

Semantic business process for improved exception handling

Ziv Ben-Eliahu and Michael Elhadad

Department of Computer Science
Ben Gurion University of the Negev
P.O.Box 653, Be'er Sheva 84105, Israel
(ben-elia|elhadad)@cs.bgu.ac.il

Abstract. The execution of a business process (BP) often interacts with multiple independent entities, whose behavior cannot always be predicted. This is why, although a business process may have a single ideal execution path, in practice, many executions will encounter events, errors or missing deadlines, that lead the process off this path. Exception handling is not a favorite issue for programmers or analysts. They often focus on the likely or ideal business scenarios, and end up ignoring error handling.

The goal of this paper is to facilitate the design of complete business processes that have appropriate built-in error handling at modeling time. Such support requires identifying potential errors from the semantic and structural aspects of the business process. The method we present relies on the explicit annotation of activities (the primitive building blocks of BPMs) with a small number of semantic tags. We also classify the function of each activity within a BPM as part of a small number of process schemas. We infer the existence of potential errors from the analysis of the structure of the BPM and the semantic attributes of the activities and sub-processes the BPM includes. We have implemented this analysis tool as part of the BPM modeling tool of the Prosero system, which relies on a semantic repository of business activities. We present an empirical analysis of a dataset of 11 complex BPMs including more than 100 activities. We show that the proposed error specification effectively helps analysts capture at modeling time a high proportion of the key business error conditions in these models, without alerting for too many false positives.

Topics: Process modeling methods, metadata, and semantics

1 Introduction

Business Process Models (BPMs) are explicit descriptions of the methods used by a business to perform routine processes. The value of such explicit descriptions is that it can be shared by all stakeholders, it enables monitoring, identification of bottlenecks and inefficiencies and refactoring. In recent years, BPMs also serve as abstract specifications of executable workflow applications, using executable orchestration languages such as BPEL within Service Oriented Architectures.

We address in this paper the issue of handling error conditions that occur while executing processes. Our initial assumption is that exceptional business conditions and ways to handle them should be explicitly specified as early as possible as part of complex BPMs. We distinguish between technical exceptions (*e.g.*, network failures) and business exceptions (*e.g.*, customer does not pay in time, delivered package is lost). The identification of such potential execution failures should not be left as an implementation detail.

We study the way such business exceptions are expressed in a dataset of complex business processes. We present these models using the Business Process Modeling Notation (BPMN) [1], which has acquired the status of standard in recent years as the modeling language for business processes.

BPMN includes the *Intermediate Error Event* element, to allow the business analyst to describe an exception in the workflow or the handling of an “unexpected” exception event. For example, a BPMN diagram could express using this construct that in case you fail to deliver a package, call the recipient. Identifying expected exceptions is an important part of BPMs specification to produce safe and robust business processes.

Unfortunately, the Intermediate Error Event, and its related Exception Flow, are extremely uncommon in present BPDs [2]. Our empirical analysis confirms this observation. Four claimed reasons for this situation are: (a) current modeling tools are unable to execute exception handling [3]; (b) business analysts are not equipped with the required training [3]; (c) sometimes, exceptions are deliberately excluded from the business process to reduce complexity [4]; and (d) it is common practice that exceptions are dealt with off-line, outside of the system (by a human agent) [5].

We address the first two issues in this paper: we design a modeling tool that supports exception handling and compensates for the required training by actively recommending relevant exceptions as the analyst proceeds.

The alternative to not specifying exceptions is to either specify them as regular alternate execution flows (using BPMN conditional Gateways) or to simply leave them out of the scope of the models and keep them implicit. We believe both approaches are less attractive than explicit specification, at modeling time, as exceptional events. The specification as exceptions allows the modeling tool to hide the exceptions and focus on the happy path of execution.

Our assumption is that for complex business processes, the business analyst is more likely to miss a probable business level exception as he is focused on a long “happy path”. In this paper, we hypothesize that by giving the business analyst a small set of possible exceptions, appropriate to the type of the business activity, we will encourage him to consider possible exceptions. The major issue this research faces is how to produce this set of possible exceptions — how to infer the exceptions from semantic data, such as the Activity’s type — and how to obtain the semantic data in the first place.

