Semantics based aspect oriented management of exceptional flows in business processes

Chiara Ghidini, Chiara Di Francescomarino, Marco Rospocher, Paolo Tonella, Member, IEEE, Luciano Serafini

Abstract—Enriching business process models with semantic annotations taken from an ontology has become a crucial need both in service provisioning, integration and composition, and in business processes management. We represent semantically annotated business processes as part of an OWL knowledge base that formalises the business process structure, the business domain, a set of criteria describing correct semantic annotations, and a set of constraints describing requirements on the business process itself.

In this paper we show how semantic web representation and reasoning techniques can be: (i) exploited by our aspect-oriented approach to modularize exception handling (as well as other crosscutting) mechanisms; (ii) effectively applied to formalise and automatically verify constraints on the management of exceptional flows (as well as other relevant flows) in business processes. The benefits of the semantic web and the aspect-oriented technologies are illustrated on a case study, where exceptional flows are modularized separately and independently at the semantic level thanks to the proposed approach.

Index Terms—Business process modelling, semantic annotation, ontology, exception handling.

I. INTRODUCTION

Semantic Business Process Management [1], [2] aims at improving the level of automation in the specification, implementation, execution, and monitoring of business processes by extending business process management tools with the most significant results from the area of Semantic Web. When the focus is on process modelling, i.e. the activity of specification of business processes at an abstract level (descriptive and non executable), annotating process descriptions with labels taken from a set of domain ontologies provides additional support to the business analysis (see e.g. [3]).

A crucial step in process modelling is the creation of valid and robust diagrams that not only comply with the basic requirements of the process semantics but also satisfy properties related to the domain specific semantics and are able to care about exceptional behaviours as well as verify their correct management. For instance, a possible requirement devoted to ensure the robustness of an on-line shopping process could be the fact that the activity of reserving products must be able to handle errors due to product unavailability, by means of proper exceptional flows in the business process. Caring about exceptional behaviours and verifying their correct management, in fact, is one of the key factors contributing to the process robustness [4]. On the other hand, enriching the so called “happy path” of process models with exceptional flows, hence increasing the complexity of the models, not only requires an understanding of the properties related to the domain specific semantics, and is often uncomfortable for the designers when modelling the processes, but it has also drawbacks for the process readability, thus harming the process model capability of acting as a means of communication [5].

The AOP (Aspect Oriented Programming) literature [6] has widely investigated this problem for general purpose languages by proposing a solution based on the separation of exception handling concerns by means of their modularization into aspects [7]. An aspect is a module that encapsulates a secondary behaviour of a main view (e.g., an exceptional flow on the “happy path”). Taking advantage of the separation of concerns, designers can deal with aspects (specifying for example exceptional behaviours) separately and independently from the main view (e.g., the “happy” path). If needed, aspects can be added to the principal perspective in a weaving phase, thus generating the “woven” (integrated) process. The weaving is performed by exploiting the information about the concern location specified in the aspect itself by using both the process and the domain specific semantic information.

The main purpose of this paper is to support designers in the modelling phase by proposing the “aspectisation” of requirements crosscutting a BPMN (Business Process Modelling Notation [8]) annotated process, with particular emphasis on exception handling. In particular, we show how techniques taken from the Semantic Web and the AOP can be used to:

(i) encode requirements on exception handling in formal constraints, expressed using the OWL-DL language;
(ii) verify whether a given BPMN diagram satisfies the requirements;
(iii) exploit the requirements to semi-automatically build the aspect(s) used to modularize the exception handling mechanisms;
(iv) verify whether the woven version of the business process enriched with aspects meets the initial requirements.

Previous works on the usage of formal semantics to support Business Process Management can be roughly divided into two groups: (i) those adding semantics to specify the dynamic behaviour exhibited by a business process (e.g., [9], [10], [11]), and (ii) those adding semantics to specify the meaning of the entities of a business process in order to improve the automation of business process management (e.g., [12], [13], [3], [14]). The approach we take in this paper, which extends the work introduced in [15] and [16], belongs to
the second group and, both for the “aspectisation” and the encoding and verification of constraints, we focus on the usage of the semantic web technology on structural requirements, that is, requirements which refer to descriptive properties of the annotated process diagram and not to its execution. The reason for such a choice is threefold. First, structural requirements complement behavioural properties, as they can be used to express properties of the process which cannot be detected by observing the process execution. Second, structural requirements provide an important class of expressions whose satisfiability can be directly verified with existing Description Logic (DL) reasoners. Third, the language required for describing aspects related to structural requirements is very close to the one used by business designers for process modelling.

The paper is structured as follows: in Section III we recall how to represent semantically annotated BPMN processes within an OWL-DL Business Process Knowledge Base (BPKB); in Section IV we illustrate how to encode typical classes of structural requirements with particular emphasis on a new type of structural constraints: the exception handling constraints; in Sections V and VI we describe how to modularize exception handling mechanisms into aspects by introducing an aspect-oriented visual language for annotated processes; and finally, in Section VII, we analyse the performances of the tools implementing the proposed approach in order to evaluate its applicability for on-line modelling purposes. An example, introduced in Section II, is used throughout the paper to illustrate the proposed approach, while a comparison with related works is contained in Section VIII.

II. A MOTIVATING EXAMPLE

In this section, we describe a portion of an on-line shopping process which we use throughout the paper as a motivating and explanatory example. The annotated process we refer to is illustrated in Figure 1 in the Business Process Modelling Notation (BPMN) [8], the (graphical) language used to draw Business Process Diagrams (BPD). In our semantic variant of BPMN we allow for the annotation of objects of a BPD with concept annotations taken from domain ontologies, i.e. shared formalizations of a specific domain. Semantic annotations are preceded in a BPD by the “@” symbol. Due to space limitations we consider here only the initial steps of the on-line shopping process (e.g. the product presentation and selection and the customer authentication), leaving out the last phases, e.g. the checkout. The process is structured in two sides: the server side, represented by the On-line Shop pool, which describes the on-line buying process from the point of view of the shop, and the client side, represented by the Customer pool, describing the process from the point of view of the buyers.

