine (i.e., an interpreter.) The Business Process Execution Language for Web Services (BPEL4WS, or BPEL for short) is a de-facto standard for implementing processes based on web services. According to BPEL, processes can be described as executable processes, modeling the behavior of a participant in a business interaction, or as abstract processes, specifying the mutually visible message exchange among the parties involved in the protocol, without revealing their internal behavior. To obtain an executable BPEL process, modelers need to specify primitive and structured activities, execution ordering, messages exchanged, and fault and exception handling. Furthermore, a recent proposal, BPEL4People [IBM], extends BPEL4WS specification to describe scenarios where users are involved

in business processes. BPEL is a powerful and a widely adopted standard. Among its major drawbacks there are its inherent complexity, the verbosity of the XML encoding and the lack of a specific graphical representation. Such characteristics make it scarcely accepted by business people. The XML Process Definition Language (XPDL) is a WfMC standard for interchanging process models among process definition tools and workflow management systems. It provides the modeling constructs of BPMN and allows a BPMN process to be specified as an XML document. XPDL process models can be run on compliant execution engines, even if has been originally conceived as a process design and interchange format specifically for BPMN. It represents the linear form of the process definition based on BPMN graphics.

Formal Methods

In this section we present three formal methods conceived to model a business process in formal terms. Process Specification Language (PSL) is a general process ontology developed at the National Institute of Standards and Technology (NIST) for the description of basic manufacturing, engineering and business processes. In the manufacturing domain, PSLs objective is to serve as an interlingua for integrating several process-related applications (including production planning, process planning, workflow management and project management) throughout the manufacturing process life cycle [GM99a]. In view of our objective of associating a formal semantics to BPMN, PSL is a good candidate, hence we will present it in more details, after the description of the Petri Nets and Pi calculus, below. Petri Nets (also known as place/transition nets) is one of several mathematical representations of discrete distributed systems [Pet77]. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. A Petri net consists of places, transitions, and directed arcs. Arcs run between places and transitions. Places may contain any number of tokens. Transitions act on input tokens by a process known as firing. Execution of Petri nets is nondeterministic. This means two things: multiple transitions can be enabled at the same time, any one of which can fire, none are required to fire they fire at will, between time 0 and infinity, or not at all. Since firing is nondeterministic, Petri nets are well suited for modeling the concurrent behavior of distributed systems [Pet81]. In theoretical computer science, the Pi-Calculus (-calculus) is a process calculus originally developed by Robin Milner, Joachim Parrow and David Walker as a continuation of the body of work on the process calculus CCS (Calculus of Communicating Systems). The aim of the -calculus is to

be able to describe concurrent computations whose configuration may change during the computation. The -calculus belongs to the family of process calculi, mathematical formalisms for describing and analyzing properties of concurrent computation [Mil93]. In fact, the Pi- calculus, like the Pi-calculus, is so minimal that it does not contain primitives such as numbers, booleans, data structures, variables, functions, or even the usual flow control statements (such as if... then...else, while...). Central to the Pi- calculus is the notion of name. The simplicity of the calculus is due to the fact that names play a dual role as communication channels and variables. The process constructs available in the calculus are the following: concurrency, representing two processes or threads executed concurrently; communication, with input/output prefixing c¡y¿; replication, when a process creates a new copy. Although the minimality of the -calculus prevents us from writing programs in the normal sense, it is easy to extend the calculus. In particular, it is easy to define both control structures such as recursion, loops and sequential composition and datatypes such as first-order functions, truth values, lists and integers [Pic01]. Formal methods are not suited to be directly released to the end users. They are mainly conceived to study formal properties of certain categories of processes. The objective of our work is to select a formalism suited to be associated to BPMN. For its characteristics, we identified PSL, as a good candidate, therefore it will be presented in more detail in the next subsection.

PSL

The primary component of PSL is an ontology designed to represent the primitive concepts that, according to PSL, are adequate for describing basic manufacturing, engineering, and business processes [GM99a]. The challenge is to make the meaning of the terminology in the ontology explicit. Any intuitions that are implicit are a possible source of ambiguity and confusion. The PSL ontology provides a rigorous mathematical characterization of process information as well as precise expression of the basic logical properties of that information in the PSL language [SC00]. In providing the ontology the creators specify three notions: language, model theory and proof theory.

The Language

A language is a lexicon (a set of symbols) and a grammar (a specification of how these symbols can be combined to make well-formed formulas). The lexicon consists of logical symbols (such as boolean connectives and quantifiers)

and nonlogical symbols. For PSL, the nonlogical part of the lexicon consists of expressions (constants, function symbols, and predicates) chosen to represent the basic concepts in the ontology. Notably, these will include the 1-place predicates activity, activity- occurrence, object, and timepoint for the four primary kinds of entity in the basic PSL ontology, the function symbols beginof and endof that return the timepoints at which an activity begins and ends, respectively; there are also the 2-place predicates is-occurring-at, occurrence-of, exists-at, before, and participates-in, which express important relations between various elements of the ontology. The underlying grammar used for PSL is roughly based on the grammar of KIF [MRG92] (Knowledge Interchange Format). KIF is a formal language based on first-order logic developed for the exchange of knowledge among different computer programs with disparate representations. KIF provides the level of rigor necessary to unambiguously define concepts in the ontology, a necessary characteristic to exchange manufacturing process information using the PSL Ontology. Like KIF, PSL provides a rigorous BNF (Backus-Naur form) specification [GM99b]. The BNF provides a rigorous and precise recursive definition of the class of grammatically correct expressions of the PSL language. Furthermore, the rigorous specification eases the development of translators between PSL and other, similarly well-defined BP representation languages.

Model Theory

The model theory provides a rigorous, abstract mathematical characterization of the semantics, or meaning, of the language of PSL. This representation is typically a set with some additional structure (e.g., a partial ordering, lattice, or vector space). The model theory then defines meanings for the terminology and a notion of truth for sentences of the language in terms of this model. The objective is to identify each concept in the language with an element of some mathematical structure, such as lattices, linear orderings, and vector spaces. Given a model theory, the underlying mathematical structures becomes available as a basis for reasoning about the concepts intended by the terms of the PSL language and their logical relationships, so that the set of models constitutes the formal semantics of the ontology.

Proof Theory

The proof theory consists of three components: PSL Core, one or more foundational theories, and PSL extensions. PSL Core. The purpose of PSL-Core

is to axiomatize a set of intuitive semantic primitives that is adequate for describing the fundamental concepts of business processes. Consequently, this characterization of basic processes makes few assumptions about their nature beyond what is needed for describing those processes, and the Core is therefore rather weak in terms of logical expressiveness. In particular, PSL-Core is not strong enough to provide definitions of the many auxiliary notions that become necessary to describe all intuitions about business processes. The PSL Core is a set of axioms written in the basic language of PSL. The PSL Core axioms provide a syntactic representation of the PSL model theory, in that they are sound and complete with regard to the model theory. That is to say, every axiom is true in every model of the language of the theory, and every sentence of the language of PSL that is true in every model of PSL can be derived from the axioms. Because of this tight connection between the Core axioms and the model theory for PSL, the Core itself can be said to provide a semantics for the terms in the PSL language.

Foundational Theories

The purpose of PSL Core is to axiomatize a set of intuitive semantic primitives that is adequate for describing basic processes. Consequently, their characterization does not make many assumptions about their nature, beyond their elementary description. The advantage of this is that the account of processes given in PSL Core is relatively straightforward and uncontroversial. However, a corresponding liability is that the Core is rather weak in terms of pure logical strength. In particular, the theory is not strong enough to provide definitions of the many auxiliary notions that become needed to describe an increasingly broader range of processes in increasingly finer detail. For this reason, PSL includes one or more foundational theories. A foundational theory is a theory whose expressive power is sufficient for giving precise definitions of, or axiomatizations for, the primitive concepts of PSL. Moreover, in a foundational theory, one can define a substantial number of auxiliary terms, and prove important metatheoretical properties of the core and its extensions. For PSLs purposes, a suitable foundation is a modified and extended variation of the situation calculus. The reason for this is that the situation calculuss own primitives situation, action, fluent (roughly, proposition) are already highly compatible with the primitives of PSL. It is very natural to identify PSL primitives with, or define them in terms of, the primitives of the situation calculus. In addition, the situation calculus is also strong enough to define a wide variety of auxiliary notions and, with the addition of some set theory, it can be used as a basis for

proving basic metatheoretic results about the Core, and its extensions as well.

Extensions

The third component of PSL are the extensions. A PSL extension provides the resources to express information involving concepts that are not part of PSL Core. Extensions give PSL a clean, modular character. PSL Core is a relatively simple theory that is adequate for expressing a wide range of basic processes. However, more complex processes require expressive resources that exceed those of PSL Core. Rather than clutter PSL Core itself with every conceivable concept that might prove useful in describing one process or another, a variety of separate, modular extensions have been, and continue to be, developed that can be added to PSL Core as needed. In this way a user can tailor PSL precisely to suit his or her expressive needs. To define an extension, new constants and/or predicates are added to the basic PSL language, and, for each new linguistic item, one or more axioms are given that constrain its interpretation. In this way one provides a semantics for the new linguistic items. When combined with a foundational theory like the situation calculus, a distinction can be drawn between definitional and nondefinitional extensions. A definitional extension is an extension whose new linguistic items can be completely defined in terms of the foundational theory and PSL Core. Theoretically, then, definitional extensions add no new expressive power to PSL Core + foundational theory, and hence involve no new theoretical overhead. However, because definitions of many subtle notions can be quite involved, definitional extensions can prove extremely useful for describing complex processes in succinct manner. Nondefinitional extensions, called also core theories are extensions that involve at least one notion that cannot be defined in terms of PSL Core and the chosen foundational theory. All extensions within PSL must be consistent extensions of PSL-Core, and may be consistent extensions of other PSL extensions. However, not all extensions within PSL need be mutually consistent [SC00]. As anticipated, seen its rich articulation, PSL has been the first candidate to associate a formal semantics to BPMN. Next we illustrate BPMN and then we will return on this issue.

Business Process Modeling Notation

As anticipated, among the methods accepted by business people, we select BPMN for a number of reasons. Besides being the most recent one, outcome of several research activities, it allows a BP to be modeled with a single diagram

type, avoiding the fragmentation inherent in the UML and IDEF solutions. With respect to EPC it has a wider scope, being capable to capture a good number of business notions, from actors to messages.

The main constructs of BPMN

The main goal of BPMN is to standardize a business process modeling notation in order to provide a simple means of communicating process information among business users, customers, suppliers, and process implementers. It defines a diagrammatic notation and an intuitive semantics for business process modeling. In the following, the basic BPMN constructs are presented with the support of a practical example. We have categorized the BPMN constructs using four basic conceptual categories as it is shown in figure 2.1: actor, behavior, object, and co-action.

- **Actor**. This category refers to the constructs devoted to model any relevant entity that is able to activate or perform a process. The BPMN elements of this category are pool, and lane, representing more aggregate organization units and more specific ones, respectively. They allow for a partitioning of activities according to the performers.

- **Behavior**. This category refers to the constructs used to model the dynamic aspects of a domain. An activity is a generic term for work performed within a company. It can be atomic or non-atomic (compound). The types of activity are: task, process, and sub- process. A task is an atomic activity included within a process. Processes are either contained within a pool or unbounded. Sub-processes are composed activities included within a process. There are two types of sub-processes: embedded and independent. The independent are re-usable in different processes. A gateway is a modeling element used to represent the interaction of different sequence flows, as they diverge and converge within a process. When the sequence flows arrive at a gateway, they can be merged together on input and/or split apart on output. There are different types of gateway according to the types of behavior they define in the sequence flow. Decisions and branching are represented by the following gateways: OR-Split, exclusive-XOR, inclusive-OR, and complex. Merging is represented by the OR-Join gateway. Finally, forking is represented by the AND-Split gateway, and joining by the AND-Join gateway. Another construct referring to this modeling category is event. An event is something

that happens, like a trigger or a result, during the execution of a business process affecting the flow of the process. Since an event can start, suspend, or end the flow, we can distinguish between start events, intermediate events, and end events4. The last basic construct referring to this category is the sequence flow (modeled with an arrow), used to represent the ordering of activities within a process. Their source and target must be events, activities, and gateways. A sequence flow can not cross the boundaries of a sub-process or a pool (messages are used instead).

- **Object**. This category refers to the constructs devoted to model passive entities involved in one or more business processes. The main BPMN element of this category is data object. A data object is used to represent how data and documents are used within a process. They can be used to define input and output of activities. Furthermore, data objects are used to represent the state of a document (e.g. request document issued or received) and how this state changes during the process.

- **Co-action**. The last category refers to the constructs devoted to model the interaction between different actors. Message events represent start, intermediate, and end events associated to the sending or receiving of a message. The message flow is used to show the flow of messages between two participants of a process. It can be connected to the boundary of a pool, representing a participant in a process, or to an activity within the pool.

An example: Existence Verification of a Company

To better illustrate the BP modeling by using BPMN, below we introduce a simple example (Figure 2.2), drawn from a case study addressed in the LD-CAST European Project[1]: the search for a partner in a cross-border off-shoring. In particular, it is addressed the activities related to the Supply Sub-Contracting, according to a specific EU directive.

The case study concerns a Polish company that needs to purchase a set of lift components to be locally assembled. To this end, it publishes in the Official EU Gazette a call for proposals. The company receives tenders submitted by a Rumanian company for supplying the cables, by an Italian company for providing the engine, and by a Bulgarian company for the cabin. Having received

[1]Local Development Cooperation Actions Enabled By Semantic Technology (LD-CAST) Project: http://www.ldcastproject.com

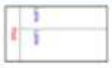
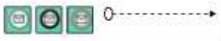
Modeling categories	BPMN elements	Graphical Notation
Actor	Pool, Lane	
Behavior	Process, Task, Sub-process	
Decision	Gateway (Data/Event based, AND/OR/XOR)	
Event	Event (Start, Intermediate, End)	
Transition	Sequence Flow	
Object	DataObject	
Coaction	Message Event, MessageFlow	

Figure 2.1: Categorization of BPMN constructs and corresponding graphical notation

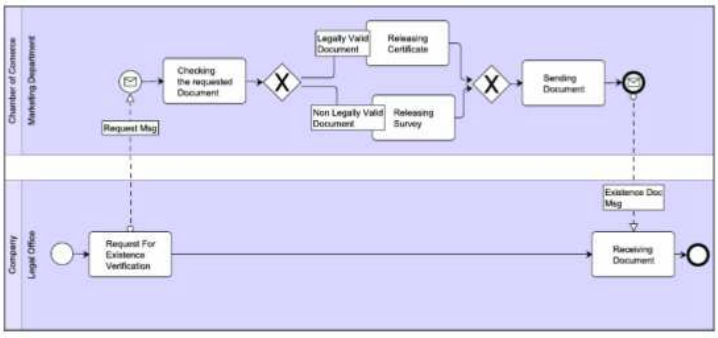


Figure 2.2: An excerpt of the existence verification process

the tenders, the Polish company needs to perform a number of verifications. In particular, for each proponent company, the following verifications are per-

formed:

1. existence verification;
2. fiscal verification;
3. technical requirements and quality standards verification.

Such verifications are performed through the local Chambers of Commerce. The Figure 2.2 represents the BPMN process concerning point (i), the request for verifying the actual existence of the potential partner. According to the process in the figure, the legal office of a company sends a request message to the marketing department of the local Chamber of Commerce (CoC) to obtain an existence verification of that company. The CoC checks if the requested document should have a legal validity or not. Accordingly, the CoC releases a certificate or a survey, respectively. Finally, the CoC sends the document, the company receives it and the process ends. This example will be represented in formal terms by using BPAL, as reported in the appendix.

Chapter 3

Business Rules

Business rules are used by managers to indicate invariants that impact on the organization and its way of operating. Rules represent a very powerful modelling paradigm. Here we focus on the impact of rules guiding the design and the evolution of BPSs. BRs are usually specified in natural language, but it is well-known that NL is often ambiguous and error-prone. For this reason, there are interesting proposals of using structured (controlled) natural language. Recently, OMG promoted the use of structured English in the business rules framework SBRV [**?**]. Another interesting proposal is ACE (Attempto Controlled English [**?**]): a rich subset of standard English, designed for specification and knowledge representation. By using ACE, BRs can be expressed in rigorous way, having at the same time a simple (almost) natural language interface that can be easily used by business experts. ACE relies on a platform capable of translating a sentence expressed in a controlled natural language into a first order logic (FOL) formula. The latter can be then verified by using a theorem prover. In this work is showed how the problem of BR formulation can be effectively solved in the context of a BPAL ontology. It is showed how is it possible to support a business expert in formulating an example BR of the kind: A precedes B. The BR will be initially expressed in generic terms, then incrementally refined, by prompting the business expert with simple, specific questions, aiming at progressively removing the ambiguities.

3.1 On the semantic of a BR

As anticipated, in order to illustrate the BPAL approach to BR formulation, we use an example BR, informally expressed by the sentence: A precedes B. Although the intuitive meaning seems to be clear, a more careful analysis shows different possible interpretations. Here we will consider, as an illustrative example, a subset of the cases reported in [**?**]. The semantics of each case will be represented at an intuitive level.

