



An Ontological Approach to Business Process Modeling

Antonio de Nicola, Mario Lezoche, Michele Missikoff

► To cite this version:

Antonio de Nicola, Mario Lezoche, Michele Missikoff. An Ontological Approach to Business Process Modeling. 3th Indian International Conference on Artificial Intelligence 2007, Dec 2007, India. pp.ISBN 978-0-9727412-2-4. hal-00656686

HAL Id: hal-00656686

<https://hal.science/hal-00656686>

Submitted on 5 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An Ontological Approach to Business Process Modeling

Antonio De Nicola, Mario Lezoche, Michele Missikoff

LEKS, IASI-CNR
Viale Manzoni 30, 00185 Rome Italy
{denicola, lezoche, missikoff}@iasi.cnr.it

Abstract. Business people use informal methods to represent business processes (BP), having the main objective to support an enterprise organization. On the other hand, application software is increasingly based on Service Oriented Architectures, where the application logic is represented by executable BP (e.g., by using BPEL.) Despite both are aiming at BP modelling, the methods used by business people and IT specialists are quite different. The former use informal, descriptive methods, with an intuitive semantics difficult to be translated to the formal representation needed in the IT world. This paper presents the main lines of an ontological framework for the representation of BP semantics: BPAL (Business Process Abstract Language.) It is primarily conceived to provide a formal semantics to BPMN, an informal BP modelling method that is emerging in the business world. The modelling categories of BPAL are based on well accepted business notions, such as *activity*, *decision*, *role*. We believe that it may be useful beyond BPMN, in more general business contexts. BPAL is an abstract language (no drawing symbols are provided) having a procedural semantics (allowing a translation to an executable form, BPEL), and a declarative semantics, to be processed by an inference engine.

1. Introduction

This paper illustrates a method aimed at an effective involvement of business people in the development of software applications. This is achieved by providing a framework that allows the semantic enrichment of process models produced by business experts. Semantic enrichment is obtained by the semantic annotation of the business models based on a core business process ontology (CBO).

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 [27], the recent advent of Service Oriented Architectures [28] 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 [9] [10]), 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 implementation 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 [29].) However, current semantic technologies [30] 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 [17], [16], [15].

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 [3], IDEF [5] [6] UML-Activity Diagram [1], and BPMN [7] [8]. 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 [9] and XPD [8] [11].

- *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 [13] [20], Pi-Calculus [21], Petri-Nets [24].

Finally, there are the ontology-based process languages, such as OWL-S [17], WSDL-S [31], WISMO [15], and Meteor-S [16]. 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 paper 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: intuitivity and ease of use, rigorous mathematical basis, and possibility of execution. 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 [12];
- axiomatization to allow inference mechanisms to be applied;
- possibility of execution, by mapping BPAL to the popular BPEL.

The rest of the paper is organised as follows. Section 2 presents the related work, organised in the three mentioned groups. Section 3 introduces the main BPMN constructs while Section 4 is the central part that introduces BPAL. Section 5 provides a mapping between BPMN, PSL and BPAL, followed by Section 6 with the conclusions.

2. Related Work

In this section we briefly present the major BP modelling methods and languages, organised according to the three groups identified in the previous section.

2.1 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.

4 Antonio De Nicola, Mario Lezoche, Michele Missikoff

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 1990's. 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 [4].

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

¹ www.uml.org

² Object Management Group: www.omg.org

³ Meta-Object Facility

and monitor those processes. Since we selected BPMN as the candidate BP modelling method, we will specifically elaborate on it in the next Section 3.

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 [14], 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.

2.3 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, PSL's 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 [18]. 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 [24]. 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 [25].

In theoretical computer science, the Pi-Calculus (π -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 π -calculus is to be able to describe concurrent computations whose configuration may change during the computation.

The π -calculus belongs to the family of process calculi, mathematical formalisms for describing and analyzing properties of concurrent computation [22]. In fact, the π -calculus, like the λ -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 π -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 π -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 [23].

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.

2.4 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 [18]. 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 [20]. 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 [12] (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 [19]. 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 PSL's purposes, a suitable foundation is a modified and extended variation of the *situation calculus*. The reason for this is that the situation calculus's 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 [20].

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.

3. BPMN: an emerging modeling method for business people

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.

3.1 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 (see Table 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 events*⁴.

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).

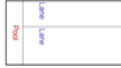





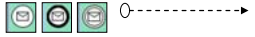
Modeling categories	BPMN elements	Graphical Notation
Actor	Pool, Lane	
Behavior	Process, Task, Sub-process	
Decision	Gateway (Data/Event based, AND/OR/XOR)	
Event	Event (Start, Intermediate, End)	
Transition	Sequence Flow	
Object	DataObject	
Coaction	Message Event, MessageFlow	

Table 1 Categorization of BPMN constructs and corresponding graphical notation

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.