The modelling of the on-line shopping process depicted in Figure 1 involves business designers and analysts who may wish to impose and verify requirements on the process itself. In [16] we have introduced and studied classes of constraints used to specify requirements which cover different aspects of the process, ranging from the correct annotation of business processes to security issues, as in the following examples:

(a) requirements related to the semantic annotation: “to manage” is a complex action and can be used only to annotate BPMN sub-processes (and not atomic activities);
(b) requirements related to privacy issues: the activity of providing personal data is always immediately preceded by an activity of reading the policy of the organization;
(c) requirements related to security issues: the activity of user authentication is a sub-process which contains an activity of checking the customer data;
(d) requirements related to general issues: in the on-line shopping process there must be a Customer pool and an On-line Shop pool.

Beyond general requirements, designers and analysts are also interested in guaranteeing that processes correctly capture and manage exceptions. In this paper, we extend the range of constraints for which it would be desirable to have support, so as to include also requirements related to the management of exceptions and exception handling mechanisms. Examples range from simple requirements specifying which activities should be associated with an exception handling mechanism to more complex requirements specifying how exceptions should be handled:

(e) Existence of product unavailability exception: the activity of reserving products in the On-line Shop pool has always to catch a “product unavailability” error event;
(f) Handling of product unavailability: the “product unavailability” error event caught by the activity of reserving products in the On-line Shop pool has to be handled by executing in parallel two activities. The first one is an activity for warning the buyer; the second one is a sub-process for ordering the unavailable products;
(g) Handling of compulsory-login failure: the activity of “sending customer data” in the “log-in” process has always to allow receiving a “compulsory-login failure” error event from the On-line Shop pool. The “log-in” process has, in turn, always to catch this error and the error event has to be handled by stopping the process;

In the rest of the paper we show how semantic annotations can be used to support the formal specification and verification of exception handling constraints and the production of aspects used to modularize the exception handling mechanism. Designers and analysts are hence supported in the heavy task of exception and exception handling management and encouraged to enlarge their focus from the “happy path” only to include also the exceptional behaviours, specified separately to avoid cluttering the main view.

III. REPRESENTING SEMANTICALLY ANNOTATED PROCESSES

In order to represent semantically annotated BPDs, and to support automated reasoning on the requirements that can be expressed on them, we recall the notion of Business Process
A Business Process Knowledge Base (BPKB) described in [16], and schematised in Figure 2.

A BPKB is composed of four modules: a BPMN ontology, a domain ontology, a set of constraints and the BPD instances. We have implemented BPKB using the standard semantic web language OWL-DL based on Description Logics (DL) [17]. The terminological part (Tbox), which is the stable description of the domain, is provided by the upper level modules of Figure 2. Instead, the changeable part, which corresponds to a specific process description, is provided in the form of assertional knowledge (Abox).

A. The BPMN Ontology

The BPMN ontology, hereafter called BPMN0, formalizes the structure of a BPD. It is a formalization of the BPMN standard as described in Annex B of [8], and consists of a set of axioms that describe all the BPMN elements (e.g., tasks, gateways, pools) and the way in which they can be combined for the construction of BPDs. The ontology has currently the expressiveness of $ALCHQ1N(D)$ and a detailed description is contained in [18]. We remark that BPMN0 provides a formalization of the structural part of BPDs, i.e. which are the basic elements of a BPD and how they are (can be) connected. BPMN0 is not intended to model the dynamic behaviour of BPDs, i.e. how the flow proceeds within a
process. Ontology languages are not particularly suited to specify behavioural semantics, which can be better modelled using formal languages for Workflow or Business Process Specification based on Petri Nets, as proposed in [11].

B. The Domain Ontology

The domain ontology component, hereafter called BDO, consists of a (set of) OWL ontology(es) that describes a specific business domain. The concepts defined in the BDO are the terms that can be used to annotate a business process, thus allowing to give a precise semantics to the annotations. The BDO can be an already existing business domain ontology (e.g. RosettaNet or similar standard business ontologies), a customization of an existing ontology, or an artefact developed on purpose. Top level ontologies such as DOLCE [19] can also be included as components of the domain ontology and used to provide typical annotation patterns to the BPD objects. The BDO plays a key role in the formalization of requirements involving the domain semantics, as well as in the quantification mechanism used in aspect definition.

C. The Constraints

Constraints are used to ensure that important process requirements are satisfied. We distinguish between two different kinds of constraints: merging axioms, and process structural constraints (including exception handling constraints). Merging axioms state the correspondence between the domain ontology and the BPMN ontology, formalizing the criteria for correct/incorrect semantic annotations. Process structural constraints are expressions used to state structural requirements that apply to the process under construction. A detailed description of the constraints is provided in Section IV.

D. The BPD Instances

The BPD instances component of the BPKB consists in a set of ontology individuals and assertions which represent the elements of an annotated BPD in terms of instances of BPMNO and BDO classes. Each graphical object \( g \) of an annotated BPD \( \beta \) corresponds to an ontology individual in BPD instances. The assertions on these individuals can be divided into three groups, the first two of them involving concepts from BPMNO only, while the third one involves concepts from BDO only. BPMN-type assertions are used to store information on the BPMN type of graphical object \( g \). Thus we represent the fact that \( g \) is an exclusive gateway with the BPMN-type assertion data_based_exclusive_gateway(\( g \)). BPMN-structural assertions are used to store information on how the graphical objects are connected. Thus for every connecting object \( c \) of \( \beta \) that goes from \( a \) to \( b \), we generate two structural assertions of the form SourceRef(\( c,a \)) and TargetRef(\( c,b \)). For instance, the assertion has_sequence_flow_source_ref(\( c,a \)) states that the sequence flow \( c \) originates from gateway \( a \). Finally, BPMN-semantic assertions are used to represent annotation of graphical elements. For instance the assertion to_log_in(\( t \)) states that \( t \) is an instance of concept to_log_in and is obtained from the semantic annotation to_log_in of the BPD in Figure 1.

In [15], [16] we have described a tool (population tool) for the automated transformation of an annotated BPD into an OWL Abox. Given BPMNO and BDO, and a BPD annotated with concepts from BDO, the tool creates the Abox and populates the ontology with instances of BPMN elements belonging to the specific process. The current version of the tool works on BPMN processes specified using the Eclipse SOA Tools Platform and the Intalio Process Modeler tools, but can be adapted to work with different business process editors.

IV. Specifying and Verifying Process Requirements

To ensure that relevant process requirements are satisfied, we make use of constraints. In our previous work we have introduced two kinds of constraints: merging axioms, and structural constraints. Merging axioms, whose formalization is denoted by MA(BPMNO,BDO), are not directly relevant to the purpose of this paper, and their description is therefore omitted. Structural constraints are expressions used to state specific properties that relate to the structure of the process under construction. These expressions can have many different forms to match a variety of different properties of the process.