1. **Response**: every time activity A executes, activity B has to be executed after it. B does not have to execute immediately after A, and another A can be executed between the first A and the subsequent B. Furthermore, an execution of B does not require to be preceded by A. i.e. in Figure 3.1;

Figure 3.1: Response semantics.

2. **Precedence**: if activity B is executed, its (possibly multiple) executions must follow the execution of A. i.e. in Figure 3.2;

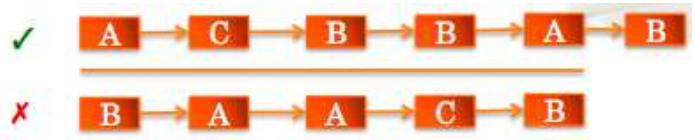


Figure 3.2: Precedence semantics.

3. **Alternate response**: after activity A there must be an activity B, and before that activity B there can not be another activity A (but a B not following A is ok). i.e. in Figure 3.3;

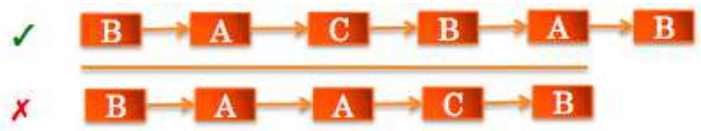


Figure 3.3: Alternate Response semantics.

4. **Alternate precedence**: every instance of activity B has to be preceded by an instance of activity A and the next instance of activity B can not be executed before the next instance of activity A is executed. i.e. in Figure 3.4;

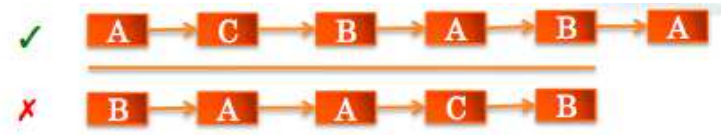


Figure 3.4: Alternate Precedence semantics.

5. **Chain response**: the next activity after each activity A has to be activity B, then the execution in Figure 3.5 is a correct one;

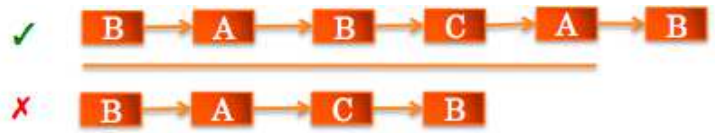


Figure 3.5: Chain Response semantics.

6. **Chain precedence**: requires that the activity A is the first preceding activity before B (but is not necessary that a B follows an A) and, hence, the sequence in Figure 3.6 is correct.

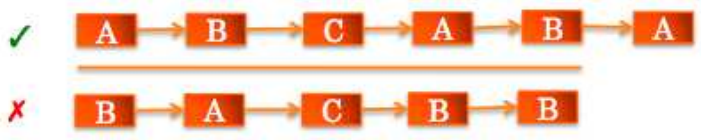


Figure 3.6: Chain Precedence semantics.

Clarifying the semantics of a BR

Our goal is to disambiguate the BR A precedes B, with respect to the six reported cases, by posing few and simple questions to the user. Instead of using a question for each of the six (fairly complex) cases reported above, we propose a more intuitive disambiguation method, requiring a lower number of simple yes/no questions.

1. Can I execute any activities between A and B?
2. Can an activity B be executed before an activity A?
3. Does an activity B has to be executed after an A?

The proposed disambiguation method is based on these three questions and a Decision Table, as reported below. By answering these simple yes/no questions the user will be driven to clarify the intended meaning of the BR prec(A,B), by selecting it among possible alternative meanings. Figure 3.7 represents the mapping from answer combinations to possible meanings.

3.2 Business Rule Approach

The business rules approach has drawn a lot of attention already and a number of architectures, technologies, frameworks have been proposed and a number of tools have been developed (both for research and commercial purposes). The proposed BR-centric approaches are applicable for explicit work with business rules at various phases of IS development cycle. However, there are just few results reported on business rules automation. By business rules automation it can be assumed the automatic generation of executable business rules specification from the declarative business rules statements or business rules model in some modelling language or interchange format.

		Rules							
Questions	Can I execute any activities between A and B?	Y	Y	Y	Y	N	N	N	N
	Can an activity B exist before an activity A?	Y	Y	N	N	Y	Y	N	N
	Must, an activity B, be executed between two activities A?	Y	N	Y	N	Y	N	Y	N
Semantics	Response (1)		X						
	Precedence (2)				X				
	Alternate Response (3)	X							
	Alternate Precedence (4)			X					
	Chain Response (5)					X			
	Chain Precedence (6)								X

Figure 3.7: Decision Table to disambiguate the BR: A precedes B.

Business Rule Concept

A number of definitions for business rule were developed. Business rules definition may be analysed from two perspectives: business perspective and IS development perspective [**?**], [**?**]. From business perspective ('Zachman [**?**] row-2') business rule is a statement that defines or constrains some aspect of the business; it is intended to assert business structure, or to control or influence the behaviour of the business. From IS perspective ('Zachman row-3') business rule is a statement which constrains certain business aspect, defines business structure, and controls business processes that are supported by enterprise information systems. In IS business rules may be implemented as facts registration (as data) and constraints applied during registration process. Von Halle in [**?**] summarizes the business rule definition problem as follows: depending on whom you ask, business rules may encompass some or all relationship verbs, mathematical calculations, inference rules, step-by-step instructions, database constraints, business goals and policies, and business definitions. In our research we use the definition given above providing the concept in business context business rule defines the way of operating enterprise business (policies, guidelines, behaviours, etc.). However, we require that enterprise business rules model comprised only atomic business rules. Atomic business rules are such that cannot be broken down or decomposed further into more detailed business rules because, if reduced any further, there would be loss of important information about the business. This limitation comes from the IS implementation perspective and is reasonable because not-implementable business rules model

can guarantee neither consistency, nor unambiguity. The various taxonomies of business rules discussed in details in [**?**], [**?**] show the lack of standards in business rules community on types, classes and categories of business rules. However, the surveys show that the taxonomies of business rules presented by different authors depend on the intended purpose (for example, enterprise management or implementation of business rules in IS, or implementation of business rules in rules engines).

BR-Centric Frameworks

According to the survey of BR-centric frameworks, architectures and technologies given in [**?**] [**?**], [**?**], the proposed ideas are rather diverse depending on perspective or intended purpose and can be summarised as follows:

- From implementation perspective the proposed business rules approaches can be classified into three broad types: rules implemented as application logic components, rules implemented using active databases technologies, and rules implemented in rules engines (enforcement, inference, etc).
- From architectural focus: different authors stress different IS development life cycle phases from elicitation to maintenance; accordingly their proposed frameworks vary. Some concentrate on business objects definitions and modelling, others go for automatic implementation frameworks and technologies.
- From modelling perspective: a lot of attention is paid to the modelling issues of business rules. Some proposed modelling techniques, for example, by Ross [**?**], are both modelling language and modelling method in one. Another approaches stem from adapting popular modelling languages, such as UML and OCL, to business rules modelling activity. However, none of the proposed languages or methods are accepted as technology standard yet.

BR-Centric Frameworks

Business Rules Tools Available
There is a number of different business rule management and enforcement tools available today. The common component of the majority of tools is the business rules repository which is later used for different tasks. However, as it was mentioned above, they employ different, sometimes very specific modelling

languages. As for the functionality that the tools offer it is centred on the following tasks:

- Manage rules components for rules input and modification (rule editors);
- Store rules rules repository;
- Enforce rules rules engines or similar mechanisms.

The rules enforcement components offer the reference (transparent for business users) from declarative business rules statements stored in business rules repository to the actual enforcement mechanism thus achieving the required automatic dependency of the business rules implementation on declarative business rules statements. Blaze Advisor [9] system offers the complete process for designing, running, and maintaining e-business applications. Blaze Advisor consists of the following 5 components:

- Builder a rule creating tool targeted for developers;
- Innovator a rule management and maintenance tool targeted for business users;
- Rule Engine a scalable processing engine that determines and executes the control flow of rules, works together with the Rule Server;
- Rule Server a dedicated rule server which supports rule execution, session management, scheduling, and dynamic load balancing.

Blaze Advisor uses its unique Structured Rule Language for input of business rules; decision trees and decision tables can be employed as well. Infrex [**?**] is another type of tool while Blaze Advisor is a totally stand alone application, Infrex can be embedded into applications written in C/C++/Java/C#. Infrex uses classes and variables of the application with the support for high level operators for rules specification. Rule Translator component generates C/C++/Java/C# code which is compiled and linked with the application to create an executable. The executable has the rules to be called at run-time, through the engine. Thus the adaptivity feature is achieved by the ability of the tool automatically create executable code from the rules specification. However, the language for rules specification is not suitable for business users because it directly operates with classes and variables which are not exactly the business terms. QuickRules [**?**] is a business rules management system which allows inserting and editing business rules using the specific QuickRules Rules

Mark-up Language. Rules are stored in XML format and may be executed by Business Rules Engine component. ILOG rules are of ECA (Event, Condition, Action) type and consist of three parts: header, condition and action specifications [**?**]. The rules can be defined using BAL (Business Action Language), IF-THEN-ELSE rule format and TRL (Technical Rule Language). The above mentioned tools have their own business rules engines therefore it is obligatory for enterprises to buy the respective product suit in order to be able to use their offerings. The opposite solution would be to use a wide spread technology (e.g., of active database management systems [**?**]) for rules repository and as an enforcement engine. The Dulcian company (an Oracle consulting firm that specializes in data warehousing, systems development and products that support the Oracle environment) is working in this direction offering BRIM Business Rules Information Manager [**?**] which offers the rules editing, designing and implementation functionality. The rules editing is done over two steps analysis rules are entered using weakly structured RuleSpeak language (proposed by Business Rules Solutions [[**?**]) and implementation rules are specified using UML [**?**] class and activity diagrams. The mapping between analysis and implementation rules may be done by business users manually associating rules with classes, states and diagrams. The automatic code generation component creates text files that either creates triggers on tables or alters the existing triggers on tables. The technology is completely Oracle- centred. The survey of the tools given above is by no means complete but it highlights the trends within the field of interest. The lack of standards is obvious for business rules modelling languages, repositories format, architectures. It is not possible to exchange business rules among different products or present to the business users more or less standard rules language. However, the promising step in business rules repository is the usage of XML as the business rules storage format which would enable the business rules sharing and exchange.

Chapter 4

An Ontological Framework: BPAL

4.1 Business Process Abstract Language

Below, we illustrate a first version of a proposal aimed at defining the key modelling notions of a Process Ontology, referred to as BPAL: Business Process Abstract Language. It has been conceived to support the formal definition of a BP, having in mind the modelling notation proposed by OMG, mainly directed to business people: Business Process Modeling Notation (BPMN.) Furthermore, in the proposed representation method we include, inherently, modelling facilities to support a stepwise development of a BP. Summarising, we propose BPAL, as a Core BP ontology to support the use of BPMN (but it is general enough to be used in conjunction with any other BP modelling method), characterised by the following properties:

- key constructs corresponding to concepts and modelling notions drawn from the business community;
- formal grounding, with the possibility of translating it to a formal language such as KIF [MRG92];
- axiomatization to allow inference mechanisms to be applied;
- possibility of execution, by mapping BPAL to the popular BPEL.

The set of symbols denoting such modeling notions is the lexicon of the BPAL. The corresponding concepts, in the form of atomic formulae (atoms),

represent the core BP ontology. BPAL atoms are used to build abstract diagrams that, once validated with respect to the BPAL Axioms, yield to abstract processes. An abstract process is isomorphic to a BPMN7 process and provides its formal semantics.

- **BPAL Atoms**: represented in the form of predicates. They are the core of the BPAL ontological approach and a specific BP ontology is modeled by instantiating8 one or more Process Atoms. Atoms represent unary and n-ary business concepts. Furthermore, there are special atoms aimed at supporting the modeling process (see below.)
- **BPAL Axioms**: represent the rules and constraints that a BPAL Diagram must respect to be a BPAL Process.
- **BPAL Diagram**: a set of BPAL Atoms represents an abstract diagram. In general, a diagram is an intermediate form assumed by a modeling artifact during the design process. It is not required to satisfy all the axioms.
- **BPAL Process**: this is a BPAL Diagram that has been validated with respect to the BPAL Axioms. In the paper, we will refer to it as an abstract process.
- **BPAL Application Ontology**: a collection of BPAL Processes cooperating in a given application. Below we report a list of the BPAL Atoms and then we give a few examples of their use for building abstract diagrams and BPs. Please remember that BPAL is not a diagrammatic notation, for this reason we refer to a BPAL artefact as an abstract diagram. Seen it originates from BPMN, the latter is used whenever we need a concrete (displayed) diagram.

BPAL Atoms

The BPAL atoms are predicates where functors represent ontological categories, while arguments are typed variables representing concepts in the Core Business Ontology (CBO). For instance, an activity variable can be instantiated with a process name in the CBO. Variables are characterized by a prefixed underscore and, in building a BP ontology, will be instantiated with concept names of the category indicated by the functor. A process ontology is built by instantiating the following unbound predicates with the constants (i.e., concept names) declared in the CBO.

Unary predicates (upre)

- act(_a): a business activity, element of an abstract diagram.
- role(_x): a business actor, involved with a given role in one or more activities.
- dec(_bexp): a generic decision point. Its argument is a Boolean expression evaluated to true, false. It is used in the preliminary design phases when developing a BP with a stepwise refinement approach. In later phases, it will be substituted with one of the specific decision predicates (see below).
- adec(_bexp), odec(_bexp): decision points representing a branching in the sequence flow, where the following paths will be executed in parallel, or in alternative respectively.
- cont(_obj): an information structure. For instance a business document (e.g., purchaseOrder).
- cxt(_obj): a context, represented by a collection of information structures.

Relational predicates

- prec(_act—_dec,_act—_dec): a precedence relation between activities, decisions, or an activity and a decision.
- xdec(_bexp,_ trueAct): this is a decision where only one successor will receive the control, depending on the value of _bexp.
- iter(_startAct,_endAct,_bexp): a subdiagram, having startAct and endAct as source and sink, respectively. It is repeated until the boolean expression _bexp evaluates to true.
- perf(_role,_act): a relation that indicates which role(s) is dedicated to which activities.
- msg(_obj,_sourceNode,_destNode): a message, characterized, for instance, by a content (_obj), a sending activity (_sourceNode), and a receiving activity (_destNode).

Development predicates The following predicates are used during the BP development process. They are part of the BPAL core ontology, but are not used to categorise business concepts (therefore will not contribute to the generation of the executable image.)

- pof(_upre,_upre): Part of relation that applies to any unary predicate. It allows for a top-down decomposition of concepts.
- isa(_upre,_upre): Specialization relation that applies to any unary predicate. It allows to build a hierarchy of BP concepts, supporting a top-down refinement.

Finally, we have two operations acting on a BP abstract diagram:

- Assert (BP_Atom). It allows a new atom to be included in the ontology;
- Retract (BP_Atom). It allows an existing atom to be removed from the ontology.

To improve readability, multiple operations of the same sort can be compacted in a single operation on multiple arguments (see the example below.)

BPAL diagrams and processes

By using BPAL atoms it is possible to compose an abstract diagram first and, after its validation, a BPAL process. An abstract diagram is a set of BPAL atoms respecting the (very simple) formation rules. In Figure 4.1 we illustrate an abstract diagram; the presentation is supported by a concrete diagram, drawn with a BPMN style. The node labels are concepts in the CBO.

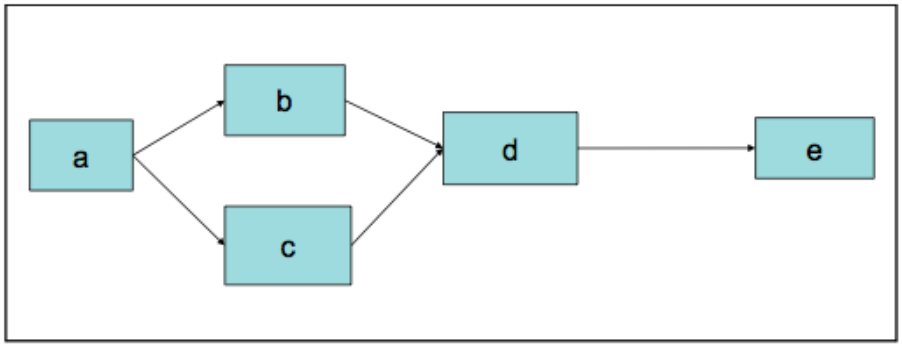


Figure 4.1: A simple BPMN Diagram.

Here, in Figure 4.2 we have the corresponding BPAL abstract diagram.

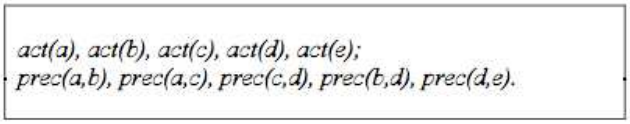


Figure 4.2: A BPAL Abstract Diagram.

Please note that in a BPAL diagram the order of the atoms is ininfluent and, in the punctuation, comma and colon are equivalent, while the full stop ends the abstract diagram.