⁴ Despite *event* is basic modeling construct in BPMN, its semantics is not clear. In fact, when mapped into BPEL, they are considered as atomic activities (e.g., receive, wait, and throw) [7].

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*.

3.2 An example: Existence Verification of a Company

To better illustrate the BP modeling by using BPMN, below we introduce a simple example (Figure 1), drawn from a case study addressed in the LD-CAST European Project⁵: the search for a partner in a cross-border off-shoring. In particular, we address the activities related to the *Supply Sub-Contracting*, according to a specific EU directive.

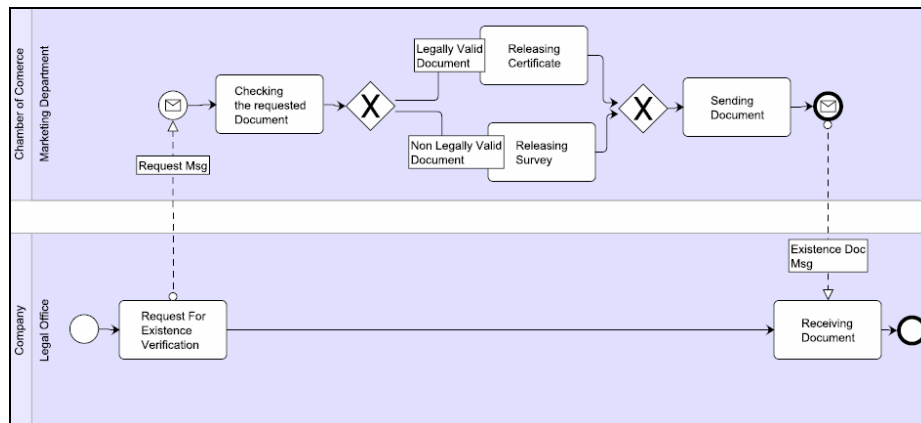


Figure 1 An excerpt of the existence verification process

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 the tenders, the Polish company needs to perform a number of verifications. In particular, for each proponent company, the following verifications are performed: (i) existence verification, (ii) fiscal verification, and (iii) technical requirements and quality standards verification. Such verifications are performed through the local Chambers of Commerce. The Figure 1 represents the BPMN process concerning point (i), the request for verifying

⁵ Local Development Cooperation Actions Enabled By Semantic Technology (LD-CAST) Project: <http://www.ldcastproject.com>

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.

4. BPAL: an ontological framework⁶ for BP

In this section we present BPAL, a framework for the management of BP ontologies, in the context of BPMN business process modeling method.

BPAL is structured according to a number of modeling notions defined in accordance with the business culture, corresponding also to the constructs of BPMN. 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

⁶ The term “framework” is overloaded. Here we mean a language (lexicon and grammar), a set of axioms, an inference mechanism, and a collection of methods, tools, and best practices aimed at producing a valid BP model. Here we illustrate the BPAL framework at a descriptive level, since a formal treatment falls outside of the scope of this paper.

⁷ For sake of conciseness, here we will refer to a simplified version of BPMN, sufficient to present the BPAL methodology.

⁸ Please note that this is a second order instantiation, very different from the more usual round instantiation. In fact, BPAL contains meta-concepts used to create a BP ontology.

diagram. Seen it originates from BPMN, the latter is used whenever we need a concrete (displayed) diagram.

4.1 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 CBO⁹.

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*).¹⁰

⁹ Here, for sake of simplicity, we refer to the CBO as a sort of catalogue of concepts. Indeed, it is implemented by a reference ontology built according to the OPAL framework [29]

¹⁰ Please note that the message flow is not constrained by the control flow, unlike BPMN.

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.)

4.2 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 2 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.

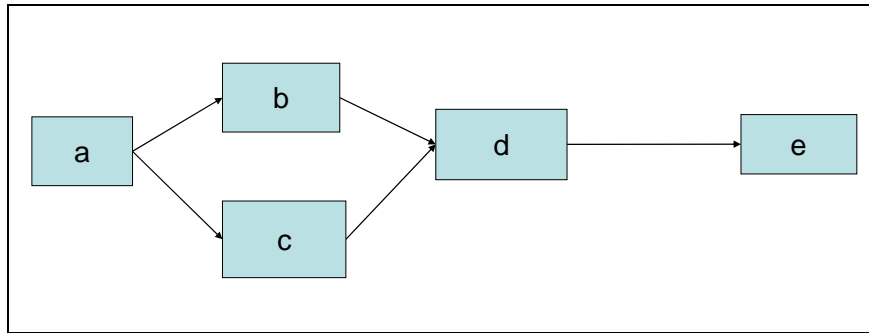


Figure 2 A simple BPMN Diagram

Here we have the corresponding BPAL abstract diagram.

```

act(a), act(b), act(c), act(d), act(e);
prec(a,b), prec(a,c), prec(c,d), prec(b,d), prec(d,e).
  