In this section we briefly describe the two main forms introduced in [16], that is, (i) containment constraints (including existence constraints), and (ii) precedence constraints; then, we introduce exception handling constraints; finally, we describe how to use key reasoning services to verify the constraints.

A. Containment constraints

Containment constraints are expressions of the form \( X \ contains \ Y \) or \( X \ is \ contained \ in \ Y \) and are used to represent the fact that the BPD or certain graphical elements contain other graphical elements. As such they can be used to express also informal statements of the form \( exists \ X \) and \( non \ exists \ X \), which are rephrased in the containment constraint \( diagram \ X \ contains \ Y \) and \( diagram \ X \ does \ not \ contain \ Y \). A simple containment constraint of the form \( X \ contains \ Y \) which can be expressed over the on-line shopping process is provided by requirement (c), while a constraint of the form \( exists \ X \) is given by requirement (d).

Containment constraints can be encoded in DL using specific BPMNO roles which formalise the containment relations existing between different BPD objects as described by specific attributes in [8]. An example of these specific roles is has_embedded_sub_process_sub_graphical_elements (hereafter abbreviated to has_elements), which corresponds to the “GraphicalElement” attribute of an embedded sub-process, and represents all the objects contained within an embedded sub-process. This role can be used to formalise Requirement (c) as follows:

\[
\begin{align*}
\text{BDO:to\_authenticate} & \sqsubseteq \text{BPMN:embedded\_sub\_process} \\
\text{BDO:to\_authenticate} & \sqsubseteq \text{BPMN:has\_elements} \\
\text{(BPMN:activity \sqcap \text{BDO:to\_check\_customer\_data})} & \text{ (2)}
\end{align*}
\]

\(1\)With the notation ontology:concept we indicate that B is a concept defined in ontology A.
B. Precedence constraints

Precedence constraints are used to represent the fact that certain graphical objects appear before others in the BPD. They can be of several forms. Significant examples are: X is always preceded by Y in all possible paths made of sequence flows and X is once preceded by Y in at least a path composition of sequence flows. Particular cases of these constraints are X is always immediately preceded by Y and X is once immediately preceded by Y. These constraints also require that X is a graphical object immediately preceded by Y by means of a sequence flow. Finally the precedence constraint X is activated by Y requires that X is activated by Y by means of a message flow. An example of X is always immediately preceded by Y constraint is provided by requirement (b).

Precedence constraints can be encoded in DL using specific BPMNO roles which formalize the connection between graphical objects. In particular the key roles we can use are:

- has_sequence_flow_source_ref;
- has_sequence_flow_target_ref.

These roles represent the “SourceRef” and “TargetRef” attributes of BPMN and identify which graphical elements the sequence flow is connected from and to respectively. Similar roles are also defined for message flows.

These roles can be used to formalise requirement (b) as follows:

\[
\begin{align*}
\text{BDO:to\_provide\_personal\_data} & \sqsubseteq \text{BPMN:activity} \\
\forall \text{BPMN:has\_sequence\_flow\_target_ref}. \\
\forall \text{BPMN:has\_sequence\_flow\_source_ref}. \\
\text{BDO:to\_read\_policy}
\end{align*}
\]

A detailed description of how to formalise more complex containment and precedence requirements, as well as a discussion of the expressivity limitations imposed by the usage of a decidable version of OWL are given in [16].

C. Exception Handling Constraints

These constraints are expressions used to represent the way specific exceptions should be handled. They describe structural properties of BPMN diagrams and are therefore a kind of structural constraints. Similarly to the other types of structural constraints they can have many different forms, and can specify simple requirements stating the need for an exception handling mechanism at a certain point in the process, as in the requirement (e), or more complex issues, as in the requirement (g).

The formalisation of exception handling constraints is based on the representation of BPMN Intermediate Events (None, Message, Timer, Error, Cancel, Compensation, Conditional, Link, Signal, and Multiple) in BPMNO. Intermediate events belong to the extended set of BPMN graphical elements described in [8] and are the mechanism BPMN suggests to use to represent exception or compensation handling. Intermediate events are part of the graphical objects of BPMNO, as depicted in Figure 3. In accordance with the properties of intermediate events, specified in [8] and encoded in BPMNO, these elements are further classified, via reasoning, in different groups: the events classified as activity_boundary_intermediate_events can only appear on the boundary of an activity (i.e., on the boundary of its graphical representation), the events classified as not_activity_boundary_intermediate_events must not appear on the boundary of an activity, and finally the events which are not classified under any of these concepts can appear in both circumstances. Two BPMNO roles are used to describe the attributes of intermediate events:

- has_intermediate_event_target (hereafter abbreviated in has_target) encodes the “Target” attribute. This attribute is used to describe the fact that the intermediate event is attached to the boundary of an activity.
- has_intermediate_event_trigger encodes the “Trigger” attribute. This attribute is used to describe the type of trigger expected for an intermediate event.

The constraint corresponding to requirement (e) can be formalized in DL by means of the following statement where, for the sake of presentation, we assume to have already defined a BDO:reserve_product_On-line concept used to denote the activities contained in the On-line Shop pool and annotated with the to_reserve_product concept.\(^2\)

\[
\begin{align*}
\text{BDO:reserve\_product\_On-line} & \sqsubseteq \text{BPMN:activity} \\
\exists \text{BPMN:has\_target}. (\text{BPMN:activity\_trigger} \sqsubseteq \text{BPMN:has\_sequence\_flow\_source_ref} \sqcap \\
\text{BDO:to\_read\_policy}) \\
\text{BDO:product\_unavailability} \\
\text{BDO:product\_unavailability\_in\_On-line} \equiv \\
(\text{BPMN:has\_target} \sqcap \\
\text{BDO:reserve\_product\_On-line})
\end{align*}
\]

Requirement (f) can instead be formalized in DL by a statement which makes use of precedence constraints. In the following we use \(\forall \text{BPMN:is\_followed\_by}\) as a shorthand for \(\forall \text{BPMN:has\_sequence\_flow\_target_ref}\).\(\forall \text{BPMN:has\_sequence\_flow\_source_ref}\). Analogously for \(\exists \text{BPMN:is\_followed\_by}\).

\[
\begin{align*}
\text{BDO:product\_unavailability\_in\_On-line} \equiv \\
(\text{BPMN:has\_target} \sqcap \\
\text{BDO:product\_unavailability}) \\
\forall \text{BPMN:is\_followed\_by}
\end{align*}
\]