BPAL Axioms

The BPAL framework is characterised by a number of axioms that must be satisfied by a BPAL Process. They are conceived starting from the guidelines for building a correct BPMN process. As anticipated, there is neither a formal specification nor a widely accepted view of the formation rules for a BPMN process, therefore we provide a reasonable solution, derived from the analysis of a number of publications and practical experiences. In any case, we are ready to update the proposed axiomatization as soon as an official specification will be available. Here we do not intend to present a complete treatement of the BPAL axiomatic theory, we rather wish to achieve a clear description of our proposal, trading completeness with conciseness. BPAL axioms address different features of a BPMN process formalization. Here we will formalize just one axiom, to provide a first insight in the BPAL methodology.

Branching Each time a node is followed by more than one immediate successor activities, such a node must be a decision.

$$\forall x, y \in CBO : act(y) \wedge S(x) = \{y : prec(x, y)\} \wedge \mid S(x) \mid > 1 \rightarrow dec(x) \quad (4.1)$$

According to Axiom1, the diagram of Figure 1 is invalid and needs to be transformed in the diagram of Figure 4.3.

This transformation is obtained by a number of updates on the original BPAL abstract diagram, sketchily summarised as follows:

$assert : dec(k), prec(k, b), prec(k, c), prec(a, k)$

$retract : prec(a, b), prec(a, c)$

Accordingly, the BPAL abstract diagram in Figure 2 is transformed into:

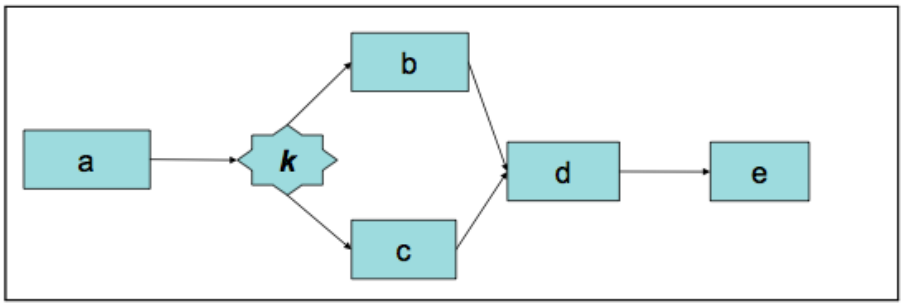


Figure 4.3: A Generic BPAL Process.

$act(a), act(b), act(c), act(d), act(e), dec(k)$;
$prec(a,k), prec(k,b), prec(k,c), prec(c,d), prec(b,d), prec(d,e)$.

Abstract BPAL generic process

Please note that we have now a Generic BPAL abstract process. It is a process since the Axiom1 is no more violated and therefore the Diagram 4.3 is validated. However, it is Generic, since there is a generic atom dec(k) that need to be substitute with a specific atom (one of: adec, odec, xdec.) Such a substitution, assuming that in the following steps of the BP design we discover that we need an and branching, will be achieved by the following design operations:

$assert : adec(k)$
$retract : dec(k)$

Further steps of design refinement will involve other BPAL atoms, to specify roles, messages, etc.

Transforming BPAL abstract process in an executable form

The BPMN tool that we considered as a reference point allows a BPEL executable process to be generated starting from the built diagram. Since there is a tight correspondence between the BPMN and the BPAL constructs, it is easy to adopt an equivalent correspondence between BPAL and BPEL constructs, and then provide the generation of an executable BPEL file starting from a complete BPAL process.

4.2 The semantic enrichment of BPMN: a mapping to PSL and BPAL

One of the initial objective of this work was to provide a formal account of a BPMN diagram. This can be achieved by building a mapping between the constructs of the latter and a formal language for BP modeling. To this end we considered as a candidate PSL, presented in Chapter . However, we realized that BPMN-PSL mapping is not straight, being PSL a sound and rich BP modeling framework, not primarily conceived having the business modeling needs in mind. Therefore we decided to propose BPAL. In the Figure 4.4, we report a sketchy representation of a comparison between the two mappings: BPMN with PSL and BPAL, for an easy checking the better mapping yield by the latter. A first consideration concerns the fact that PSL Core is far to succinct and it must be considered with some extensions in any concrete case. For a fair comparison we considered the following PSL extensions (the semantics of the constructs is intuitive):

- PSL-Core:
  - Activity
  - Before
  - Object
- Outer Core:
  - SubActivity
- Activity Extension:
  - Branch
- Actor and Agent theories
  - Activity performance
- Duration and Ordering Activity
  - Iterated Activity

Modeling categories	BPMN constructs	PSL	BPAL
Actor	Pool, Lane	Activity performance	role
Behavior	Process, Task, Sub-process	Activity, Subactivity	activity, iter, isa
Decision	Gateway (Data/Event based, AND/OR/XOR)	Branch	dec, adec, odec, xdec
Event	Event (Start, Intermediate, End)		activity
Transition	Sequence Flow	Before	precedence
Object	DataObject	Object	content, context
Coaction	Message Event, MessageFlow	Envelop	message

Figure 4.4: Comparison between the BPMN, PSL, and BPAL.

From the Figure 4.4 it emerges that, with respect to the BPMN modeling paradigm, BPAL shows a better coverage than PSL. Furthermore, BPAL is compact, while PSL is fragmented in different extensions. These considerations justify our work on BPAL. The different mappings should not be confused with the respective expressive power (with its various extensions, PSL is more expressive than BPAL.) Here we prefer to talk about adequacy, i.e., the affinity between the two modeling methods.

4.3 Business Process schema and instances in BPAL

BPAL [11] is an ontology modelling framework that allows a predicative specification of a BP Schema to be formulated. The simplified version of BPAL adopted in the rest of the work uses symbols for constants (a, b, c,), variables (?a, ?b, ?c,), and conditionals (?h, ?k,), as well as the following atoms to represent:
- Activities: act(?a)
- Precedence relations: prec(?a,?b)
- Decisions: dec(?k,?a)

Note that decision atoms can be specialized to AND, OR or XOR type, e.g. a XOR decision will be represented by xdec(?k, ?a). Where ?a represents the then-activity in a typical IF-THEN-ELSE decision. When more than two mutually exclusive branches are available, it is necessary to use a sequence of xdec. Conversely, the AND, OR decisions essentially act as a semaphore go/no-go and therefore they only require a conditional. We distinguish a BP Schema (BPS), which is a set of predicative atoms, from a BP Instance (BPI) represented by a chain of ground terms. A BPI originates from the actual execution of a BP Schema and each element of the chain (activity instance) corresponds to an execution of an activity (variable) in BPS. Activity variables are assumed to be typed, according to a set of activity kinds, denoted by capital letters (A, B, C). To keep the notation lightweight, in this paper we avoid the explicit typing of activities, and the implicit typing is achieved by matching the first letter of the term. For example, type(?a, A) is always true. A similar method is applied to instantiation and activity instances are denoted by letters that correspond to the activity kinds. A relation inst(?a,a) says that a is an instance activity of ?a. We will also refer to a BPI as a BP Log. Activity kinds are essential to express rules and constraints, since they are general and do not apply to a single BPS or BPI, but to all of them (unless otherwise specified.) Therefore a rule of the form: prec(A,B) imposes a control flow constraint on all activities act(?a) and act(?b) of type A and B, respectively. Then, prec(?a,?b) is consistent with a BR stating prec(A,B) and consequently all its executions will produce valid ground terms of the form: prec(a,b). In the following we show a simple example of a BPS modelled as a set of atoms, and its BPIs, modelled as chains of ground terms. In Figure 4.5, the following BPS is displayed:

bps1 = act(?a), act(?b), act(?c), act(?d), act(?e), prec(?a,?b), prec(?b,?d), prec(?d,?e), prec(?a,?c), prec(?c,?e)

The execution of the BPS in Figure 4.5 will produce one of the following instances, depending on the kind of branching:
1) In case of an AND branching: bpiAND = $\{ < a, b, c, d, e >, < a, c, b, d, e > , < a, b, d, c, e > \}$.
2) In case of an OR branching: bpiOR = $\{< a, b, c, d, e >, < a, c, b, d, e >, < a, b, d, c, e >, < a, b, d, e >, < a, c, e >\}$.
3) In case of a XOR branching: bpiXOR = $\{< a, b, d, e >, < a, c, e >\}$.

Please note that an activity instance is registered in the log at the moment of its completion. Therefore prec(b,c) may hold, even if the activity instance c

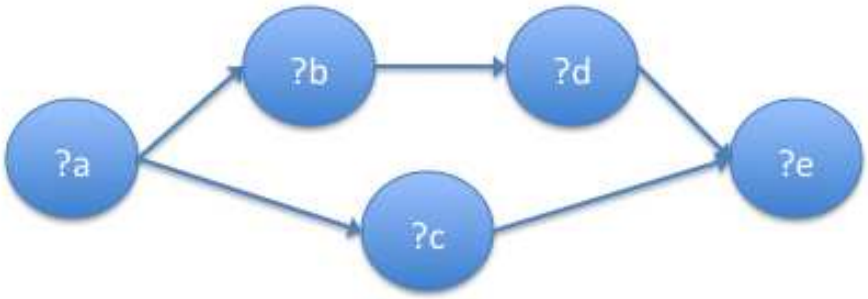


Figure 4.5: A simple BPS diagram.

has started before b, because c execution may take longer, causing c completion to follow that of b.

4.4 Knowledge Representation Framework

Knowledge representation is an area in artificial intelligence that is concerned with how to formally "think", that is, how to use a symbol system to represent "a domain of discourse" - that which can be talked about, along with functions that may or may not be within the domain of discourse that allow inference (formalized reasoning) about the objects within the domain of discourse to occur. Generally speaking, some kind of logic is used both to supply a formal semantics of how reasoning functions apply to symbols in the domain of discourse, as well as to supply (depending on the particulars of the logic), operators such as quantifiers, modal operators, etc. that along with an interpretation theory, give meaning to the sentences in the logic.

When we design a knowledge representation (and a knowledge representation system to interpret sentences in the logic in order to derive inferences from them) we have to make trades across a number of design spaces. The single most important decision to be made, however is the expressivity of the KR. The more expressive, the easier (and more compact) it is to "say something". However, more expressive languages are harder to automatically derive inferences from. An example of a less expressive KR would be propositional logic. An example of a more expressive KR would be autoepistemic temporal modal logic. Less expressive KRs may be both complete and consistent (formally less

expressive than set theory). More expressive KRs may be neither complete nor consistent.

In Figure 4.6 is presented the Knowledge Representation Framework used in this work. L2 shows the high level concepts and the relations between them. In it we have unambiguous concept definition. L1 comprehends the abstract language that allows a predicative specification of a Business Process to be formulated. In it we have the n times uses of that concept in the process schema. L0 shows execution level with all the execution traces of the BPAL schema.

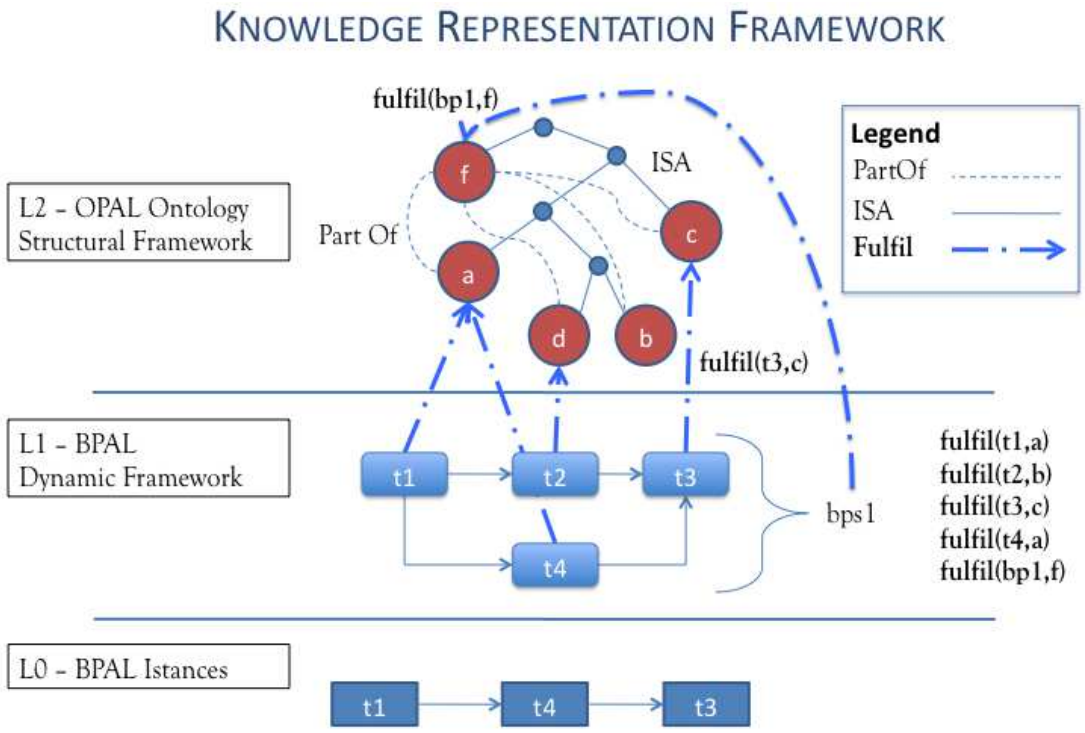


Figure 4.6: Knowledge Representation Framework used.

Chapter 5

Pilot Case

In this chapter will be presented a pilot case which will let a better understanding of the study faced.
The case showed is an e-procurement process that involve communication between a seller and a buyer. Several actions have to be performed by the two actors to let the process come to a successful end.

5.1 The E-procurement Community

The concept of a Community is used in the Organisational Perspective to discuss the collection of entities (e.g. individuals, organisations, information systems, resources, or various combination of these), established to meet some objective. A Community is specified in terms of community roles and a community contract. In the case of e-procurement, the objective to be achieved is the exchange of documents in order to affect the purchase of goods or services. The community consists of several community roles. In this case the most important of these are a buyer and a seller, and other roles can include e- procurement hubs, banks, shippings agents and others. The way in which these roles relate to one another is known as a community contract, and it is well understood by all parties involved in e-procurement.

5.2 Buyer and Seller

eProcurement is an inter-enterprise process that is conducted with two main roles: a buyer and a seller with the purpose of the buyer purchasing goods

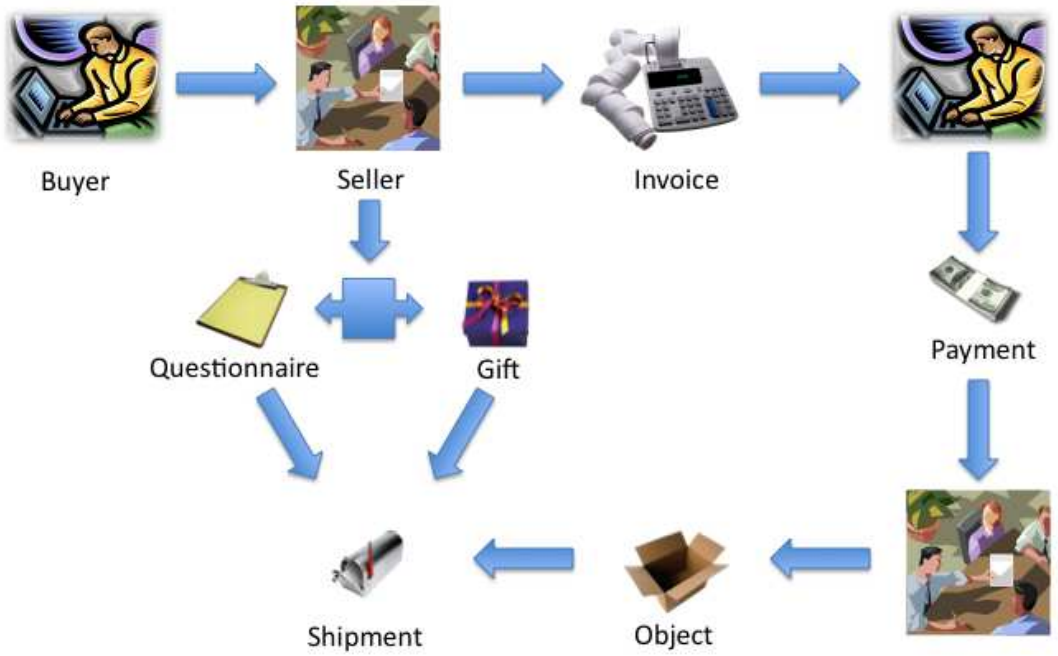


Figure 5.1: Informal representation of an E-procurement process.

(including services) from the seller. This is done by the exchange of various documents, followed by the exchange of goods and money. The scope of this presentation is limited to the exchange of a small number of documents. The logistics of goods delivery and receipt, and financial transactions are explicitly out of scope.
The eProcurement processes have been narrowed down to a core set of interactions involving documents starting with Purchase Order and ending with Invoice. All catalogue, order fulfilment logistics and payment related parts of these processes have been deemed out of scope for the first version of the e-procurement model. It is important when exchanging actual e-procurement documents that the process context of the documents is understood, so that correct error handling can be in place when expected interactions do not occur.

5.3 E-procurement Process

The online buying process starts with a Buyer who performs a research on Internet or on a magazine to find what he needs. If he finds what he's looking for on Internet he will compile a web form and will send the data to the seller otherwise he will use the traditional post methods. At this point the purchase order is sent to the Seller who will process it and will start two simultaneous set of actions.
The first set is about the payment while the second is relative to the client care.