```

A BPAL Abstract Diagram

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

4.3 The 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 treatment 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.

Axiom1:

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

According to Axiom1, the diagram of Figure 1 is invalid and needs to be transformed in the diagram of Figure 3.

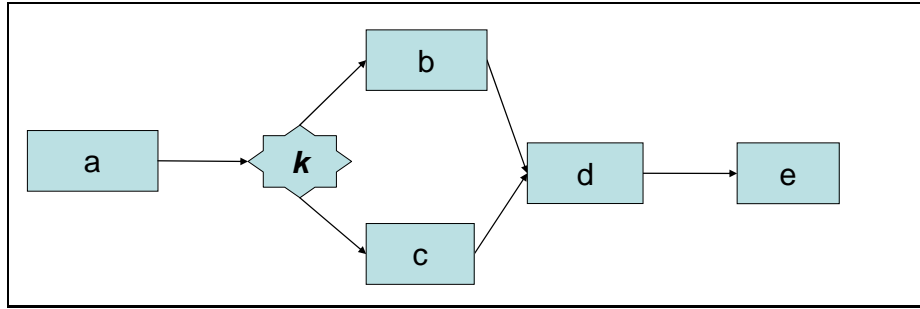


Figure 3 A Generic BPAL Process

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:

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 1 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.

4.4 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. A full elaboration of this part falls outside the scope of this paper.

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

The initial objective of this paper 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 Section 2. 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 table 2, 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 too 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 (under construction)

Duration and Ordering Activity

- Iterated Activity

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

Table 2 Comparison between the BPMN, PSL, and BPAL

From the table 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 (our intuition is that, 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.

6. Conclusions and future work

In this paper we presented the main ideas of BPAL, an ontological framework for business process modeling. There are a very large number of BP modeling languages, so the decision to build a new proposal in this area started from a number of motivations.

- the advent of a new BP modeling method, BPMN, proposed by OMG;
- the fact that BPMN does not have a formal grounding, being it specified in an informal way;
- BPAL has been initially conceived to develop a formal account of BPMN processes, carefully taking in consideration the business experts needs;
- The BPAL approach also considers the BP design process, therefore including primitives that support a stepwise refinement of processes being designed;
- BPAL remains at an abstract level, since it relies on BPMN for its concrete diagrammatic representation and on BPEL for its actual execution;
- BPAL framework includes an axiom system that allow an automatic validation check to be performed. Since its formal semantics is based on KIF, the inference component is based on a KIF reasoner.

After a careful analysis of the literature we identified PSL as a possible candidate to match the above requirements, but finally we decided that BPAL was more suited to our needs.

The current BPAL version is a preliminary one. We are consolidating it by proceeding along two lines. One is methodological, necessary to strengthen its formal grounding, and another is empirical, necessary to check its value in real business settings. About the latter, we are using BPAL in several pilot applications, the most relevant being the LD-CAST European project, for the Chambers of Commerce, and TOCAL, a national project aiming at an advanced platform for autonomic logistics services.

References

- [1] Unified Modeling Language: Superstructure version 2.1.1. (<http://www.omg.org/docs/formal/07-02-03.pdf>)
- [2] G. Engels, A. Förster, R. Heckel, S. Thöne. *Process Modeling Using UML*. In “Process-Aware Information Systems”. Edited by M. Dumas, W. van der Aalst, A.H.M ter Hofstede. WILEY-INTERSCIENCE, Pages 85-117, (2005).
- [3] A.-W. Scheer, O. Thomas, O. Adam. *Process Modeling Using Event-Driven Process Chains*. In “Process-Aware Information Systems”. Edited by M. Dumas, W.