\(^2\)This can be done using the containment constraints described in Section IV-A.
Axiom (5) defines the class product_unavailability error events caught by the activity of reserving products in the On-line Shop pool, while axiom (6) states that error events of this BDO-type must be followed by a parallel gateway which leads to two activities: a task for warning the buyer, and a sub-process which takes care of ordering the missing products. The encoding of requirement (g) is omitted for lack of space. Note that if we abstract away from the specific activities in requirement (e) we can obtain a pattern of the form

\[
\text{The activity (annotated with BDO concept) } X \text{ has always to catch an error event (annotated with BDO concept) } Y
\]

which can be encoded in an axiom skeleton of the form

\[
\text{BDO:} X \sqcap \text{BPMN:activity} \sqsubseteq \text{BPMN:has_target}^{-}. \\
(\text{BPMN:error_intermediate_event} \sqcap \text{BDO:} Y)
\]

Similar patterns can be devised for other activity boundary intermediate events (e.g., cancel intermediate event). The definition of patterns which specify how exceptions are handled is more complex. This because of the specific ways in which an exception may be handled in a single process. Nevertheless, a number of papers focused on the description of error handling patterns exists (see e.g., [20]). These efforts can provide some guideline on how to define classes of constraints for typical exception handling patterns. Hereafter, we denote with \(SC(\text{BPMNO}, \text{BDO})\) the set of axioms encoding structural constraints (including exception handling constraints).

D. Verification of Constraints

By encoding all the information about a semantically annotated business process into a logical knowledge base, several reasoning services over it can be implemented. Key reasoning services we present in this section are: compatibility checking of process constraints and constraints verification over an annotated BPD.

1) Compatibility Checking of Process Constraints: In formalising the requirements that an annotated business process has to satisfy, the constraints specified by the user may generate inconsistencies in the resulting BPKB. This is due to the introduction of at least a process constraint which is incompatible with the axioms encoded in BPMNO or in BDO, or with other process constraints. The detection of incompatible process constraints can be automatically performed by verifying the consistency of the Tbox component of the BPKB

\[
\text{BPMNO} \cup \text{BDO} \cup \text{MA(} \text{BPMNO, BDO} \text{)} \cup \text{SC(} \text{BPMNO, BDO} \text{)}
\]

with a standard state-of-the-art OWL DL reasoner. In the case of inconsistency, when some unsatisfiable class is detected, the usage of DL reasoners and explanation techniques similar to the ones described in [21] can be also useful to provide justifications to the business experts. In [16], we described an example of explanation obtained by using the Explanation Workbench plugin for Protege-4, and an experimental evaluation on the performance of OWL DL reasoners to check the BPKB consistency.

2) Constraints Verification over an Annotated BPD: Given an Abox \(A_B\) which contains the OWL representation of a semantically annotated BPD \(B\), checking in OWL that the BPD satisfies the process requirements imposed on it requires some carefulness. This because OWL semantics is based on the Open World Assumption (i.e., the knowledge of the world is incomplete, thus a fact cannot be inferred to be false on the basis of a failure to derive it) and does not satisfy the Unique Names Assumption (i.e., two entities with different identifiers are distinct objects), which make it difficult to use OWL for data validation where complete knowledge can be assumed (i.e., a closed world), like in the case of an annotated BPD. For example, OWL allows to encode the requirement (e), as shown in equation (4), but having an activity of BDO-type \text{reserve_product_On-line} with no error intermediate event of BDO-type \text{product_unavailability} attached to its boundary would not cause a logical inconsistency in the BPKB. In fact by reasoning in Open World Assumption a failure of proving that any of the \text{product_unavailability} error intermediate events explicitly mentioned in the Abox \(A_B\) is attached to the boundary of an activity of BDO-type \text{reserve_product} would not imply that this element does not exist.

However, as discussed in [22], it is possible to define an Integrity Constraint (IC) semantics for OWL axioms in order to enable closed world constraints validation: constraints are written as standard OWL axioms but are interpreted with a different semantics for constraint validation. To support the validation of IC in OWL, [22] describes the Pellet IC Validator\(^4\), a prototype tool that extends the Pellet OWL reasoner by interpreting OWL axioms with IC semantics. Technically speaking, each axiom representing an IC can be firstly translated to some SPARQL query, and then executed by a SPARQL query engine over Pellet reasoner to perform the validation over a given set of individuals\(^5\). By reasoning in IC semantics, the existence of an activity of BDO-type \text{reserve_product_On-line} such that none of the error intermediate events of BDO-type \text{product_unavailability} defined in the BPKB is attached to its boundary, would cause a violation of requirement (e).

V. SEMANTICALLY ENHANCED ASPECTS

Beyond the process workflow itself, business processes usually involve several other concerns, often scattered across the whole process and tangled with the main view (crosscutting concerns). Though allowing to represent classic process perspectives, existing process modelling languages do not provide any constructs to describe crosscutting concerns. We

\(^4\)http://clarkparsia.com/pellet

\(^5\)It also provides some basic automatic explanations of why an IC is violated.
consider the use of aspects [6] to cope with this lacking capacity of process languages at design time. By separating crosscutting concerns from the main view of the process, aspects allow business designers to modularize information in separate views, hence easing the job of modelling, reading and understanding process models that involve crosscutting concerns.

An aspect is a separate module that adds behaviour to the principal decomposition of the process by specifying where the behaviour has to be added (“where?”) and what kind of behaviour has to be added (“what?”). In detail a so called pointcut designator answers the “where?” question, by providing a condition that allows to intercept a set of precise points (join points) in the execution flow (quantification). An example of pointcut in BPMN processes, could be a sub-process, thus indicating that the new behaviour has to be added at each sub-process occurrence. An advice, instead, answers the “what?” question by specifying how to realize the concern. An example of advice for BPMN processes could be an intermediate error event directly followed by an end event, to be added on the boundary of the specific sub-process (pointcut). The main view is hence oblivious of (i.e., has no reference to) aspects, that are woven (i.e., integrated) in the core functionality only when needed [6]. This allows the business experts to manipulate each crosscutting concern locally, leaving the weaver with the responsibility of propagating the changes consistently and completely to all process portions that match the change to be applied. For instance, given the aspect example described above, the weaver will enrich each sub-process in the process model with an exception handler terminating the sub-process whenever an exception occurs (i.e., with the error event on the sub-process boundary and with the connected end event).