We test our hypothesis by implementing a BPMN modeling tool and evaluating it on 11 business processes, of various complexity levels. The tool is implemented as part of the Prosero architecture [7]. Our tool suggests possible

semantic classification for a business activity, then infers potential exceptions, relevant to the classification and to the structural location of the activity within the BPM, and, finally, warns if selected exceptions are not handled.

Our empirical evaluation indicates that while the current wizard-based implementation is not user-friendly enough, it shows great promise by producing more robust business processes. We identify specific ways of improving the implementation with domain-specific heuristics and usability features (such as filtering exceptional events in diagrams).

In the rest of the paper, we briefly present in Sect. 2 an overview of BPMN with focus on exception specification, we present the way semantic information is acquired and exploited in the Prosero BPM architecture. Section 3 presents our method of exploiting semantic annotations of business activities used in BPMs to infer potential business exceptions. We describe our implementation in Sect. 4. Finally, we evaluate the benefits of the approach in Sect. 5.

2 Background

2.1 BPMN Overview

In the rest of the paper, we use BPMN to discuss how exception handling is specified in BPMs. We first shortly describe the most basic components of BPMN. Version 1.0 of the Business Process Model Notation [1] final specification was published in February 2006. According to BPMN, a business process diagram (BPD) is made up of a set of graphical elements. The three most important BPMN elements [8, 9] are the Activity — a rectangular shape; the Gateway — a diamond shape; and the Event — a circular shape. These elements are connected by a Connection element — a line with an arrow. The three elements have different sub-types, graphically depicted by an additional icon inside the basic shape.

Activity depicts a decision making point within a process. Some data is being prepared or reviewed, or, some action is being taken by a person or a machine. For example, “Ship Order” is a physical action, while “Send Invoice” can be performed by automated e-mail.

Gateway is either a conditional decision, splitting the sequence to several continuations, or a point where several paths join together. For example, the “Accepted or Rejected?” Gateway is an exclusive decision with one path for each scenario. A second Gateway can later join the two parallel paths.

Events are one of the innovations of BPMN. They mark a point where the process sends a message, or a signal, to another process. Alternatively, they can mark a point where the process is waiting for a message/signal. For example, a process is usually initiated (signaled to start) by a **Start Event**; and is terminated (signals its completion) by an **End Event**. An “Accept Payment” Activity could be replaced by an **Intermediate Event** — waiting for the message “payment was accepted” if the payment activity was processed by an external party.

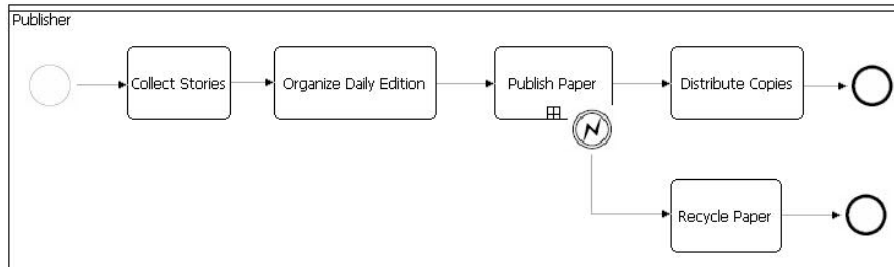


Fig. 1. BPMN example: Publisher (tynerblain.com).

Figure 1 (from [10]) is an example which depicts a simplified business process for publishing a magazine. We will focus only on the notation for exception handling. For that, we will first discuss what exception handling is.

2.2 Exceptions

Performing business activities, by a machine or a human, has the potential to fail, mostly due to unexpected conditions. This failure is called, in the software industry, an exception. An exception can also be considered a deviation from the expected control flow [5]. If an exception, in an orchestrated workflow, is not caught and handled it will propagate upwards and the entire workflow might fail — that means the business process execution failed. If not properly handled, such failures lead to costly delays and frustrations.

A common taxonomy [11] defines five types of exceptions in business workflow: Work item failure, deadline expiry, resource unavailable, external trigger, and constraint violation. A higher level categorization [8] distinguishes business exceptions from system faults, which are technical issues within the process runtime execution environment.

In this paper, we focus on the constraint violation forms of business exceptions — business data/meta-data violations — which we consider true business level exceptions.