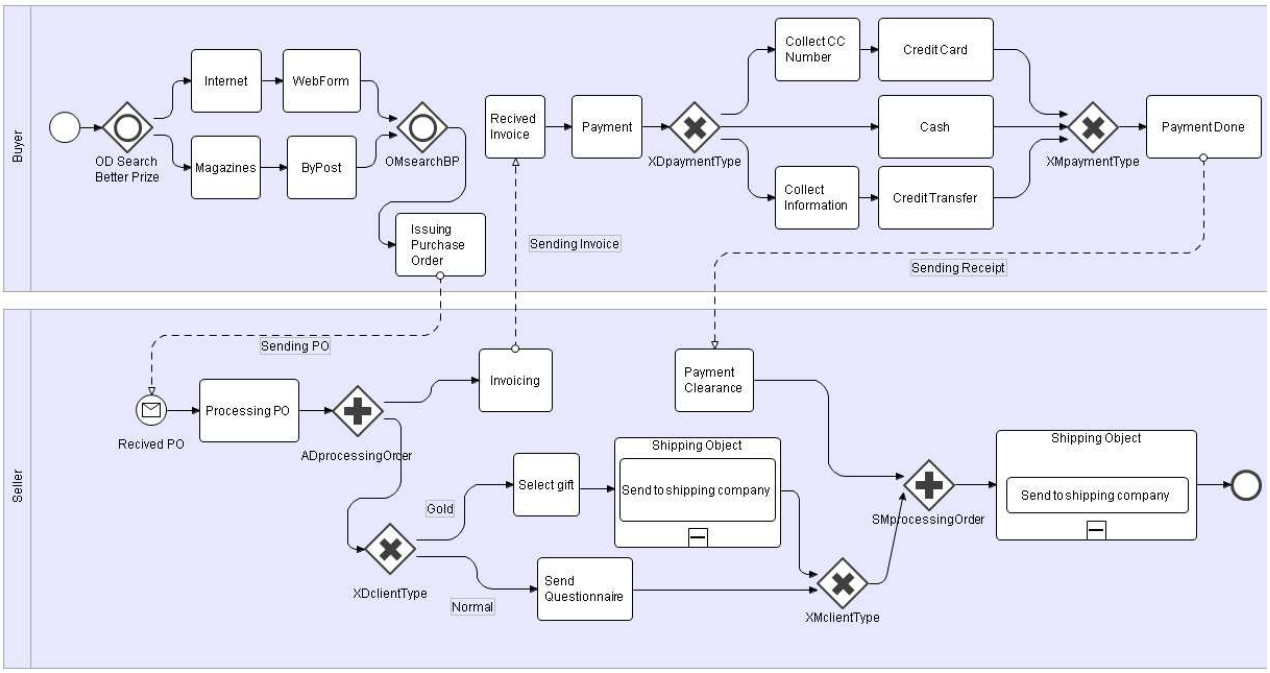


Figure 5.2: BPMN formalization of the E-procurement process.

In the first set the Seller sends the invoice to the Buyer who will receive it and will decide the payment method. When the Buyer ends his payment action will send the receipt to the Seller.
While these actions are carrying out, the other action set is performing. The

Seller will control if the client is a gold client or a normal one and in relation of the typology he will send a gift or a questionnaire. Only when both sets of actions are performed the process can go on and the Seller sends the requested object to the shipping company. In the Figure 5.1 there's an informal representation of the process. The same process can be presented in a formal and shared notation such as BPMN in fact in the Figure 5.2; presents the BPMN form of the E-procurement process.

Chapter 6

Impact of Business Rule onto Business Process Schema

The goal of this work is twofold: to support the business expert in clarifying the intended meaning of a rule (when its interpretation is ambiguous) and verifying which BPs in a repository are not consistent with the rule. In the previous chapter we addressed the first objective, here we address the latter. To this end, we consider an example of a BPS fragment and analyse the impact that the constraint prec(A,B) has on it. Note that our approach is inherently different from BP instance based approaches (e.g. that in [**?**]), since we only address the intensional level, i.e., BP schemas. Note also that our approach can be applied even if some decision nodes are not fully specified (i.e., as AND, OR, XOR), although more detailed and precise information about BR violation can be inferred when decision nodes are specified.

6.1 Testing BP consistency to BRs

Table 2 shows the validity of a process schema for each of the six different semantics of the constraint A precedes B considered above. We assume that in order to be valid a BP must satisfy the rule for all its possible execution (instances). In the first column an example of business process schema is shown with two (generic) decision nodes. The possible specification combinations for the decision nodes are listed in the second column, in particular:

- AND: The presence of an AND operator creates a non determinism in

the process execution. In fact, it is not possible to determine a-priori the execution order of two activities located on two parallel paths.

- XOR: In this case the two branches are mutually exclusive and the resulting logs will be constituted by the activity sequences of either one or the other branch.
- OR: The semantics of the OR decision node is the most articulated since the resulting logs is the union of the two previous cases.

The third column is organised in six sub-columns: one for each meaning that the natural language sentence A precedes B can assume, more precisely:

1. Response
2. Precedence
3. Alternate Response
4. Alternate Precedence
5. Chain Response
6. Chain Precedence

The row-column intersections show the validation results corresponding to the combinations of decision node specifications and sentence meanings, namely:

- V: Every instance of the schema validates the constraint.
- N: No instance of the schema validates the constraint.
- S: Some instances of the schema doesnt validate the constraint.

The values obtained (V, N, S) can be effectively used to support the BP designer in updating and refining the BP schema, by highlighting the inconsistencies and providing warnings for potentially inconsistent situations. Note that in 5 out of the 6 possible BR interpretations an inconsistency can be detected, even if both decision nodes remain unspecified. Conversely, when considering the first interpretation, the consistency can be verified in some cases, even if only one of the two decision nodes is specified.

6.2 A practical example

In Figure 2, we present an example of a BPMN process (realised by using the Intalio Editor Tool [**?**]). This process deals with a procurement scenario; the two pools are two different organizations performing the roles of Buyer and Supplier. The Buyer sends the Request for Quotation (RFQ) to the Supplier which replies sending back a Quotation. Afterwards, the Buyer analyzes the Quotation and, if satisfied, sends the Purchase Order (PO) to the Supplier, otherwise, the Quotation is rejected. Invoicing from the Supplier and Payment from the Buyer concludes the process.

The illustrated BPMN process can be represented in BPAL as follows:
act(sendingRFQ),
act(processingRFQ),
act(sendingQuotation),
act(receivingQuotation),
act(issuingPo),
act(processingPo),
act(invoicing),
act(payingInvoicing),
act(rejectingQuotation),
prec(sendingRFQ, processingRFQ),
prec(processingRFQ,
sendingQuotation),
prec(sendingQuotation,
receivingQuotation),
prec(recivingQuotation,
quotationAccepted),
prec(xdec, rejectingQuotation),
prec(xdec, issuingPO)
xdec(quotationAccepted, issuingPO),
prec(issuingPO, processingPO),
prec(processingPO, invoicing),
prec(invoicing, payingInvoice).

An example of BR to be tested on this BPS is the following: the activity of receiving a quotation must precede that of issuing a Purchase Order. In order to disambiguate the BR we apply the decision table method presented above and we assume, for example, that the three yes/no answers of the table identify the Alternate Precedence (4) as the intended BR meaning. By analysing the

consistency table for this BPS, we can infer that a path is inconsistent and some BP instances may be invalid. In particular the BPI can violate the specified BR if the quotation is not accepted. In general the management should therefore be promptly alerted that some modifications to the BPS, to make it consistent with the BR are required, sometimes could also happen that the management comprehend that the BR describes no more the reality of the company and it should be updated. In Figure (n) the branch arising the problem is highlighted.

Chapter 7

Control Coherence Method

This chapter will present the method created to solve the problem of identifying inconsistencies between business rules and business process. The adopted solution is set inside an architecture, showed in Figure 7.1, that will be presented in the next section.

7.1 Architecture

As shownin Figure 7.1, the inputs are Business processes and Business Rules. The choice made in this work is oriented towards the most common of configuration. Business Process designed in BPMN and Business Rules expressed in natural language. On the left side of the architecture can be seen a transformation, under development, that take as input the BPMN Business process and produces as output a BPAL Business Processes. On the other side of the architecture there are the Business Rules initially expressed in natural language then, through a transformation using Attempto Controlled English (ACE), presented in the subsection 7.1. ACE is a method, comprehensive of several tools, to transform natural language sentences in a first order formal language: After using ACE the rules can be transformed in BPAL rules.
At this point, with both Business Process and Business Rules in BPAL format, the Control Coherence Algorithm is activated and in output there will be the information about the inconsistent process pieces with the rules.

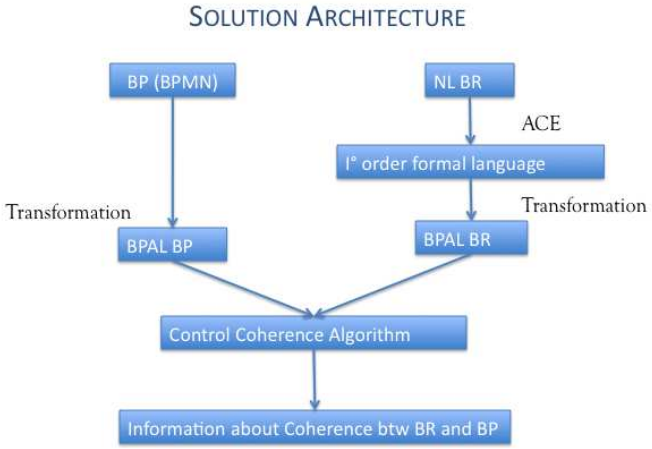


Figure 7.1: Solution Architecture.

Attempto Controlled English

Attempto Controlled English is a controlled natural language, i.e. a subset of standard English with a restricted syntax and a restricted semantics described by a small set of construction and interpretation rules.
ACE can serve as knowledge representation, specification, and query language, and is intended for professionals who want to use formal notations and formal methods, but may not be familiar with them. Though ACE appears perfectly natural it can be read and understood by anybody it is in fact a formal language.
ACE and its related tools have been used in the fields of software specifications, theorem proving, text summaries, ontologies, rules, querying, medical documentation and planning. In 2004, ACE was adopted as the controlled natural language of the EU Network of Excellence REWERSE (Reasoning on the Web with Rules and Semantics)

7.2 Coherence method

Before analyzing the Coherence Method some definitions are necessary:

Different Path types

Topological paths are those paths that include all node in a determined path passing through a block and selecting only an alternative path whatever branch type encounter. In Figure 7.2 is presented a graph with its topological paths associated.

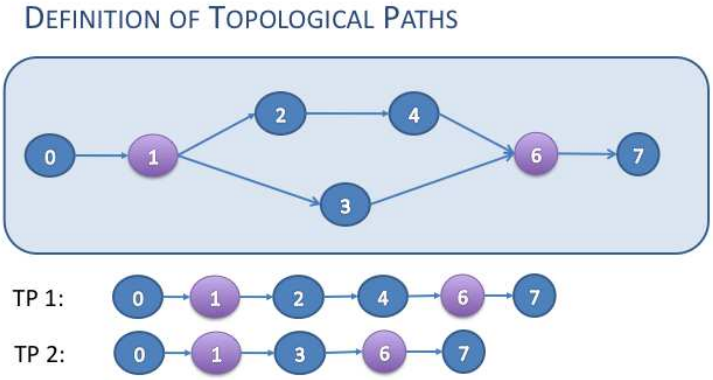


Figure 7.2: Topological Paths.

Semantic paths, instead, are those paths that include all node in a determined path passing through a block and selecting all the nodes that are semantically valid in relation to the branch type encountered. In Figure 7.3 is presented the same graph with its semantic paths associated. The branch is an AND type.

Different Branch types

There are three different types of branch studied:

- **AND**: every node between an AND branch and his merge point has to be visited to pass after the Merge Point.

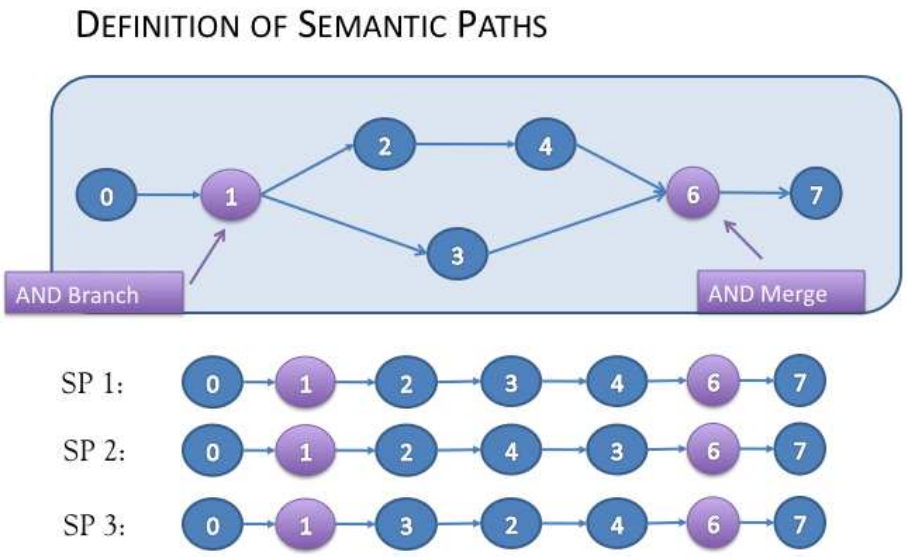


Figure 7.3: Semantic Paths.

- **OR**: every branch can be visited, but the first one that is completed stops the other and the control goes behind the Merge point.
- **XOR**: its a semaphore only the nodes of one branch are visited

Algorithm Operation

The algorithm used to control the coherence is composed by four steps

1. Find all the Topological Paths;
2. Find the blocks composed by same branch-merge point;
3. Compose the node to solve the block in partial Semantical Paths, applying the Rules during the composition;
4. Create the complete list of inconsistent Semantical Paths;

Algorithm Data

- Graph
- From Adjacency Matrix:
  - Vector: a[i] : edge number out from node i
  - Matrix: S[i,j] : j-esimo adjacent node to i

The graph used to show how the algorithm works is a semplification of the example in the chapter 5.3. The simplified process is presented in Figure 7.4, where is missing two branch, the initial one when the user decides what type of tool will use to select his object and the payment type branch. This semplification permits to better understand the algorithm created.

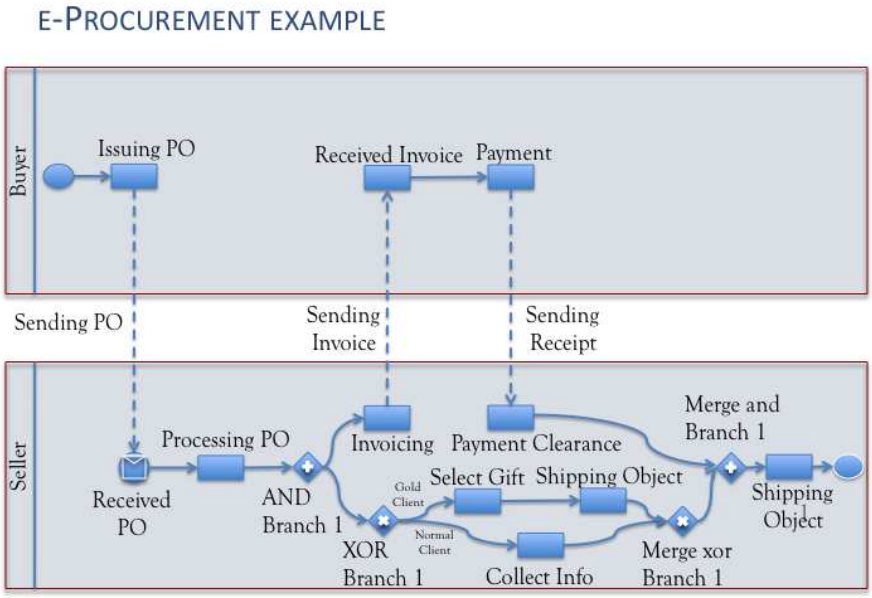


Figure 7.4: E-Procurement Process simplified Example.

From the Adjacency Matrix are derived two objects a Vector $a[i]$, that represents the edges going out from each node and the Matrix $S[i, j]$, that informs on the $j - esimo$ node adjacent to node i

Block Elaboration

Once, with a search in depth method, every topological path is found the next step is made of the search of the set of nodes having the same branch node and merge node. These sets are called blocks and they can be simple or composed. The simple blocks are the ones without other branch and merge nodes in their inside, while the composed are the blocks containing other blocks inside. This category can be of various complexity degree gaining a complexity level each time a block is nested inside another block. A block example is presented in Figure 7.5.