In order to support business designers in the use of aspects for the modularization of concerns crosscutting BPMN processes, we propose an aspect-based language for BPMN process models based on the BPMN itself, the BPMN VRL (BPMN Visual Rule Language). For the quantification (“where?”), the BPMN VRL exploits the BPMN VQL (BPMN Visual Query Language) [23], an extension of BPMN that supports querying the process and documenting its crosscutting concerns. For the process updates (“what?”), BPMN VRL provides a mechanism for the business process manipulation.

Each BPMN VRL rule is expressed in a visual language which consists of two parts: the “matching” and the “update” part. The first (matching pattern) represents the pattern to be matched (in terms of graph matching and domain semantics). It looks like a BPMN VQL query, in which the selection sub-pattern (i.e., the subset of matching pattern components, whose occurrences are returned to the user) is not explicitly represented with a darker background (as in BPMN VQL queries) but it is inferred from the “update” part of the rule. In detail, the selection sub-pattern is the subset of matching pattern components directly involved in the modification (e.g., BPMN elements to be removed). The “update” part, with a darker background, thicker lines and bold font style, represents the modifications (behaviour addition or removal) to apply whenever a match occurs. The addition to the process of new behaviour is represented by using BPMN elements (with a darker background and thicker lines), while the REMOVE operator is introduced for representing the removal of one or more BPMN elements. The REMOVE operator is depicted as a filled cross over the BPMN elements to be removed. In the following some examples of the use of BPMN VRL are provided.

Figure 4a shows an example of a BPMN VRL rule that inserts a message intermediate event in the direct connection between a task of BDO-type concept_A and another task. The BPMN elements with white background and thinner lines (i.e., the two tasks connected by at least a sequence flow and the sequence flow itself), whose domain semantics is specified by the annotations (the first task is of BDO-type concept_A), represent the pattern to be matched against the process. The BPMN elements with darker background and thicker lines (the message intermediate event and the two sequence flows connecting it to the two tasks) represent instead the behaviour to be added to the process, whenever the pattern matches. Finally, the BPMN VRL REMOVE operator, specifies the BPMN elements of the matching pattern to be removed (the sequence flow connecting the two tasks). In this case, since all the elements in the matching pattern (the tasks and the sequence flow) are directly involved in the modification, the selection sub-pattern is the whole matching pattern.

The mechanism used for quantifying over the process can be enhanced by exploiting stereotypes (i.e., categories of BPMN elements graphically represented as the name of the BPMN category class enclosed by guillemots) and composition operators provided by the BPMN VQL. The BPMN VQL operators can be classified into three groups: (a) operators for the composition of domain semantic concepts; (b) operators for the logical composition of sub-patterns; and (c) the operator for the structural composition of sub-patterns. The first group is made of the classic logical operators: \( \land \) (AND), \( \lor \) (OR) and \( \neg \) (NOT), denoting respectively the intersection, the union and the complement of more/one domain concepts/concept. The second group includes (i) the OR operator (depicted as a dotted table, listing a set of possible alternatives to match), that allows to represent the union of the sub-patterns to be matched; and (ii) the NOT operator (depicted as an empty cross over the negated elements) that allows to denote the complement of a sub-pattern with respect to the universe of the non-negated part of the pattern to be matched. Finally, the PATH operator (depicted as a BPMN sequence flow with double head) represents the existence of at least a path between the pair of flow objects that it connects.

An example of their use is shown in the rule in Figure 4b. It has the same selection sub-pattern and advice of the rule in Figure 4a. However, the matching criterion for the pointcut identification is restricted. The task of BDO-type concept_A has to be directly preceded (via sequence flow) by either (OR operator depicted as a dotted table) (i) an activity (i.e., task or subprocess) of BDO-type concept_B but not concept_C (concept_B \( \land \neg \) concept_C); or (ii) a data-based XOR gateway having no outgoing sequence flow other than the one

\[ \text{For a more detailed and structured description of the language the interested author can refer to our Technical Report [24].} \]
a task and the second is the sequence flow connecting them. By running the query on the process in Figure 5, for example, we get two sets of results: \( \{t_1, s f_3, t_2\} \) and \( \{t_3, s f, t_4\} \). By executing the query corresponding to the “matching” part of the rule in Figure 4b on the same process, instead, only the second set of instances is returned.

The “update” part of the rule builds upon each set of instances resulting from the query execution. In detail, for each set of instances returned by the query: (i) a set of new BPMN-type, BPM-structural, BPM-semantics assertions is added to the BPKB according to the elements with darker background and thicker lines in the rule; (ii) a set of assertions in the BPKB is removed (i.e., all the assertions involving BPD instances associated to elements to be removed according to the BPMN VRL rule). For example, considering the first set of results of the SPARQL query (7), some new BPMN-type assertions (e.g., intermediate_message_event(me1), where me1 is the new instance of message event) and some new BPMN-structure assertions (e.g., has_sequence_flow_source_ref(s_f9,t_1)) are added. Finally, all the assertions related to s_f2 (e.g., sequence_flow(s_f2)) are removed.

We developed a tool for the automated weaving of aspects in the BPKB (aspectisation tool). It retrieves the BPD instances satisfying the matching criterion by querying the BPKB and it exploits the population tool for managing the modifications in the rule advice. Once business experts have separately modelled the desired aspects as BPMN VRL rules, they can generate the woven process by providing the aspects as input to the aspectisation tool, that parses them and populates the BPKB accordingly.

VI. EXCEPTION HANDLING ASPECTS

In business processes, exception handling represents a typical example of crosscutting concern. The exception handler can be tangled in different scattered points of the process, thus increasing the process complexity, when explicitly managed. Hence, though processes meeting exception handling requirements have higher robustness ([4]), business designers often focus only on the “happy path”, to make the process model easier to understand.

Aspects provide a possibility to take out the complexity added to the “happy path” by the exception handling [7]. Business experts can modularize exception handlers into aspects defined in BPMN VRL, thus separating them from the “happy path” and hence ensuring a better readability. Aspects can then be woven only when needed, e.g., for the constraint verification of the whole process.

For example, with reference to the resource unavailability exception, the BPMN VRL aspect describing a possible exception handling (compliant with the requirements (e) and (f) presented in Section II) is reported in Subfigure 6a. A product_unavailability intermediate error event has to be added on the boundary of activities reserving products in the Online Shop pool, and it has to be handled by warning the buyer and ordering the unavailable products. The matching criterion looks for process elements that require exception handling mechanisms: it intercepts occurrences of On-line

connecting the gateway to the concept_A task (NOT operator).