- van der Aalst, A.H.M ter Hofstede. WILEY-INTERSCIENCE, Pages 119-145, (2005).
- [4] B. F. van Dongen, W. M. P. van der Aalst, H. M. W. Verbeek. *Verification of EPCs: Using Reduction Rules and Petri Nets*. CAiSE 2005: 372-386
- [5] IDEF Function Modeling Method. [<http://www.idef.com/IDEF0.html>]
- [6] IDEF Process Description Capture Method. [<http://www.idef.com/IDEF3.html>]
- [7] Object Management Group. *Business Process Modeling Notation Specification. Version 1.0*. February 2006 [www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%2010%20Spec%2006-02-01.pdf]
- [8] J. Mendling, M. zur Muehlen, A. Price. *Standards for Workflow Definition and Execution*. In "Process-Aware Information Systems". Edited by M. Dumas, W. van der Aalst, A.H.M ter Hofstede. WILEY-INTERSCIENCE, Pages 281-316, (2005).
- [9] R. Khalaf, N. Mukhi, F. Curbera, and S. Weerawarana. *The Business Process Execution Language for Web Services*. In "Process-Aware Information Systems". Edited by M. Dumas, W. van der Aalst, A.H.M ter Hofstede. WILEY-INTERSCIENCE, Pages 317-342, (2005).
- [10] IBM Inc., Microsoft Corp., BEA Inc., SAP AG, and Siebel Systems Inc. *The Business Process Execution Language for Web Services*, version 1.1, May 2003. [<http://www-128.ibm.com/developerworks/library/specification/ws-bpel>]
- [11] WfMC. Process Definition Interface – XML Process Definition Language, version 2.00, October 2005. [http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf]
- [12] M. R. Genesereth, R. E. Fikes (Editors). *Knowledge Interchange Format, Version 3.0 Reference Manual*. Computer Science Department, Stanford University, Technical Report Logic-92-1, June 1992.
- [13] Bock, C., Gruninger, M., "PSL: A Semantic Domain for Flow Models," *Software and Systems Modeling Journal*, 2005.
- [14] IBM, SAP AG. *WS-BPEL Extension for People-BPEL4People*. Whitepaper. [<http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/>], 2005.
- [15]–D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel: *Web Service Modeling Ontology, Applied Ontology*, 1(1): 77 - 106, 2005.
- [16] P. Rajasekaran, J. Miller, K. Verma, A. Sheth, *Enhancing Web Services Description and Discovery to Facilitate Composition*, International Workshop on Semantic Web Services and Web Process Composition, 2004 (Proceedings of SWSWPC 2004).
- [17] The OWL Services Coalition. *OWL-S: Semantic Markup for Web Services*. <http://www.daml.org/services/owl-s/1.0/owl-s.pdf>.
- [18] Gruninger Michael, Schlenoff Craig, "Process Specification Language (PSL): results of the first pilot implementation", Proceedings of IMECE: International Mechanical Engineering Congress and Exposition, pp 1-10, 1999.
- [19] Schlenoff Craig, Ivester Rob, Knutilla Amy: "A robust process ontology for manufacturing system integration", NIST, 1998.
- [20] Schlenoff Craig, Gruninger Michael, et al "The Process Specification Language (PSL) Overview and Version 1.0 Specification", NIST, 2000

- [21] Robin Milner: Communicating and Mobile Systems: the Pi-Calculus, Cambridge Univ. Press, 1999, ISBN 0-521-65869-1
- [22] Robin Milner: The Polyadic π -Calculus: A Tutorial. Logic and Algebra of Specification, 1993.
- [23] Davide Sangiorgi and David Walker: The Pi-calculus: A Theory of Mobile Processes, Cambridge University Press, ISBN 0-521-78177-9
- [24] Peterson, James L. (1977). "Petri Nets". ACM Computing Surveys 9 (3): 223–252.
- [25] Peterson, James Lyle. Petri Net Theory and the Modeling of Systems. Prentice Hall. ISBN 0-13-661983-5.
- [26] Petri, Carl A. (1962). "Kommunikation mit Automaten". Ph. D. Thesis. University of Bonn.
- [27] Workflow Management Coalition (WfMC). [<http://www.wfmc.org/>]
- [28] OASIS. *Reference Model for Service Oriented Architecture 1.0 (Committee Specification)* www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf. (2006).
- [29] F. D'Antonio, M. Missikoff, F. Taglino. *Formalizing the OPAL eBusiness ontology design patterns with OWL*. Third International Conference on Interoperability for Enterprise Applications and Software, I-EISA 2007.
- [30] G. Antoniou, F. van Harmelen. *Web ontology language: Owl*. In Handbook on Ontologies. (S. Staab and R. Studer, ed.), Springer, pages 67-92, 2004.
- [31] W3C. *Web Service Semantics - WSDL-S*. [<http://www.w3.org/Submission/WSDL-S>, 2005.]

Appendix 1: 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)

Appendix 2. 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.

(ext ?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) (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)
             (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))).
```