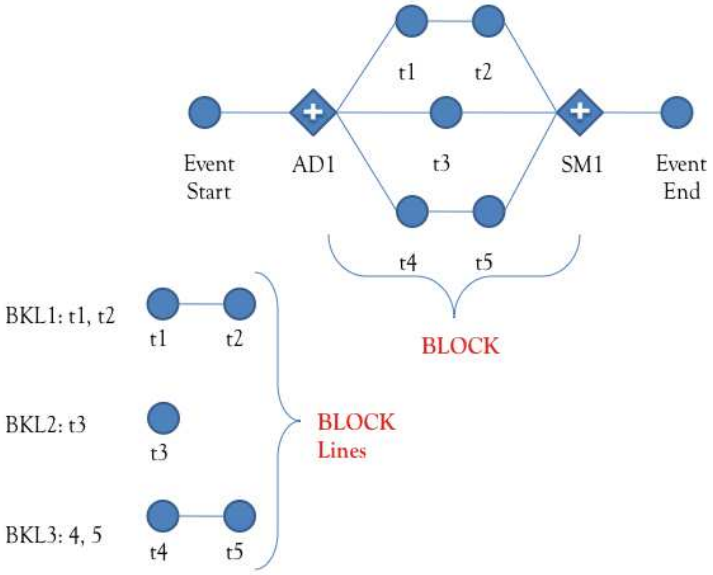


Figure 7.5: And Block.

The blocks are characterized not only from their complexity level but also from their typology, they can be of three types. They are presented in computational simplicity order:

- Xor
- And
- Or

Each type of block is elaborated in a different way. Once individuated every block type and their complexity level the algorithm starts solving the block with minimum complexity in simplicity order, first XOR type, then AND type and last OR type. In Figure 7.6 is presented the plain elaboration of the AND block of Figure 7.5.

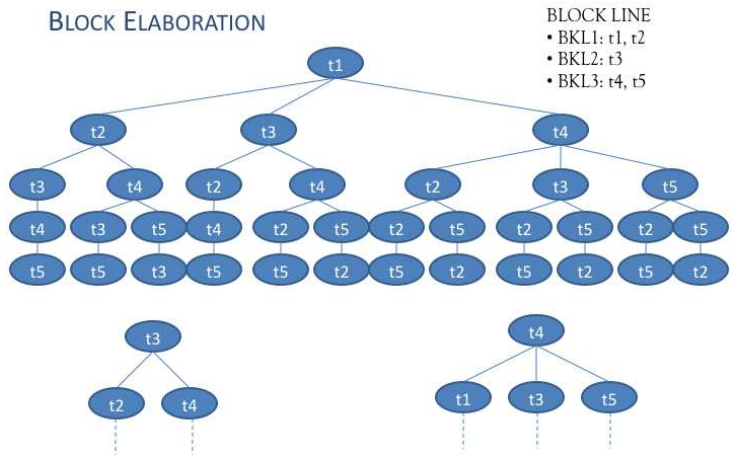


Figure 7.6: Block Elaboration.

The block elaboration is calculated dividing the block in different lines and starting creating a nodes tree with a root taken from the first node of each line strictly following the precedence order in the line. In this way it's, by default, calculated the precedence relation between the different nodes and is gained a computational advantage for not calculating every possible nodes combination.

Computational gain

In relation of what kind of block is considered there can be different computational gain. For the AND block of Figure 7.5 the possible combination number

of 5 elements is given from factorial combination of the elements. The result is 120 possible nodes sequences. With the block elaboration presented the possible combination is reduced to 30 different possibilities. 12 for the tree with $t1$ as root, 6 for the tree with $t3$ as root and 12 for the tree with $t4$ as root.

If the same nodes were part of a OR block the elaboration would be different, in fact the possible combination of 5 elements, following the OR semantics is given from the sum of the disposition of elements taken in group of 1, 2, 3, 4 and 5 elements. The result would be 325 possible nodes sequences. In Figure 7.7 is presented the computational gain for the same nodes set but inserted in an OR block.

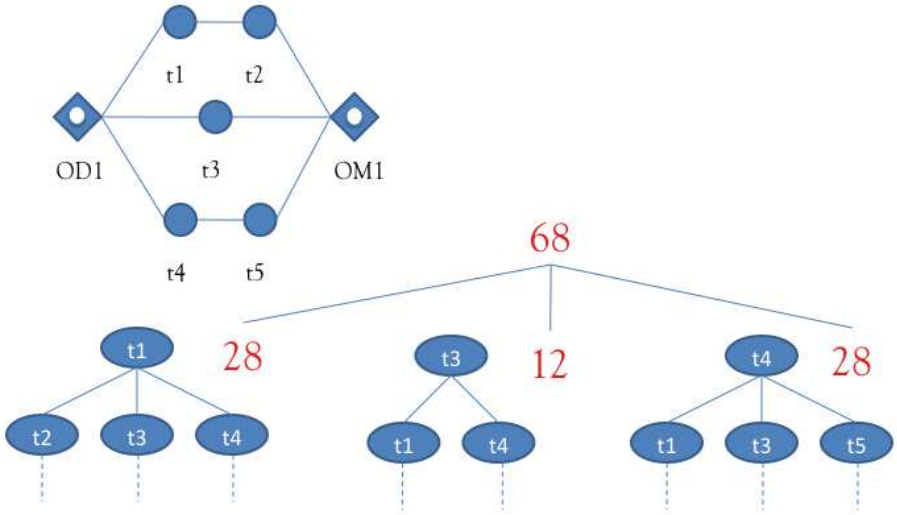


Figure 7.7: Computational Gain for an 'OR Branch'.

The simplest elaboration is relative of a XOR block. Given the XOR semaphoric semantics, the possible semantic paths created are only the single block lines.

After the recursive elaboration of every block presented in the graph they are reinserted in their place with a procedure that creates the semantic paths. If no violation is found the output is a no coherence problem business process and business rules applied.

The Business Rules are inserted in various moment of the elaboration. In the

next subsection are presented the way they are used.

Business Rules introduction and optimization

There are two moments where the Business Rule are controlled for their validity respect the elaborating Business Process. The first is during the elaboration of the single blocks in every recursive step. If theres a violation during the solving step of the block a branch of the making root is pruned out and there will be an output with the path reporting the semantic path with the problem and the violated rule. The second is during insertion of elaborated blocks inside the BP graph. Possible rules violations in semantic paths generation are reported as in the precedent case.

A series of optimization have been introduced to have better performance:

- Look for the presence of constrained task inside BP, if there are large part of BP without that task eliminate it from the input of algorithm.
- Unify, as super-task, tasks that are not affected by BRs.
- If in a part of a BP theres only a task affected by BR substitute that BP part with an equivalent simplify one.

Conclusion

In this work the focus has been on some important issues related to Business Process modeling, modifications and representation in an ontological form, there has been also the presentation of business rules and their interaction with business processes. It has been presented an ontological approach, where Business Processes and Business Rules can be formally represented in a unified context and their consistency checked and maintained by means of defined and developed tool. The main ideas for future works are relates to extend the control to other relations but the precedence, like existence, role and others. In this way can be used actors, objects and process in a unique algorithm input control BPAL graph.

Appendices

Appendix 1 - BPAL

Appendix on BPAL formalization

BPAL Example of a BP ontology for the Existence Verification process

act(request_for_existence_verification),
act(checking_the_requested_document),
act(releasing_certificate),
act(releasing_survey),
act(sending_document),
act(receiving_document),
role(company),
role(chamber of commerce),
role(legal office),
role(marketing department),
xdec(Is_The_Requested_Document_Legally_Valid, releasing_certificate),
cont(certificate),
cont(survey),
cont(request),
cont(existence_document),
prec(request for existence verification, receiving document),
prec(checking_the_requested_document, Is_The_Requested_Document_Legally_Valid),
prec(Is_The_Requested_Document_Legally_Valid, releasing_certificate),
prec(Is_The_Requested_Document_Legally_Valid, releasing_survey),
prec(releasing_certificate, sending_document),
prec(releasing_survey, sending_document)
perf(legal_office, request_for_existence_verification),

perf(marketing_department, checking_the_requested_document),
perf(marketing_department, releasing_certificate),
perf(marketing_department, releasing_survey),
perf(marketing_department, sending_document),
perf(legal_office, receiving_document),
msg(request, request_for_existence_verification, checking_the_requested_document),
msg(existence_document, sending_document, receiving_document),
pof(company, legal office),
pof(chamber of commerce, marketing department),
isa(existence_document, certificate),
isa(existence_document, survey).

Sampling of BPAL in KIF

Primitive Lexicon

The Lexicon is represented by the atoms reported in the Section 4 with a prefix syntax and variables preceded by the question mark, all enclosed in parenthesis. Example: $(act?a), (role?r), (dec?d)$.

KIF atoms

- **(act ?a)** is TRUE in an interpretation of BPAL if and only if ?a is a member of the set of activities in the universe of discourse of the interpretation. Intuitively, activities can be considered to be reusable behaviours within the domain.

- **(role ?r)** is TRUE in an interpretation of BPAL if and only if ?r is a member of the set of roles in the universe of discourse of the interpretation.

- **(dec ?d)** is TRUE in an interpretation of BPAL if and only if ?d is a member of the set of decisions in the universe of discourse of the interpretation.

- **(cont ?t)** is TRUE in an interpretation of BPAL if and only if ?t is a member of the set of contents in the universe of discourse of the interpretation.

- **(cxt ?c)** is TRUE in an interpretation of BPAL if and only if ?c is a member of the set of contexts in the universe of discourse of the interpretation.
- **(prec ?a1 ?a2)** is TRUE in an interpretation of BPAL if and only if the activity ?a1 is before ?a2 in the linear ordering over activities in the interpretation.
- **(xdec ?f ?a1 ?a2)** is TRUE in an interpretation of BPAL if and only if ?f is a function that maps sentences in truth values TRUE or FALSE, consequently, if the ?f maps to TRUE, activity ?a1 is executed otherwise activity ?a2 is executed.
- **(msg ?c ?a1 ?a2)** is TRUE in an interpretation of BPAL if and only if activity ?a1 sends a content ?c to activity ?a2.
- **(iter ?a1, ?an ?f)** is TRUE in an interpretation of BPAL if and only if the activities contained between ?a1 to ?an are repeated until the conditional ?f returns value TRUE.

AXIOMS

Axiom 1 (Branching)

Each time an activity is followed by more than one immediate successor activities, a decision atom must be interposed.

```
    (forall (?x ?y)
(if
(and  (act ?y)
(prec ?x ?y)
((abs ?y) ¿ 1))
(dec ?x)))
```

Axiom 2

The precedence relation only holds between activities and decisions.

```
    (forall (?x ?y)
(if (prec ?x ?y)
(or (and (act ?x) (dec ?y))
(and (act ?x) (act ?y))
```

(and (dec ?x) or (act ?y)))))

Axiom 3

Everything is either an activity, a role, a decision, a content, or a context.
(forall (?x)
(or (act ?x)
(role ?x)
(dec ?x)
(cont ?x)
(cxt ?x)))

Axiom 4

Activities, decisions, roles, contents, and contexts are all distinct kinds of things.
(forall (?x)
(and (if (act ?x)
(not (or (dec ?x) (role ?x) (cont ?x) (cxt ?x)))
(if (dec ?x)
(not (or (role ?x) (cont ?x) (cxt ?x)))
(if (role ?x)
(not (or (cont ?x) (cxt ?x)))
(if (cont ?x)
(not (cxt ?x)))))

Axiom 5

The msg function holds only between a content and activities or roles.
(forall (?c ?n1 ?n2)
(if (msg ?c ?n1 ?n2)
(and (cont ?c)
(or act ?n1 role ?n1)
(or act ?n2 role ?n2)))

Axiom 6

The xdec function holds only between an expression and an activity.
(forall (?f ?a1)
(if (xdec ?f ?a1)
(and (bexp ?f)
(act ?a1)))

Axiom 7. The iteration function holds only between activities and an expression.
(forall (?a1 ?a2 ?f)
(if (iter ?a1 ?a2 ?f)
(and
(act ?a1)
(act ?a2)
(bexp ?f))).

BPAL File

```
// OPAL
include procurement.opal
// Events
ev(_eventStart, _startingTime, _bexp(TRUE))
// Actions
act(_issuingPurchaseOrder_01)
// Messages
msg(_purchaseOrder, _sendingPO, _recivedPO, _eventRecivedPO)
// ISA Relations
isa(SendingPO, msg)
// Part OF Relations
pof(_sendToShippingCompany_01,_shippingObject_01)
// Fulfil Relations
fulfil(_issuingPurchaseOrder_01,_issuingPurchaseOrder)
// Precedent Relations
cpre(_eventStart, _issuingPurchaseOrder_01),
cpre(_issuingPurchaseOrder_01, _eventRecivedPO)
cpre(_eventRecivedPO, _processingPO_01)
cpre(_processingPO_01, _ADprocessingOrder)
```

```
// Gateway
adec(_ADprocessingOrder, _bexp(_processingPO_01))
xdec(_XDclientType, _bexp(_processingOrder))
odec(_ODclientType, _bexp(_processingOrder3))
andmerge(_SMprocessingOrder)
xormerge(_XMclientType)
ormerge(_OMpayment)
```

Appendix 2 - Software

In this Appendix is presented the principal UML class diagrams and the code of the application developed to control the coherence between Business Rules and Business Process.

UML class diagram

The Class Diagram in Figure A.1 is an important class that models the single node of the graph. It has, as auxiliary structures classes, the class A, an utility class, and the class S, model of the adjacent matrix and its functions. The Class A is presented in Figure A.2 and the Class S in the Figure A.3.

The algorithm is modeled in a class named Algoritmi and presented in the Figure A.4

Code

In this section is presented the program code.

Nodo.java

```
package Modello;

    /**
```

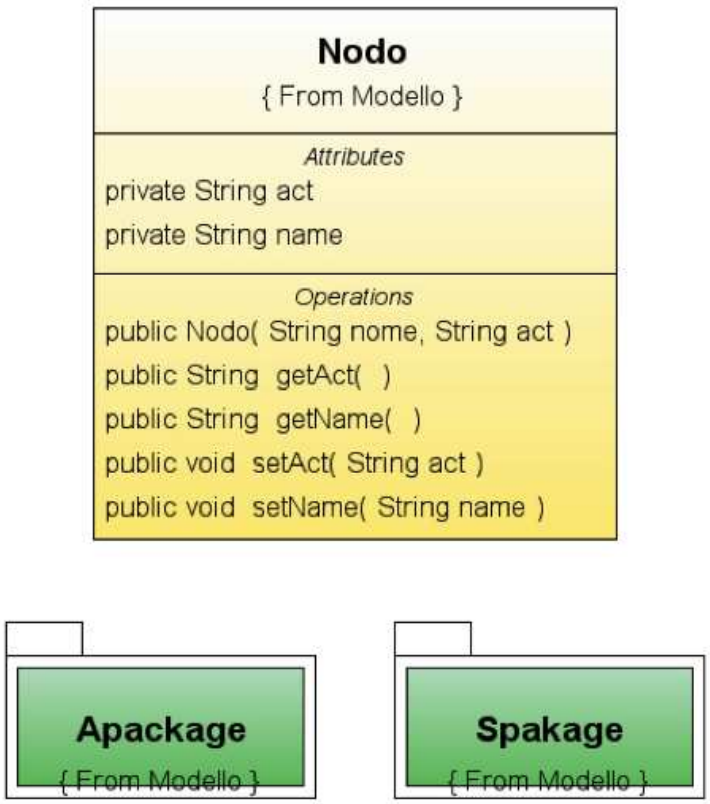


Figure A.1: Class Diagram Nodo.