Once a BPMN VRL rule has been defined by a business expert, in the rule weaving phase the changes described in the “update” part of the rule are automatically applied to the process (i.e., to the knowledge base and hence to the BPD), by changing all occurrences in the process satisfying the matching criterion. Operationally: (i) the occurrences of the semantically annotated process satisfying the matching criterion and specified by the selection sub-pattern are identified by querying the knowledge base encoding the process (the BPKB); (ii) for each retrieved occurrence, the modifications specified in the “update” part of the rule are applied, by adding/removing instances to/ from the BPKB.

In detail, the “matching” part of the rule is translated into a SPARQL query by: (i) associating a SPARQL variable to each BPMN element in the matching pattern of the rule and constraining it by means of one or more triple patterns in the SPARQL WHERE clause; (ii) using SPARQL constructs for translating BPMN VQL operators (e.g., SPARQL FILTER and OPTIONAL for the NOT operator); (iii) specifying in the SPARQL SELECT clause the variables corresponding to the BPMN elements in the selection sub-pattern of the rule.

The SPARQL query obtained by formalizing the matching criterion of the rule in Figure 4a is the following [7]:

\[
\text{SELECT} \ ?kl, \ ?seqf, \ ?k2 \\
\text{WHERE} \ \\
\{ \ ?kl \ \text{rdf:type} \ \text{BPMNO:task} \\
\ ?k2 \ \text{rdf:type} \ \text{BPMNO:task} \\
\ ?seqf \ \text{rdf:type} \ \text{BPMNO:sequence_flow} \\
\ ?kl \ \text{rdf:type} \ \text{BDO:concept_A} \\
\ ?seqf \ \text{BPMNO:has_sequence_flow_source_ref} ?k1 \\
\ ?seqf \ \text{BPMNO:has_sequence_flow_target_ref} ?k2 \} \\
\]

It retrieves all the triples of BPD instances such that the first element is a task of domain BDO-type concept_A, the third

\(7\) For space reasons we omit the SPARQL query for the rule in Figure 4b.
Shop activities reserving products (i.e., annotated with the semantic concept `to_reserve_products`). Moreover, the matching criterion locates the process elements required by the aspect advice (e.g., the event-based exclusive gateway originating from the start event of the On-line Shop pool). The advice of the aspect describes the catching of the exception and its management: the caught exception is handled by means of a parallel gateway with two outgoing edges, one connected to a `to_order_product` sub-process (followed by an end event) and the other connected to a `to_warn_buyer` activity followed by the initial event-based gateway.

Subfigure 6b describes a second example of aspect related to the log-in failure exception (according to the exception handling requirement (g) described in Section II). This rule has a semantics more complex than the one of the previous rule and it involves both the Customer and the On-line Shop pool.

With regards to the On-line Shop pool, the pointcut intercepts the occurrences of the activities checking customer data, contained in an authentication sub-process and for which it exists at least a path (the sequence flow with double head) from a message intermediate event (contained in the same authentication sub-process) generated by the `to_send_customer_data` activity in the `to_log_in` sub-process of the Customer pool. Once all the occurrences of activities responding to this criterion have been identified, the action to take is chosen according to the type of flow object following the activity checking the customer data. Three possible different cases may occur (see the three cells of the dotted table in the On-line Shop pool in Subfigure 6b):

(a) the activity checking the customer data is directly followed by a flow object that is not a data-based exclusive gateway (top cell of dotted table in Subfigure 6b). In this case: (i) the sequence flow connecting the `to_check_customer_data` activity and the non-gateway flow object is removed (filled cross); (ii) a data-based exclusive gateway is added; (iii) the new gateway is connected through an incoming sequence flow to the `to_check_customer_data` activity; (iv) the new gateway is connected through two outgoing sequence flows to the non-gateway flow object and to a new activity notifying the login failure to the Customer pool, respectively.

(b) the `to_check_customer_data` sub-process is followed by a split data-based exclusive gateway (i.e., an exclusive gateway with only an incoming edge). The middle cell of the dotted table in Subfigure 6b is matched, hence only a sequence flow connecting the gateway to the new activity notifying the login failure is added.

(c) the `to_check_customer_data` activity is directly followed by a data-based exclusive gateway with more than one incoming sequence flow (bottom cell of dotted table in Subfigure 6b). As in the first alternative: (i) the sequence flow connecting the `to_check_customer_data` activity and the data-based exclusive gateway is removed (filled cross); (ii) a new split data-based exclusive gateway is added; (iii) the new gateway is connected to the `to_check_customer_data` activity; and (iv) the new gateway is connected to the original exclusive gateway and to the new “Notify login failure” activity, respectively.

In all three cases, the added “Notify login failure” activity is followed by a `login_failure` end event, terminating the sub-process in which it is contained.

In the Customer pool, instead, the aspect manages the `login_failure` intermediate event generated by the On-line Shop pool. Similarly to the On-line Shop pool, according to the type of flow object immediately following the `to_send_customer_data` activity in the `to_log_in` sub-process, a different behaviour is specified in order to catch the failure event. Moreover, the `to_log_in` sub-process containing the received `login_failure` event has to catch the event and manage it by terminating the (sub-)process.

![Product unavailability aspect.](image)

![Compulsory log-in failure aspect.](image)

**Fig. 6:** Aspects handling exceptions in BPMN.

At weaving time, all the scattered occurrences captured by the quantification part of each rule will be identified and modified according to the corresponding advice. For example, the aspect handling the product unavailability will exploit the process semantic information in order to match the sub-process “Reserve products” and it will apply on it the corresponding modifications (as shown in Figure 7).

A generic category of exception handlers can be defined for a class of activities that are likely to require that kind of handlers. Aspects are specializations of such exception handler categories, defined according to the specific needs of the designers and of the process itself. For example any `to_log_in` activity is likely to require a `login_failure` handler,
that can specialized into different aspects. Well known exception handling strategies could be specified as reusable BPMN VRL aspect libraries.

A. Using semantic constraints to support aspect definition

After defining the exception handling aspects in BPMN VRL, business designers can verify that the initial exception handling requirements, translated into semantic constraints, are satisfied on the woven process. In addition to that, the exception handling constraints can be useful to support business designers also in the definition of the exception handling aspects. Semantic constraints are thus useful both before and after aspects are defined by business designers.