BRANCH:
0 : no branch
1 : AND
2 : XOR
3 : OR

MERGE :
0 : No merge
1 : Sync Merge
2 : Xor Merge
3 : Or Merge
/

```
public class Nodo {
private String act; // Tipologia dell'azione relativa a quel nodo
private String name; // Nome del nodo
private int branch;
private int merge;
public float x;
public float y;

    public Nodo(String nome, String act){
this.name=nome;
this.act=act;
this.branch=0;
this.merge=0;
x=0;y=0;
}

    public Nodo(String nome, String act,int branch, int merge){
this.name=nome;
this.act=act;
this.branch=branch;
this.merge=merge;
x=0;y=0;
}

    public int getMerge() {
return merge;
}

    public int getBranch() {
return branch;
}

    public void setMerge(int merge) {
this.merge = merge;
}

    public void setBranch(int branch) {
this.branch = branch;
}
```

```
public String getAct() {
return act;
}

    public String getName() {
return name;
}

    public void setAct(String act) {
this.act = act;
}

    public void setName(String name) {
this.name = name;
}

    }
```

Block.java

```
package Modello;

    import java.util.LinkedList;
import java.util.List;

    public class Block {
public List blockPaths;
public int blockType;
public List outPaths;
public String BranchName;
public String MergeName;
public int ComplexDegree;
public int blockFlag;
public boolean isReaded;

    public Block() {
this.blockPaths = new LinkedList();
this.blockType=0;
```

```
this.outPaths=new LinkedList();
this.BranchName="";
this.MergeName="";
this.ComplexDegree=0;
this.blockFlag=0;
this.isReaded=false;

    }

    public Block(Block b) {
this.blockPaths = new LinkedList();
this.blockType=0;
this.outPaths=new LinkedList();
this.BranchName="";
this.MergeName="";
this.ComplexDegree=0;
this.blockFlag=0;
this.isReaded=false;
this.BranchName=b.BranchName;
this.ComplexDegree=b.ComplexDegree;
this.MergeName=b.MergeName;
this.blockPaths.add(b.blockPaths);
this.blockType=b.blockType;
this.blockFlag=b.blockFlag;

    }

    }
```

BlockList.java

```
package Modello;

    import java.util.LinkedList;
import java.util.List;

    public class BlockList {
public List ListaBlocchi;
```

```
    public BlockList() {
this.ListaBlocchi= new LinkedList();
}

    public void addPathToBlock(Block b) {
Block bl;

    for (int i=0;i¡this.ListaBlocchi.size();i++) {
bl=(Block) this.ListaBlocchi.get(i);
if (bl.BranchName.equals(b.BranchName))bl.blockPaths.add(b.blockPaths);}

    }
}

    public void calculateComplex() {
Block b;
int complex=0;
List branchs ;
List path;
Nodo n;
for (int i=0;i¡this.ListaBlocchi.size();i++){ // per ogni blocco
branchs = new LinkedList();
complex=0;
b = (Block) this.ListaBlocchi.get(i);
for (int j=0;j¡b.blockPaths.size();j++){ // per ogni path del blocco
path = (List) b.blockPaths.get(j);
for (int k=0;k¡path.size();k++){ // per ogni nodo del path del blocco
n = (Nodo) path.get(k);
if ((n.getBranch()!=0)&&(!isNodePresenseInBranches(n.getName(),branchs)))
{
branchs.add(n.getName());
complex++;

    }

    }
}
b.ComplexDegree=complex;
```

```
}

    }

    public boolean isPresent(Block b) {
Block bl;
boolean trovato=false;
for (int i=0;i¡this.ListaBlocchi.size();i++){
bl=(Block) this.ListaBlocchi.get(i);
if (bl.BranchName.equals(b.BranchName))trovato= true;}

    }
return trovato;
}

    private boolean isNodePresenseInBranches(String name, List bran) {
boolean res = false;
String s;
for (int i=0;i¡bran.size();i++){
s= (String) bran.get(i);
if(name.equals(s)) {res=true;}
}
return res;
}

    }
```

A.java

```
package Modello.Apackage;

    import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

    public class A {
private List archiUscenti;
```

```
    public A() {
this.archiUscenti=new LinkedList();
}

    public List getArchiUscenti() {
return archiUscenti;
}

    public void insertElement(AElement element) {
this.archiUscenti.add(element);
}

    public int getNumeroArchi(String nomeNodo) {
AElement ae ;
int res=0;
for (int i=0;i¡this.archiUscenti.size();i++) {
ae = (AElement) this.archiUscenti.get(i);
if (ae.getName().equals(nomeNodo)) res=ae.getNumeroArchi(); }
}
return res;
}

    public void incrementaArchiUscenti (String nomeNodo){
Iterator it = this.archiUscenti.iterator();
AElement n;
boolean esiste = false;
while (it.hasNext()){
n = (AElement) it.next();
if (nomeNodo.equals(n.getName())) {
n.setNumeroArchi(n.getNumeroArchi()+1);
esiste = true;
}
}
if (!esiste) {
n = new AElement(nomeNodo,1);
archiUscenti.add(n);
}
}
```

```
    public void inizializzaD (int numeroNodi) {
AElement n;
for (int i=0;i¡numeroNodi;i++) {

    }
}

    public AElement getNodoTerminale(){
AElement ae;
AElement aeTrovato=null;
boolean trovato=false;

    for (int i=0;i¡this.archiUscenti.size();i++) {
ae = (AElement) this.archiUscenti.get(i);
if ((ae.getNumeroArchi()==0)&&(!trovato)) {trovato=true;aeTrovato=ae;}
}
return aeTrovato;
}

    public AElement getNodoXNome(String nome){
AElement ae;
AElement aeTrovato=null;
boolean trovato=false;

    for (int i=0;i¡this.archiUscenti.size();i++){
ae = (AElement) this.archiUscenti.get(i);
if ((ae.getName().equals(nome))&&(!trovato)) trovato=true;aeTrovato=ae;}
}
return aeTrovato;
}

    }
```

AElement.java

```
package Modello.Apackage;

    public class AElement {
```

```
private String name;
private int numeroArchi;

    public AElement(String nome, int na){
this.name=nome;
this.numeroArchi=na;
}

    public String getName() {
return name;
}

    public int getNumeroArchi() {
return numeroArchi;
}

    public void setName(String name) {
this.name = name;
}

    public void setNumeroArchi(int numeroArchi) {
this.numeroArchi = numeroArchi;
}

    }
```

S.java

```
package Modello.Spakage;

    import java.util.LinkedList;
import java.util.List;

    public class S {
private List s;

    public S(){
s= new LinkedList();
```

```
}

    public void addSElement(SElement se){
this.s.add(se);
}

    public List getAdiacenzeXNodo(String nome){
List res=null;
SElement se;

    for (int i=0;i¡this.s.size();i++){
se =(SElement) this.s.get(i);
if (nome.equals(se.getName())) {res=se.getAdiacenze();}
}
return res;
}

    public void setAdiacenzeXNodo (String nomeNodo, String nomeNodoAdi-
acente) {
SElement se, se_appo;
boolean esiste = false;

    for (int i=0;i¡this.s.size();i++){
se =(SElement) this.s.get(i);
if (nomeNodo.equals(se.getName())) {
se.addAdiacenza(nomeNodoAdiacente);
esiste = true;
}
}
if (!esiste){
se_appo = new SElement(nomeNodo);
se_appo.addAdiacenza(nomeNodoAdiacente);
s.add(se_appo);
}
}

    public List getS() {
return s;
}
```

```
    }
```

SElement.java

```
package Modello.Spakage;

    import java.util.LinkedList;
import java.util.List;

    public class SElement {
private String name;
private List adiacenze;

    public SElement() {
this.name="";
}

    public SElement(String name){
this.name=name;
this.adiacenze=new LinkedList();
}

    public void addAdiacenza(String nodo){
this.adiacenze.add(nodo);
}

    public List getAdiacenze() {
return adiacenze;
}

    public String getName() {
return name;
}

    public void setAdiacenze(List adiacenze) {
this.adiacenze = adiacenze;
}
```

```
    }
```

Algoritmi.java

```
package Algoritmi;

    import Modello.Apackage.A;
import Modello.Apackage.AElement;
import Modello.Block;
import Modello.BlockList;
import Modello.Nodo;
import Modello.Spakage.S;
import Modello.Spakage.SElement;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

    import java.util.logging.Level;
import java.util.logging.Logger;

    public class Algoritmi {

    private List ListaPath;
private BlockList Blocks;
private BlockList BlockGroups;
private List result;

    public Algoritmi() {
this.ListaPath = new LinkedList();
this.Blocks = new BlockList();
this.BlockGroups = new BlockList();
this.result = new LinkedList();
```

```
}

    public void ScriviFile(String nomeFile, List path) {
FileOutputStream fos = null;
try {
File f = new File(nomeFile);
fos = new FileOutputStream(f, true);
PrintStream ps = new PrintStream(fos);
ps.println(" ");
ps.print("Path: [");
ps.print(((Nodo) path.get(0)).getName());
for (int i = 1; i ¡ path.size(); i++) {
// Inserito il getName perche path.get(i) restituisce un nodo
ps.print(", " + ((Nodo) path.get(i)).getName());
}
ps.print("]");
} catch (FileNotFoundException ex) {
Logger.getLogger(Algoritmi.class.getName()).log(Level.SEVERE, null, ex);
} finally {
try {
fos.close();
} catch (IOException ex) {
Logger.getLogger(Algoritmi.class.getName()).log(Level.SEVERE, null, ex);
}
}
}

    private void ControllaInserimentoPath(Block blocco) {

    System.out.println("————-¿ path presa in esame: " + blocco.BranchName
+ " ¡-¿ " + blocco.MergeName);
List t = (List) blocco.blockPaths;
for (int h = 0; h ¡ t.size(); h++) {
Nodo g = (Nodo) t.get(h);
System.out.print(g.getName() + ",");
}
if (this.Blocks.ListaBlocchi.size() == 0) {
this.Blocks.ListaBlocchi.add(blocco);
System.out.println(" inserita!");
```

```
} // il primo elemento
else {
//Devo scorrere la lista dei Blocchi
boolean uguaglianzaPath = false;
int uguaglianzaConPaths = 0;
for (int i = 0; i ¡ this.Blocks.ListaBlocchi.size(); i++) {
Block b = (Block) this.Blocks.ListaBlocchi.get(i); // per ogni block di Blocks
prendere il path
if ((b.BranchName.equals(blocco.BranchName)) && (b.MergeName.equals(blocco.MergeName))
&& (b.blockPaths.size() == blocco.blockPaths.size())) {
//Ne controllo il nome e la lunghezza Se uguali allora passo al controllo dei
path
uguaglianzaPath = true;
for (int j = 0; j ¡ b.blockPaths.size(); j++) {
//Controllare che il path sia diverso da quello che voglio inserire nodo per nodo
Nodo b_Nodo = (Nodo) b.blockPaths.get(j);
Nodo blocco_Nodo = (Nodo) blocco.blockPaths.get(j);
if (!(b_Nodo.getName().equals(blocco_Nodo.getName()))) {
//Se i Nodi sono diversi allora i path sono diversi ed esco dalla for
uguaglianzaPath = false;
break;
}
}
if (uguaglianzaPath) {
uguaglianzaConPaths++;
}
}
}
if (uguaglianzaConPaths == 0) {
this.Blocks.ListaBlocchi.add(blocco);
System.out.println(" inserita!");
} else {
System.out.println(" NO!");
}
}
}

    private void StampaPathSemantici() {
```

```
    for (int i = 0; i ¡ this.result.size(); i++) {
System.out.println("Path Semantica n: " + i);
List l = (List) this.result.get(i);
for (int j = 0; j ¡ l.size(); j++) {
Nodo n = (Nodo) l.get(j);
System.out.print(n.getName() + ",");
}
System.out.println("");
}
}

    private List clona(List l) {
List a = new LinkedList();
for (int i = 0; i ¡ l.size(); i++) {
Nodo n = (Nodo) l.get(i);
a.add(n);
}
return a;
}

    private Nodo getNodoDaListaNodiXNome(String nome, List nodi) {
Nodo res = null;
boolean trovato = false;
Nodo app;
for (int i = 0; i ¡ nodi.size(); i++) {
app = (Nodo) nodi.get(i);

    // if (res.getName().equals(nome)) {
if (app.getName().equals(nome)) {
trovato = true;
res = app;
}
}
return res;
}

    public void VisitaEsaustivaSemplice(List ln, A la, A ld, S ls) { // Crea i
path topologici
List listaNodi = ln;
```

```
A listaA = la;
A listaD = ld;
S matriceS = ls;
List path = new LinkedList();

    int l = 0; // Lunghezza cammino corrente

    int z = 1; // Switch che dice se il cammino percorso andando avanti o in-
dietro

    Nodo t = new Nodo("nome", "act"); // Nodo corrente

    AElement nodoTerminale = listaA.getNodoTerminale();

    System.out.println(" ————————— VISITA ESAUSTIVA SEMPLICE IN-
IZIO ————— ");
// Inizializzazione v[t]
A V = new A();
for (int i = 0; i ¡ listaNodi.size(); i++) {
AElement newAe = new AElement(((Nodo) listaNodi.get(i)).getName(), 1);
V.insertElement(newAe);
}

    t = (Nodo) listaNodi.get(0);

    while (l ¿= 0) {

    if (z == 1) {
if (t.getName().equals(nodoTerminale.getName())) // Sono arrivato alla fine
{
path.add(t);
List pathTrue = new LinkedList();
this.ScriviFile("VisitaEsaustivaSemplice.txt", path);
System.out.println("path:");
for (int ji = 0; ji ¡ path.size(); ji++) {
Nodo no = (Nodo) path.get(ji);
System.out.print(no.getName() + ",");
pathTrue.add(no);
}
```

```
    System.out.println("");
this.ListaPath.add(pathTrue);
z = 0;
path.remove(path.size() - 1);
// Nel path.get(path.size() - 1)) ho aggiunto il .getName() per tirar fuori la
stringa
if (l ¿ 0) {
t = this.getNodoDaListaNodiXNome((((Nodo) path.get(path.size() - 1)).get-
Name()), listaNodi);
}
l–;

    } else {
if (V.getNodoXNome(t.getName()).getNumeroArchi() == 0) //Dal nodo cor-
rente non arrivo al terminale
{
z = 0;
l–;
} else {
path.add(l, t);
int NumeroarchiUscentiVisitati = ((AElement) listaD.getNodoXNome(t.getName())).getNumeroArchi();
List nodiAdiacentiAlNodoCorrente = (matriceS.getAdiacenzeXNodo(t.getName()));

    String nomeNodoSuccessivo = (String) nodiAdiacentiAlNodoCorrente.get(NumeroarchiUscentiVisitat
Nodo app_t = this.getNodoDaListaNodiXNome(nomeNodoSuccessivo, listaN-
odi);
// app_t = S[t,d[t]]; //Prendo il t-esimo nodo adiacente al nodo corrente
l++;
((AElement) listaD.getNodoXNome(t.getName())).setNumeroArchi(((AElement)
listaD.getNodoXNome(t.getName())).getNumeroArchi() + 1); // d[t]++;

    t = app_t;
}
}
} else {

    if (((AElement) listaD.getNodoXNome(t.getName())).getNumeroArchi() ¡
((AElement) listaA.getNodoXNome(t.getName())).getNumeroArchi()) // if (d[t]
```

```
¡ a[t]) // "¡" vuol dire che posso ancora andare avanti e che non ho esaurito la
verifica di tutti i nodi uscenti
{
// t = S[t,d[t]]; // Vado avanti
int NumeroarchiUscentiVisitati = ((AElement) listaD.getNodoXNome(t.getName())).getNumeroArchi();

    List lista = matriceS.getAdiacenzeXNodo(t.getName());

    String nomeNodoSuccessivo = (String) lista.get(NumeroarchiUscentiVisitati);
((AElement) listaD.getNodoXNome(t.getName())).setNumeroArchi(((AElement)
listaD.getNodoXNome(t.getName())).getNumeroArchi() + 1);
t = this.getNodoDaListaNodiXNome(nomeNodoSuccessivo, listaNodi);

    l++;
z = 1;
} else {

    ((AElement) listaD.getNodoXNome(t.getName())).setNumeroArchi(0);
path.remove(path.size() - 1);
//d[t] = 0; // La volta successiva che si ripassera dal nodo provenendo da
un'altra direzione si riniziera l'esplorazione di tutti gli archi a partire da quel
nodo.
// Nel path.get(path.size() - 1)) ho aggiunto il .getName() per tirar fuori la
stringa
if (l ¿ 0) {
t = this.getNodoDaListaNodiXNome((((Nodo) path.get(path.size() - 1)).get-
Name()), listaNodi);
}
//t = path[l - 1];
l–;
}
}
}
System.out.println(" ———————— VISITA ESAUSTIVA SEMPLICE FINE
———————— ");
SemanticPath();
}

    private void SemanticPath() {
```

```
System.out.println(" ———————— INDIVIDUAZIONE DEI BLOCCHI INIZIO ———————— ");
getBlocksFromPaths();// blocks e la struttura che contiene tutte le path con il blocco ad esse associato
StampaBlocchi();
System.out.println(" ———————— INDIVIDUAZIONE DEI BLOCCHI FINE ———————— ");
SolveBlocks();

    }

    private void getBlocksFromPaths() {// Dalle path topologiche passiamo ai blocchi
BlockList blocchi = new BlockList();
Block blocco;

    boolean stop = false;
List pathTopologica;
Nodo nodo, nodoApp;

    for (int i = 0; i ¡ this.ListaPath.size(); i++) { // Scorrimento delle path topologice
pathTopologica = (List) this.ListaPath.get(i); // per ognuna delle quali

    for (int j = 0; j ¡ pathTopologica.size(); j++) { // Scorriamo tutti i nodi della path topologica
nodo = (Nodo) pathTopologica.get(j);// per ogni nodo controllo se
if (nodo.getBranch() != 0)  //  un nodo di Branch
blocco = new Block(); // creo un nuovo blocco
blocco.blockType = nodo.getBranch();// inserisco la tipologia del blocco
blocco.BranchName = nodo.getName(); // chiamo il blocco con il nome del nodo branch
//blocco.ComplexDegree=1; // setto la complessita
blocco.blockFlag = 1; // ed il contatore dei Branch
int scorriblocco = j; // poi scorro la sub path
while (blocco.blockFlag ¿ 0)  // finche non trovo il merge di questo branch
scorriblocco++;
nodoApp = (Nodo) pathTopologica.get(scorriblocco); // prendo il prossimo nodo
```

```
if (nodoApp.getBranch() != 0) {
blocco.ComplexDegree++; // e un branch
blocco.blockFlag++;
}
if (nodoApp.getMerge() != 0) {
blocco.blockFlag–;
}
if (blocco.blockFlag != 0) {
blocco.blockPaths.add(nodoApp);

    }
if (blocco.blockFlag == 0) {
blocco.MergeName = nodoApp.getName();
ControllaInserimentoPath(blocco); // blocks e la struttura che contiene tutte
lepath con il blocco ad esse associato
}

    }
}

    }

    }

    }

    private void StampaBlocchi() {
Block b;
Nodo n;
System.out.println("");
System.out.println("———————— STAMPA DELLA STRUTTURA BLOCCHI
————————-");
for (int i = 0; i ¡ this.Blocks.ListaBlocchi.size(); i++) {
b = (Block) this.Blocks.ListaBlocchi.get(i);
System.out.println("Blocco: " + b.BranchName + "¡-¿" + b.MergeName + "
complessita del path topologico: " + b.ComplexDegree + " tipologia blocco:
" + b.blockType);
System.out.print(" ");
for (int j = 0; j ¡ b.blockPaths.size(); j++) {
```

```
n = (Nodo) b.blockPaths.get(j);
System.out.print(n.getName() + "-");
}
System.out.println("");
}
}

    private void SolveBlocks() {

    System.out.println(" ————————— CREAZIONE DI BLOCK GROUPS
INIZIO ————— ");
Block block; // Generazione della struttura BlockGroups
Block b;
for (int i = 0; i ¡ this.Blocks.ListaBlocchi.size(); i++) {
b = (Block) this.Blocks.ListaBlocchi.get(i);
if ((!b.isReaded) && (!this.BlockGroups.isPresent(b))) {
block = new Block(b);
b.isReaded = true;
this.BlockGroups.ListaBlocchi.add(block);
}
if ((!b.isReaded) && (this.BlockGroups.isPresent(b))) {
b.isReaded = true;
this.BlockGroups.addPathToBlock(b);
}
}

    // Calcolo della complessita dei singoli blocchi
this.BlockGroups.calculateComplex();

    Nodo nodo;
List path;
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
block = (Block) this.BlockGroups.ListaBlocchi.get(i);
System.out.println("blocco: " + block.BranchName + " ¡-¿ " + block.MergeName
+ " tipologia: " + block.blockType + " complessita : " + block.ComplexDegree);
System.out.println(" path coinvolte nel blocco: " + block.blockPaths.size());
for (int k = 0; k ¡ block.blockPaths.size(); k++) {
path = (List) block.blockPaths.get(k);
System.out.print(" Path: ");
```

```
for (int j = 0; j ¡ path.size(); j++) {
nodo = (Nodo) path.get(j);
System.out.print(nodo.getName() + "-");
}
System.out.println("");

    }
}
System.out.println(" ———————— CREAZIONE DI BLOCK GROUPS FINE
————————— ");
// Inizio della solve
solve();
System.out.println("");

    // StampaSemanticPaths();
}

    public void solve() {
// Carichiamo i blocchi di complessita 0 che siano XOR togliendo // I) Se ci
sono blocchi XOR di complessita 0 dobbiamo andare a togliere i nodi branch/merge
di tale blocchi nei
// blocchi (di complessita quindi ¿0) nei quali compare il blocco XOR preso in
esame
Block blocco;
String BranchStart = "", MergeEnd = "";
List path;
Nodo n;
System.out.println("");
System.out.println("————————— ELABORAZIONE BLOCCHI XOR ————————
———-");
boolean esistonoBlocchiXorInComp_0 = true;
while (esistonoBlocchiXorInComp_0) {
esistonoBlocchiXorInComp_0 = false;
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
if ((blocco.ComplexDegree == 0) && (blocco.blockType == 2)) {// Blocco
XOR di complessita 0
BranchStart = blocco.BranchName;
MergeEnd = blocco.MergeName;
```

```
esistonoBlocchiXorInComp_0 = true;
blocco.ComplexDegree–;
System.out.println("Trovato blocco xor fra: " + BranchStart + "¡-¿" + MergeEnd);
}
}

    for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
System.out.println(" Ricerca del blocco Xor nel blocco : " + blocco.BranchName
+ "¡-¿" + blocco.MergeName);
for (int j = 0; j ¡ blocco.blockPaths.size(); j++) {
path = (List) blocco.blockPaths.get(j);
System.out.println(" Ricerca del blocco Xor nella path : " + j);
int indiceBranch = 0;
int indiceMerge = 0;
boolean trovato = false;
for (int k = 0; k ¡ path.size(); k++) {
n = (Nodo) path.get(k);
if ((n.getName().equals(BranchStart))) {
indiceBranch = k;
trovato = true;
}
if (n.getName().equals(MergeEnd)) {
indiceMerge = k;
}
}
if (trovato) {
blocco.ComplexDegree–; // aggiornamento della complessita del blocco contenente il blocco xor
path.remove(indiceMerge);
path.remove(indiceBranch);
System.out.println(" Elaborazione del blocco Xor ai nodi : " + indiceBranch
+ " e: " + indiceMerge);

    }
}
}
}
System.out.println("———— ELABORAZIONE BLOCCHI XOR FINE!—
```

————————-");
System.out.println("");
System.out.println("————— ELABORAZIONE BLOCCHI AND ————————");
// RICERCA DEL BLOCCO AND CON COMPLESSITA 0
boolean esistonoBlocchiANDInComp_0 = true;
while (esistonoBlocchiANDInComp_0) {
esistonoBlocchiANDInComp_0 = false;
List permutazioniBloccoAND = null;
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
if ((blocco.ComplexDegree == 0) && (blocco.blockType == 1)) {// Blocco XOR di complessita 0
BranchStart = blocco.BranchName;
MergeEnd = blocco.MergeName;
blocco.ComplexDegree–;
esistonoBlocchiANDInComp_0 = true;
System.out.println("Trovato blocco AND fra: " + BranchStart + "¡-¿" + MergeEnd);
Permutazioni permutazioni = new Permutazioni(blocco.blockPaths);
permutazioniBloccoAND = permutazioni.executeANDPermutations();
blocco.blockPaths.clear();
for (int o=0;o¡permutazioniBloccoAND.size();o++){
blocco.blockPaths.add(permutazioniBloccoAND.get(o));
}

}
}
// SOSTITUZIONE DELLE PERMUTAZIONI NEGLI ALTRI BLOCCHI
// RICERCA DELLE PATH CHE PRESENTANO QUEL BLOCCO AND

for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
List indici = new LinkedList();
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
System.out.println(" Ricerca del blocco AND nel blocco : " + blocco.BranchName + "¡-¿" + blocco.MergeName);
for (int j = 0; j ¡ blocco.blockPaths.size(); j++) {
path = (List) blocco.blockPaths.get(j);
System.out.println(" Ricerca del blocco AND nella path : " + j);

```
for (int t = 0; t ¡ path.size(); t++) {
Nodo nodo = (Nodo) path.get(t);
if (nodo.getName().equals(BranchStart)) {
indici.add(j);
System.out.println("occorrenza trovata !!!");
}
}
}
if (indici.size() ¿ 0) {
System.out.println(" In questo blocco sono state trovate :" + indici.size() + "
path contenenti quel blocco");
sostituisciPermutazioni(blocco, BranchStart, MergeEnd, (Integer) indici.get(0),
permutazioniBloccoAND);
for (int t = indici.size(); t ¿ 0; t–) { // rimozione di tutte le path che sono
state sostituite int rimuovi = (Integer) indici.get(t - 1);
blocco.blockPaths.remove(rimuovi);

    }
}
blocco.blockPaths.addAll(blocco.outPaths);
blocco.outPaths.clear();
}

    }
System.out.println("————— ELABORAZIONE BLOCCHI AND FINE —
————————");

    System.out.println("————— ELABORAZIONE BLOCCHI OR ———
————————");
// RICERCA DEL BLOCCO OR CON COMPLESSITA 0
boolean esistonoBlocchiORInComp_0 = true;
while (esistonoBlocchiORInComp_0) {
esistonoBlocchiORInComp_0 = false;
List permutazioniBloccoOR = null;
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
if ((blocco.ComplexDegree == 0) && (blocco.blockType == 3)) {
BranchStart = blocco.BranchName;
MergeEnd = blocco.MergeName;
```

```
blocco.ComplexDegree–;
esistonoBlocchiORInComp_0 = true;
System.out.println("Trovato blocco OR fra: " + BranchStart + "¡-¿" + MergeEnd);
Permutazioni permutazioniOR = new Permutazioni(blocco.blockPaths);
permutazioniBloccoOR = permutazioniOR.executeORPermutations();
blocco.blockPaths.clear();
for (int o=0;o¡permutazioniBloccoOR.size();o++){
blocco.blockPaths.add(permutazioniBloccoOR.get(o));
}
System.out.println("");
}
}
// SOSTITUZIONE DELLE PERMUTAZIONI NEGLI ALTRI BLOCCHI
// RICERCA DELLE PATH CHE PRESENTANO QUEL BLOCCO AND

    for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
List indici = new LinkedList();
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
System.out.println(" Ricerca del blocco OR nel blocco : " + blocco.BranchName
+ "¡-¿" + blocco.MergeName);
for (int j = 0; j ¡ blocco.blockPaths.size(); j++) {
path = (List) blocco.blockPaths.get(j);
System.out.println(" Ricerca del blocco OR nella path : " + j);
for (int t = 0; t ¡ path.size(); t++) {
Nodo nodo = (Nodo) path.get(t);
if (nodo.getName().equals(BranchStart)) {
indici.add(j);
System.out.println("occorrenza trovata !!!");
}
}
}
if (indici.size() ¿ 0) {
System.out.println(" In questo blocco sono state trovate :" + indici.size() + "
path contenenti quel blocco");
sostituisciPermutazioni(blocco, BranchStart, MergeEnd, (Integer) indici.get(0),
permutazioniBloccoOR);
for (int t = indici.size(); t ¿ 0; t–) { // rimozione di tutte le path che sono
state sostituite
int rimuovi = (Integer) indici.get(t - 1);
```

```
blocco.blockPaths.remove(rimuovi);

    }
}
blocco.blockPaths.addAll(blocco.outPaths);
blocco.outPaths.clear();
}

    }
System.out.println("————— ELABORAZIONE BLOCCHI OR FINE ——
————————");
System.out.println("————— ELABORAZIONE PATH SEMANTICI DEFINI-
TIVI ——————————");
// Prendo il nome del primo branch nelle topological path che sara il blocco
piu largo
// e quindi quello che andra sostituito attraverso le semantic path trovate
List primaPathTopologica = (List) this.ListaPath.get(0);
// scorro per prendermi sinistra destra branch e merge

    String Branch = "";
String Merge = "";
List sinistra = new LinkedList();
int stato = 0; // 0=immissione normale; ¿1=salto
boolean isResultList = false;// flag per la prima esecuzione -¿ aggiungo il nodo
ad una lista non a tutte le liste
boolean stop = false; // mi avvisa ogni volta che finisce un blocco -¿ ogni volta
che lo stato torna a zero dopo un merge
for (int i = 0; i ¡ primaPathTopologica.size(); i++) {

    Nodo nodo = (Nodo) primaPathTopologica.get(i);
if (nodo.getBranch() == 0) // e un nodo generico
if (stato == 0) {
if (!isResultList) {
sinistra.add(nodo);
}// se non sono in un blocco aggiungo il nodo al result
else {
aggiungiResult(nodo);
}
}
```

```
    }
if (nodo.getBranch() != 0)  // e un nodo di branch
if (!isResultList) {
this.result.add(sinistra); // se non ho incontrato ancora nessun branch allora
metto sinistra nella lista dei risultato
isResultList = true;
} // e avviso il sistema che la 'sinistra' e terminata
if (stato == 0) {
Branch = nodo.getName();
}
stato++;

    }
if (nodo.getMerge() != 0) // e un nodo di merge
stato–;
if (stato == 0) {
Merge = nodo.getName();
stop = true;
}
}
if (stop) {
getBlocco(Branch, Merge);
stop = false;
}
}
StampaPathSemantici();
}

    private void getBlocco(String Branch, String Merge) {
Block blocco = null;
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
blocco = (Block) this.BlockGroups.ListaBlocchi.get(i);
if ((blocco.BranchName.equals(Branch)) && (blocco.MergeName.equals(Merge)))
{
break;
}
}
List resultApp = new LinkedList();
```

```
for (int i = 0; i ¡ this.result.size(); i++) {// scorro le path di result
List l = (List) this.result.get(i);

    for (int j = 0; j ¡ blocco.blockPaths.size(); j++)  // a ognuna delle quali
aggiungo le path che espletano il blocco
List k = clona(l);
k.addAll((List) blocco.blockPaths.get(j));
resultApp.add(k);

    }
}
this.result = resultApp;
}

    private void aggiungiResult(Nodo nodo) {
for (int i = 0; i ¡ this.result.size(); i++) {
List l = (List) this.result.get(i);
l.add(nodo);
}
}

    private void sostituisciPermutazioni(Block blocco, String BranchStart, String
MergeEnd, int indice, List permutazioni) {
List path = (List) blocco.blockPaths.get(indice);
int indiceBranch = 0;
int indiceMerge = 0;
List sinistra = new LinkedList();
List destra = new LinkedList();
int sez = 0;
boolean trovato = false;
for (int k = 0; k ¡ path.size(); k++) {
Nodo n = (Nodo) path.get(k);

    if (sez == 1) {
destra.add(n);
}
if ((n.getName().equals(BranchStart))) {
indiceBranch = k;
trovato = true;
```

```
sez = -1;
}
if (n.getName().equals(MergeEnd)) {
indiceMerge = k;
sez = 1;
}
if (sez == 0) {
sinistra.add(n);
}
}
if (trovato) {
blocco.ComplexDegree–; // aggiornamento della complessita del blocco contenente il blocco xor
//————————
// Creazione delle paths permutate
for (int i1 = 0; i1 ¡ permutazioni.size(); i1++) {
List appoggio = new LinkedList();
appoggio.addAll(sinistra);
appoggio.addAll((List) permutazioni.get(i1));
appoggio.addAll(destra);
blocco.outPaths.add(appoggio);
}

    System.out.println(" Elaborazione del blocco AND ai nodi : " + indiceBranch + " e: " + indiceMerge);

    }
}

    private void StampaSemanticPaths() {
System.out.println("————- PATHS SEMANTICI TROVATE ————————");
for (int i = 0; i ¡ this.BlockGroups.ListaBlocchi.size(); i++) {
Block l = (Block) this.BlockGroups.ListaBlocchi.get(i);
System.out.println(" Blocco: " + l.BranchName + " ¡-¿ " + l.MergeName);
for (int j = 0; j ¡ l.blockPaths.size(); j++) {
List path = (List) l.blockPaths.get(j);
for (int k = 0; k ¡ path.size(); k++) {
System.out.print(((Nodo) path.get(k)).getName() + ",");
```

```
}
System.out.println("");
}
}
}
}
```