The definition of exception handling aspects can be based on the verification of the constraints originated from the exception handling requirements. In fact, constraint verification does not only return the process occurrences violating the constraints (if any), but it also suggests whether and where modifications have to be applied in order to solve constraint violations. For instance, given a violated inclusion axiom, by exploiting the concept on the left of the inclusion as matching criterion and the part on the right for the advice, a skeleton aspect (i.e., a starting point for the definition of the exception handling aspect) can be automatically generated. For example, in the process in Figure 1, the requirement (e) is violated by the sub-process “Reserve product/s”. A skeleton aspect can be automatically generated starting from the violated inclusion axiom (1). The generated aspect will capture all the reserve_product_On-line instances (i.e., all the to_reserve_product activities in the On-line Shop pool) and it will add a product_unavailability intermediate event on its boundary. By taking advantage of the visualization of the process elements violating the constraints, the business designer can complete this skeleton aspect in order to handle the exception according to the requirements. The skeleton aspect automatically produced from the violated constraint can describe the exception handling with different levels of detail according to the type of violated constraint, ranging from just exception catching (“where?”) to the complete exception handling (“what?”), as in case of the skeleton aspect that is generated by the violation of the constraint (f). Constraint verification on the woven process will finally check whether all the constraint violations have actually been solved.

VII. Experimental Evaluation

We performed some experiments in order to provide a first evaluation of the performance of semantic reasoning techniques used to support the management of exception handling over annotated BPD. In particular, the goal of the evaluation was to provide an estimate of the impact on the business process modelling activities of (i) checking the consistency of the BPKB, (ii) transforming an annotated BPD into an OWL Abox, (iii) checking constraints verification over the BPKB, and (iv) weaving the aspects in the BPKB. This extends the preliminary evaluation we presented in [16], in which only the time for computing the consistency (and classification) of the BPKB was considered.

The evaluation study that we conducted comprised two experiments. In the first experiment we considered six different processes of increasing size (with a number of process graphical elements ranging from 92 to 475), and, for each one of them, a single exception handling process requirement (of the same kind of requirement (e)). The purpose of this first experiment was to study the performance of the semantic technology based tools used, as the size of the BPKB (in terms of instances) grows. The main characteristics of the domain ontologies used to annotate the processes are reported in the top rows of Table I, while the DL expressivity of the BPKBs considered is $\text{ALCHQI}^0(\mathcal{D})$.

The first experiment consisted of five phases organized as follows. First, we checked the consistency of the Tbox of the BPKB (Consistency Phase). Second, we ran the population tool to transform an annotated BPD into an OWL Abox (Population Phase). Third, we validated the BPKB against the constraint considered, to check whether the given process satisfied or not the requirement (Validation Phase I). Fourth, once the appropriated aspect handlers have been selected for the process concerns violating the requirements, we ran the aspectisation tool to weave the aspect parts into the main process in the BPKB (Aspectisation Phase). Finally, we validated again the BPKB against the constraint considered, to check whether the process, integrated with the woven parts, satisfied the exception handling requirement imposed on the process (Validation Phase II). The reasoning tasks required in each phase have been performed with the support of the Pellet reasoner (v2.0.2), integrated with the Pellet IC Validator (v0.4) for the constraint validation tasks.

The results of this first experiment are reported in the lower half of Table I. As shown by the results, the most demanding phases in terms of performances are the constraint validation ones, for which the computation time increases considerably as the size of the process (and, hence, of the BPKB) grows.

8 The machine used for all the experiments is a desktop PC with an Intel Core i7 2x2.80GHz processor, 6Gb of RAM, and running Linux Red Hat 5.
9 We recall that checking the consistency of an $\text{ALCHQI}^0(\mathcal{D})$ ontology is an NEXPTIME-hard problem.
10 The time values reported in Tables I and II are in the form $\text{avg (sd)}$, where $\text{avg}$ and $\text{sd}$ are respectively the arithmetic mean and the standard deviation of the execution times of 100 runs on the same input data.
In the second experiment we considered a single process (Process D of the first experiment) and an increasing number (1, 5, 10, 50, and 100) of process constraints (of the same kind of requirement (e)). The purpose of this second experiment was to study the performance of the constraint validation phase as the number of process constraints grows. The results of the second experiment are reported Table II10. As shown by the results, the number of constraints to validate do not significantly impacts on the performance of the constraint validation phase.

Given the results of this initial evaluation, we can conclude that these experiments show the applicability of the current set of (state of the art) tools, which are compatible with an on-line usage at modelling time on small / medium size processes.

VIII. RELATED WORK

We can roughly divide the existing proposals for adding semantics to business processes into two groups: (1) those adding semantics to specify the dynamic behaviour of a business process [9], [10], [11], and (2) those adding semantics to specify the meaning of the entities of a BPD in order to improve the automation of business process management [12], [13], [3], [14], [25]. We clearly belong to the second group.

Thomas and Fellmann [3] consider the problem of augmenting EPC process models with semantic annotations. They propose a framework which joins process model and ontology by means of properties (such as the “semantic type” of a process element). Markovic [26] considers the problem of querying and reasoning on business process models. He presents a framework for describing business processes which integrates functional, behavioural, organizational and informational perspectives: the elements of the process are represented as instances of an ontology describing the process behaviour (based on π-calculus), and the annotations of these elements with respect to the ontologies formalizing the aforementioned perspectives are described as relation instances. Born et al. [27] propose to link the elements of a business process to the elements of an ontology describing objects, states, transitions, and actions. These proposals differ substantially from ours, which establishes a set of subsumption (aka subclass or is-a) relations between the classes of the two ontologies being integrated (BPMN meta-model and domain ontology), instead of associating annotation properties to the process instances. This difference has a direct impact on the kind of constraints that can be automatically enforced (e.g. BPMN elements annotatable by domain concepts).

De Nicola et al. [14] propose an abstract language (BPAL) that bridges the gap between high-level process description (e.g. in BPMN) and executable specification (e.g. in BPEL). The formal semantics offered by BPAL refers to notions such as activity, decision, etc., while the problem of integrating process model and domain ontology is not their focus. In the SUPER project [13], the SUPER ontology is used for the creation of semantic annotations of both BPMN and EPC process models in order to support automated composition, mediation and execution. Di Noia et al. [28] semantically annotate building blocks (BBs), i.e., flexible and transparent pieces of functionality within ERP (Enterprise Resource Planning) systems. By exploiting standard and non-standard reasoning services, they provide a framework that automatically select the set of BBs needed for satisfying a requested business process and, if it does not exist, provides explanations on what is in conflict and what is still missing to cover the request. Recently, Groener and Staab [25] presented a pattern-oriented approach in which OWL representation and reasoning capabilities enable expressive process modelling and retrieval. Their process formalisation considers the primitives of the UML-Activity Diagram, a language which allows to express simpler processes than BPMN, and the connection with the domain knowledge involves the representation of terminological knowledge about activities and sub-activities only. Our work represents an extension of the existing literature in that we provide an ontology based approach that supports automated verification of semantic constraints about BPMN process entities and structure, their relations with domain specific concepts, as well as the way exceptions are handled.