ManipolaFile.java

```
package esempi;

    import Algoritmi.Algoritmi;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.util.LinkedList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import Modello.Apackage.A;
import Modello.Apackage.AElement;
import Modello.Nodo;
import Modello.Spakage.S;
import Modello.Spakage.SElement;
import graphics.Grafo;
import javax.swing.JFrame;
import javax.swing.JPanel;

    public class ManipolaFile {

    public File fileLettura;
public File fileScrittura;
public List righeFile;
```

```
public List listaNodi;
public A listaA;
public A listaD;
public S matriceS;
private Algoritmi algoritmi;

    public ManipolaFile(String filediLettura, String filediScrittura){
fileLettura = new File(filediLettura);
fileScrittura = new File(filediScrittura);
this.righeFile = new LinkedList();
this.listaNodi = new LinkedList();
this.listaA = new A();
this.listaD = new A();
this.matriceS = new S();
algoritmi= new Algoritmi();
}

    public void LeggiFile (){
FileInputStream fis = null;
try System.out.println("————————— READ FILE START ————————
————————");
fis = new FileInputStream(fileLettura);
InputStreamReader isr = new InputStreamReader(fis);
BufferedReader br = new BufferedReader(isr);
String linea = br.readLine();
while (linea != null) {
righeFile.add(linea);
System.out.println(linea);
linea = br.readLine();
}
System.out.println("————————— READ FILE STOP ————————
————————");
System.out.println();
System.out.println();
} catch (IOException ex) {
Logger.getLogger(ManipolaFile.class.getName()).log(Level.SEVERE, null, ex);
} finally {
try {
fis.close();
```

```
} catch (IOException ex) {
Logger.getLogger(ManipolaFile.class.getName()).log(Level.SEVERE, null, ex);
}
}
}

    public void ScriviFile() {
FileOutputStream fos = null;
try {
String azione;
// Per usare l'Append allora fos = new FileOutputStream(fileScrittura,true);
fos = new FileOutputStream(fileScrittura);
PrintStream ps = new PrintStream(fos);
for (int i=0;i¡this.righeFile.size();i++){
azione=(String)this.righeFile.get(i);
ps.println(azione);}
} catch (FileNotFoundException ex) {
Logger.getLogger(ManipolaFile.class.getName()).log(Level.SEVERE, null, ex);
} finally {
try {
fos.close();
} catch (IOException ex) {
Logger.getLogger(ManipolaFile.class.getName()).log(Level.SEVERE, null, ex);
}
}

    }

    public void ParserFile (){
String appoggio;
String fulfil = "fulfil(";
String prec = "prec(";
String ev = "ev(";
String msg = "msg(";

    Nodo n ;
AElement a_elem;

    for (int i=0;i¡this.righeFile.size();i++){
```

```
appoggio = (String) righeFile.get(i);
if (!appoggio.contains("//")){
if (appoggio.contains(fulfil)){
n= new Nodo(appoggio.substring(appoggio.indexOf("_")+2, appoggio.indexOf(",")),
appoggio.substring(appoggio.indexOf(", _")+3, appoggio.indexOf(")")));
this.listaNodi.add(n);
}
else if (appoggio.contains(ev)){
n= new Nodo(appoggio.substring(appoggio.indexOf("_")+2, appoggio.indexOf(",")),
"Evento");
this.listaNodi.add(n);
}
/* else if (appoggio.contains(msg)){
n= new Nodo(appoggio.substring(appoggio.indexOf("_")+2, appoggio.indexOf(",")),
"Messaggio");
this.listaNodi.add(n);
}*/
else if (appoggio.contains(prec)) {
String primoNodo = appoggio.substring(appoggio.indexOf("_")+2, appoggio.indexOf(","));
String secondoNodo = appoggio.substring(appoggio.indexOf(", _")+3, appog-
gio.indexOf(")"));
listaA.incrementaArchiUscenti(primoNodo);
matriceS.setAdiacenzeXNodo(primoNodo, secondoNodo);
}
else if (appoggio.contains("adec(_")) // BRANCH = AND
String nodoInBranch = appoggio.substring(appoggio.indexOf("_")+2, appog-
gio.indexOf(","));
n = new Nodo(nodoInBranch,"Branch");
n.setBranch(1);
this.listaNodi.add(n);
}
else if (appoggio.contains("xdec(_")) // BRANCH = XOR
String nodoInBranch = appoggio.substring(appoggio.indexOf("_")+2, appog-
gio.indexOf(","));
n = new Nodo(nodoInBranch,"Branch");
n.setBranch(2);
this.listaNodi.add(n);

    }
```

```
else if (appoggio.contains("odec(_")) // BRANCH = OR
String nodoInBranch = appoggio.substring(appoggio.indexOf("(_")+2, appog-
gio.indexOf(","));
n = new Nodo(nodoInBranch,"Branch");
n.setBranch(3);
this.listaNodi.add(n);

    }
else if (appoggio.contains("syncmerge(_")) // MERGE = SYNC
String nodoInMerge = appoggio.substring(appoggio.indexOf("(_")+2, appog-
gio.indexOf(")"));
n = new Nodo(nodoInMerge,"Merge");
n.setMerge(1);
this.listaNodi.add(n);

    }
else if (appoggio.contains("omerge(_")) // MERGE = SIMPLE
String nodoInMerge = appoggio.substring(appoggio.indexOf("(_")+2, appog-
gio.indexOf(")"));

    n = new Nodo(nodoInMerge,"Merge");
n.setMerge(3);
this.listaNodi.add(n);
}
else if (appoggio.contains("xmerge(_")) // MERGE = SIMPLE
String nodoInMerge = appoggio.substring(appoggio.indexOf("(_")+2, appog-
gio.indexOf(")"));

    n = new Nodo(nodoInMerge,"Merge");
n.setMerge(2);
this.listaNodi.add(n);
}
}
}

    for (int i=0;i¡listaNodi.size();i++) {
boolean presente=false;
String stringaListaNodo = ((Nodo)listaNodi.get(i)).getName(); //gt name sul
nodo mi restituisce una stringa
```

```
for (int k=0;k¡listaA.getArchiUscenti().size();k++) {
String stringaListaA = ((AElement) listaA.getArchiUscenti().get(k)).getName();
if(stringaListaA.equals(stringaListaNodo))presente=true;}

    // Fare il controllo per inserire i nodi che non contengono successori in A
// listaNodi.get(i)).getName()
// listaA.getArchiUscenti().get(i)
}
if (!presente){
AElement newAe = new AElement(stringaListaNodo,0);
listaA.insertElement(newAe);
}
}

    for (int i=0;i¡listaA.getArchiUscenti().size();i++) {
a_elem = new AElement(((AElement)listaA.getArchiUscenti().get(i)).getName(),0);
listaD.insertElement(a_elem);
}
}

    public void ApplicaAlgoritmoVisitaEsaustivaSemplice(){
this.algoritmi.VisitaEsaustivaSemplice(listaNodi, listaA, listaD, matriceS);
}

    public static void main(String[] args) {
ManipolaFile manipolaFile = new ManipolaFile("Prova_grande.txt", "out.txt");
manipolaFile.LeggiFile();
manipolaFile.ParserFile();
manipolaFile.StampaStrutture();
//manipolaFile.ScriviFile();
manipolaFile.ApplicaAlgoritmoVisitaEsaustivaSemplice();

    }

    private void DisegnaGrafo() {
JFrame frame = new JFrame();
JPanel grafo = new Grafo(this);
frame.add(grafo);
```

```
    frame.setBounds(50, 50, 800, 600);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

    private void StampaStrutture() {
System.out.println("———————— GRAPH START ————————
————————");
for (int i=0;i¡this.listaNodi.size();i++)
{System.out.println("Nodo "+i+" :"+ ((Nodo)this.listaNodi.get(i)).getName()+
", act: "+ ((Nodo)this.listaNodi.get(i)).getAct()+", branch: "+((Nodo)this.listaNodi.get(i)).getBranch()
merge: "+ ((Nodo)this.listaNodi.get(i)).getMerge());

    }
System.out.println(" ");
AElement ae;
for (int i=0;i¡this.listaA.getArchiUscenti().size();i++){
ae = (AElement) this.listaA.getArchiUscenti().get(i);
System.out.println("AElement "+ i + " :"+ae.getName()+" numero archi:
"+ae.getNumeroArchi());
}
System.out.println(" ");

    SElement se;
for (int i=0;i¡this.matriceS.getS().size();i++)
{se = (SElement) this.matriceS.getS().get(i);
System.out.println(" SElement "+i+" :"+se.getName());
for (int k=0;k¡se.getAdiacenze().size();k++)
{System.out.println(" Adiacenza "+k+" :"+((String)se.getAdiacenze().get(k)));

    }

    }
System.out.println("———————— GRAPH STOP ————————
————————");
System.out.println();
System.out.println();

    // DisegnaGrafo();
```

```
}
    }
```

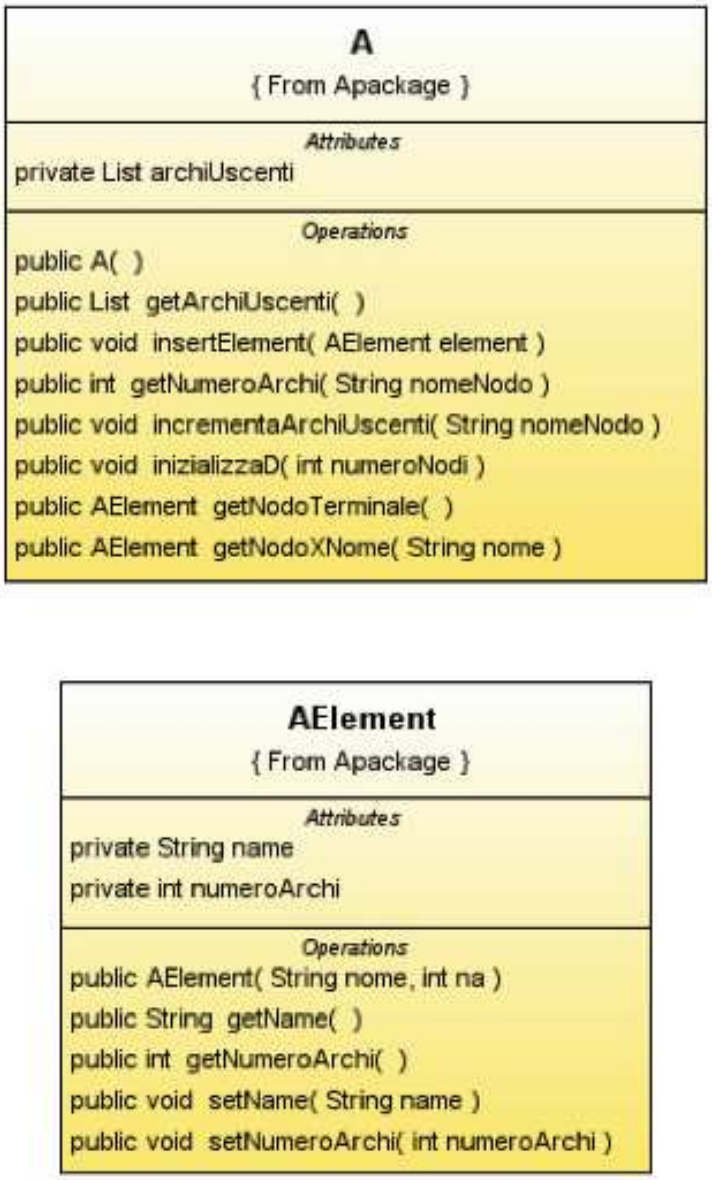


Figure A.2: Class Diagram A, Utility.

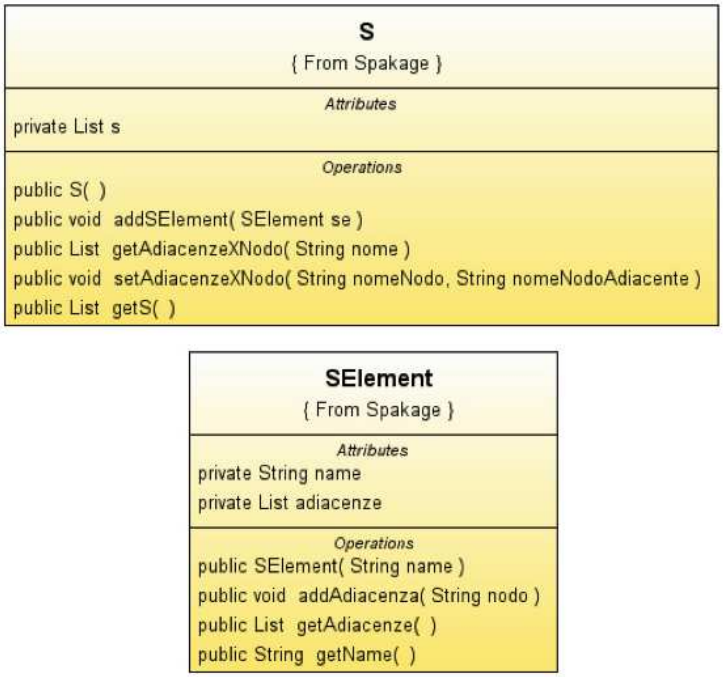


Figure A.3: Class Diagram S, Adjacent Matrix.

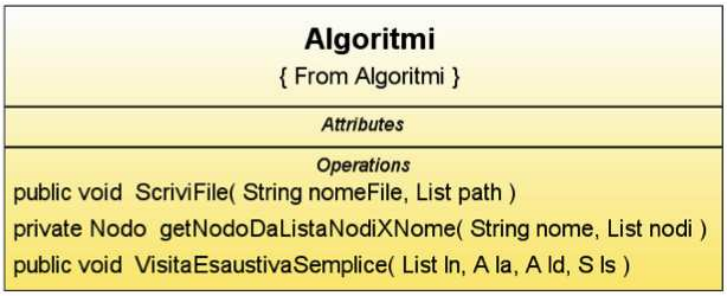


Figure A.4: Class Diagram Algoritmi.

Bibliography

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(832.843), 1983.

[Baj00] Rupnik R. Krisper M. Bajec, M. Using business rules technologies to bridge the gap between business and business applications. In Ed G Rechnu, editor, *Information Technology for Business Management*, pages 77–85, 2000.

[Boc05] Gruninger M. Bock, C. Psl: A semantic domain for flow models. In *Software and Systems Modeling Journal*, 2005.

[Boo00] Rumbaugh J. Jacobson Booch, G. *The Unified Modelling Language User Guide*. Booch, G., Rumbaugh, J., Jacobson,, 2000.

[Coaa] The OWL Services Coalition. Owl-s: Semantic markup for web services.

[Coab] Workflow Management Coalition. Wfmc.

[Con96] ACT-NET Consortium. The active database management systems manifesto: A rulebase of adbms features. In *ACM Sigmod Record*, pages 40–49, 1996.

[dA99] Van der Aalst. Formalization and verification of event-driven process chains. In *Information & Software Technology*, volume 41, pages 639–650, 1999.

[D.M91] Gabbay. D.M. *Temporal Logic: Mathematical Foundations and Computational Aspects.* Oxford University Press, 1991.

[Dor] P. Dorsey. Business rules analysis in the real world.

[DRF05] H. Lausen J. de Bruijn R. Lara M. Stollberg A. Polleres C. Feier C. Bussler D. Roman, U. Keller and D. Fensel. *Applied Ontology*, volume 1, chapter Web Service Modeling Ontology. Applied Ontology, 2005.

[FD07] F. Taglino. F. D'Antonio, M. Missikoff. Formalizing the opal ebusiness ontology design patterns with owl. *Third International Conference on Interoperability for Enterprise Applications and Software, I-ESA*, 2007.

[Fuc08] Kaljurand K. Kuhn T. Fuchs, N.E. Discourse representation structures for ace 6.0. ifi 2008.02, Department of Informatics, University of Zurich, 2008.

[GA04] F. van Harmelen. G. Antoniou. *Handbook on Ontologies.*, chapter Web ontology language: Owl., pages 67–92. Springer, 2004.

[GM99a] Schlenoff Craig Gruninger Michael. Process specification language (psl): results of the first pilot implementation. In *Proceedings of IMECE: International Mechanical Engineering Congress and Exposition*, pages 1–10, 1999.

[GM99b] Schlenoff Craig Gruninger Michael. *A robust process ontology for manunfacturing system integration.* NIST, 1999.

[Groa] Object Management Group. Business process modeling notation specification. version 1.0. february 2006.

[Grob] Object Management Group. Business rules in models: Request for information.

[groc] OMG UML group. Unified modeling language: Superstructure version 2.1.1.

[Gro00] Business Rules Group. *efining Business Rules What Are They Really?* Business Rules Group, 3rd edition, 2000.

[Gro06] OMG Group. *Semantics of Business Vocabulary and Business Rules Specification.* OMG, 2006.

[hom] Business Rules Solutions homepage.

[IBM] SAP AG IBM. Ws-bpel extension for people–bpel4people 2005.

[idea] idef. Idef function modeling method.

[ideb] idef. Idef process description capture method.

[IIIa] BEA Inc SAP AG IBM Inc, Microsoft Corp and Siebel Systems Inc. The business process execution language for web services. version 1.1, may 2003.

[IIIb] BEA Inc SAP AG IBM Inc, Microsoft Corp and Siebel Systems Inc. The business process execution language for web services, version 1.1, may 2003.

[Inf] Infrex. Infrex. product overview.

[Int] Intalio. Intalio business process management suite.

[JM05] A. Price J. Mendling, M. zur Muehlen. *Process-Aware Information Systems*, pages 281–316. WILEY-INTERSCIENCE, 2005.

[Mil93] Robin Milner. *The Polyadic -Calculus: A Tutorial. Logic and Algebra of Specification.* Milner, 1993.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus.* Cambridge Univ. Press, 1999.

[MRG92] R. E. Fikes M. R. Genesereth. Knowledge interchange format, version 3.0 reference manual. Technical Report 92-1, Computer Science Department, Stanford University, 1992.

[OAS] OASIS. Reference model for service oriented architecture 1.0 (committee specification).

[Pet77] James L. Peterson. *Petri Nets*, volume 3, pages 223–252. ACM Computing Surveys, 1977.

[Pet81] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems.* ISBN 0-13-661983-5. Prentice Hall, 1981.

[PF99] Reiter R. Pirri F. Some contributions to the metatheory of the situation calculus. In ACM, editor, *Journal of the ACM*, volume 46, pages 325–361. ACM, 1999.

[Pic01] *The Pi-calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

[PR04] K. Verma A. Sheth P. Rajasekaran, J. Miller. Enhancing web services description and discovery to facilitate composition. In *International Workshop on Semantic Web Services and Web Process Composition*, 2004.

[RM86] Kowalski R. and Sergot M. A logic-based calculus of events. In *New Generation Computing.*, number 4(1), pages 67–96. 1986.

[Ros97] R.G. Ross. *The Business Rule Book: Classifying, Defining and Modelling Rules.* Database Research Group, 2nd edition, 1997.

[Ros02] Greenspan S. Wild C.: Rosca, D. Enterprise modelling and decision-support for automating the business rules lifecycle. In *Automated Software Engineering*, volume 9, 2002.

[SC00] et al Schlenoff Craig, Gruninger Michael. *The Process Specification Language (PSL) Overview and Version 1.0 Specification.* NIST, 2000.

[Sol00] et al. Soley, R. *Model Driven Architecture.* OMG, 2000.

[Sow92] Zachman J.A. Sowa, F. Extending and formalising the framework for information systems architecture. In *BM Systems Journal.* IBM, 1992.

[Tec] Yasu Technologies. Quickrules discovery guide.

[Val04] Vasilecas O. Valatkaite, I. On business rules approach to the information systems development. In *Proc. of Twelfth International Conference on Information Systems Development. Constructing the Infrastructure for the Knowledge Economy.*, 2004.

[VdA06] Pesic M Van der Aalst, W.M.P. Decserflow: Towards a truly declarative service flow language. In *DecSerFlow.* WS-FM, 2006.

[VH94] B. Von Halle. *Back to Business Rule Basics, Database Programming and Design*, pages 15–18. Business Rules Group, 1994.

[W3C] W3C. Web service semantics - wsdl-s 2005.

[WFM] WFMC. Process definition interface – xml process definition language, version 2.00, october 2005.

[Whi04] S.A. White. *Introduction to BPMN*. IBM, 2004.

[WK04] Loucopoulos P. Wan Kadir, W.M.N. Relating evolving business rules to software design. In *Journal of Systems Architecture*, pages 367–382, 2004.