Among the requirements business designers are interested to ensure, exception handling holds a crucial role for enhancing the process robustness since the modelling phase. With the aim of supporting process designers in the correct management of exceptions, several works in the literature have investigated exceptions and their handling. Some of them classify exception...
processors and their formalization. We will also study how
to detect possible specification violations, a set of constraints
describing the desired exception management. For
the first group, for example, Dellarocas and Klein [4]
propose the use of a reusable and extensible body
of knowledge describing and classifying exceptions and their
handlers for detecting, diagnosing and resolving exceptions.
Eliahu and Elhadad [30] infer the existence of likely errors by
analysing the structure of the process and some semantic information
added to process activities in the form of semantic tags.
In our work we exploit semantics both for verifying the correct
handling of exceptions and for modularizing their management
into aspects. The use of aspects for exception handling has been
deployed in the AOP literature, for example for
the AOP refactoring of object oriented code [7].

The AOP approach has also been applied to process execu-
table description languages. AO4BPEL [31] is a dynamic
aspect oriented extension of BPEL [32] (Business Process Execution Language), designed to be as close as possible to the
AOP programming language AspectJ, thus providing similar concepts for describing pointcut designators, join points
and advices. Similar approaches, mainly differing from AO4BPEL
for the choice of the advice language (Java), have been proposed by Courbis and Finkelstein [33] and by Verheeecke,
Cibrán and Jonckers [34], who introduced WSML (Web Service Management Layer). Padus [35] by Braem et al. is another
aspect-oriented BPEL extension, without dynamic weaving and with some improvements that increase portability.

Our use of aspects differs from these works mainly for the
focus on the designers’ perspective rather than on the develop-
ers’ one. In practice, business experts prefer a higher level
modelling notation, such as BPMN, to a process executable
language and the benefits provided by the use of aspects
at modelling time, as for example separate representation of exception handling concerns, are even more valuable.

IX. CONCLUSIONS

In this paper we have presented an ontology-based aspect-
oriented approach to manage exceptional flows and exception
handling in business process models. Our approach includes
both the specification of exception handling mechanisms as
separate aspects (that can be woven, if needed, into the
main process) and the automated verification of specific con-
straints describing the desired exception management. For
the aspect definition we have proposed a visual language to
support designers in separating exceptional behaviours from
the “happy path”. Moreover, to help business process analysts
to detect possible specification violations, a set of constraints
involving both knowledge about the domain (e.g., policies
for the management of exceptional flows), and the process
structure and semantics, are formalized in DL and verified via
DL reasoners.

In our future work we will investigate in more detail generic
and reusable classes of constraints for exception handling
patterns and their formalization. We will also study how to
simplify the tasks of requirement specification and checking by
means of user-friendly notations and tools. Finally, we intend
to validate our approach further, on additional case studies.

ACKNOWLEDGMENT

Some of the ideas presented in this paper were first explored
while the first author was a Visiting Researcher in the Manag-
ning Complexity Theme at NICTA and she would like to thank
the Centre for its hospitality and fruitful discussions. NICTA
is funded by the Australian Government as represented by the
Department of Broadband, Communications and the Digital
Economy and the Australian Research Council through the
ICT Centre of Excellence program.

REFERENCES

mantic business process management: A vision towards using semantic
web services for business process management,” in ICEBE’05: Proc. of
[2] B. Wetzstein, Z. Ma, A. Filipowska, M. Kaczmarek, S. Bhiri, S. Losada,
J.-S. Lopez-Cob, and L. Cicirello, “Semantic business process manage-
ment: A lifecycle based requirements analysis,” in Semantic Business
culing using ontology languages,” in Semantic Business Process and
Product Lifecycle Management, ser. CEUR, June 2007, pp. 64–75.
robust business processes,” in Business Process Management, Models,
Proceedings of the CAiSE-05 Workshops, ser. LNCS. Springer,
semantic business process modelling environment,” in Semantic Business
to business process modeling,” in Proc. of the 3rd Indian International
Conference on Artificial Intelligence (IICAI), 2007, pp. 1794–1813.
of the 6th Int. Conference on Service Oriented Computing (ICSOC’08),
BPMMN/1.1/PDF.
Proceedings of the Workshop on Semantic Business Process and Product
Lifecycle Management (SBPM), June 2008.
of the 1st Int. Workshop on the Foundations of Coordination Languages
and Software Architectures (FOCLASA08), 2008.
tion,” in Proceedings of the CAiSE-05 Workshops, ser. LNCS. Springer,
semantic business process modelling environment,” in Semantic Business
to business process modeling,” in Proc. of the 3rd Indian International
Conference on Artificial Intelligence (IICAI), 2007, pp. 1794–1813.
[19] C. Di Francescomarino, C. Ghidini, M. RosPOCHER, L. SERAFINI, and
of the 6th Int. Conference on Service Oriented Computing (ICSOC’08),
BPMMN/1.1/PDF.

mantic business process management: A vision towards using semantic
web services for business process management,” in ICEBE’05: Proc. of
[2] B. Wetzstein, Z. Ma, A. Filipowska, M. Kaczmarek, S. Bhiri, S. Losada,
J.-S. Lopez-Cob, and L. Cicirello, “Semantic business process manage-
ment: A lifecycle based requirements analysis,” in Semantic Business
culing using ontology languages,” in Semantic Business Process and
Product Lifecycle Management, ser. CEUR, June 2007, pp. 64–75.
robust business processes,” in Business Process Management, Models,
Proceedings of the CAiSE-05 Workshops, ser. LNCS. Springer,
semantic business process modelling environment,” in Semantic Business
to business process modeling,” in Proc. of the 3rd Indian International
Conference on Artificial Intelligence (IICAI), 2007, pp. 1794–1813.
of the 6th Int. Conference on Service Oriented Computing (ICSOC’08),
BPMMN/1.1/PDF.