2.3 Exception Handling in BPMN

An exception, at modeling time, is a construct which refers to either catching or throwing an exception event. In BPMN, the Intermediate Events constructs have Triggers that define the cause for the event. The “Error” trigger is used for error handling — both to set (throw) and to react to (catch) errors. It sets (throws) an error if the Event is part of a Normal Flow. It reacts to (catches) an error when attached to the boundary of an Activity (Task or Sub Process).

Handling an exception means catching an Error Event (the catching element is called the handler) and continuing with an Exception Flow (see Fig. 2). For

example, the “Recycle Paper”, in Fig. 1, is part of the Exception Flow, while “Distribute Copies” is part of the Normal Flow (a.k.a., Happy Path).

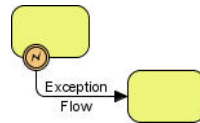


Fig. 2. BPMN Graphical Notation: Exception Flow.

2.4 Exception Handling in Programming Languages

Popular programming languages, such as Java, offer compile-time support to make sure specified exceptions are handled. Any methods are annotated with the exceptions they may throw. Any method invoking a method that throws an exception must either handle the exception locally, or specify that it may throw it (thus relegating the responsibility to handle it to its own callers). In most Java-oriented editors, calling a method which can throw an exception will trigger a warning to remind the programmer he needs to address the exception in one of the two possible ways (catch it or propagate it), as shown in Fig. 3.

```
public String getName() throws Exception {
    throw new Exception();
}

public void myMethod() {
    String name = getName();
}
```

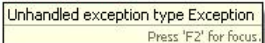


Fig. 3. Unhandled exception notification in a Java editor.

Such exception validation and the associated warning at programming time form our original motivation — to have BPMN modeling tools generate similar warnings for unhandled exceptions and encourage the business analysts to handle exceptions.

2.5 Semantic Business Process Modeling in Prosero

With the adoption of Service Oriented Architectures (SOA), BPMN and BPEL, software-based modeling tools emerged, to help modeling, designing, transforming, and executing business processes as part of end to end modeling to runtime architectures.

The Prosero project [7] relies on the analysis of gaps between the level of analysis used by business analysts (at the business strategy level) and software developers (at the technical level). To bridge this gap, the Prosero project introduces a set of reference models, which are semantically annotated and describe the activities and data objects manipulated by typical business processes. Business analysts rely on a **semantic repository** to compose composite business processes that can be safely mapped to executable SOA Web Service orchestrations.

A business analyst defines a process, for a customer, using a prepared reference model repository (RMR). This repository includes predefined tasks and data-objects to be used in building the business process. The defined process is entered into a customer-specific model repository (CMR). The analyst may then customize the business process according to the customer's specific mode of operation. The Prosero system will then transform the customized business process into a SOA process-sequence of web services. The composed web services are taken from a Web Service Repository (WSR) and are selected by the Prosero Matcher to fit functional and non-functional requirements of the overall customer-specific business process.

In this paper, we specifically address the issue of identifying potential business exceptions at modeling time on the basis of semantic annotation of activities. The Prosero repository contains a rich description of activities, with semantic annotations. Our general approach is to exploit this semantic infrastructure for the purpose of inferring potential business exceptions.

2.6 Related Work

Most of the current research on business process exception handling focuses on automatic exception handling [5, 12–14] at execution time, or on exception handling in an executable language [15–17], such as BPEL.

On the modeling side, many have noticed the problem in analyzing semantic correctness of BPMN/workflow models, especially in regard to exception handling [11, 18, 19]. The solutions proposed generally treat workflow patterns as semantic data, and suggest verification methods relying on abstract models with verification semantics (e.g., Petri Nets).

While we found very few solutions that extend BPMN with semantic data, in order to allow for better analysis and verification of models [20], we did not encounter any attempts to infer possible exceptions from semantic data, which is not pattern related.

3 Suggesting Exceptions and Enforcing their Handling

Our starting point of analysis corresponds to a standard, repository-based business process modeling environment: a business analyst works on a BPM, using a graphical modeling tool. The business process contains BPMN Activities referencing Activities in a shared repository such as Prosero's customer model repository. An Activity might throw a business level exception — an Error Event with

a business impact. But, (a) most business analysts don't consider the exception path and (b) current modeling tools cannot indicate what Error Event might occur, if any.

Our objective is to encourage the analyst to (1) identify potential business exceptions at modeling time and (2) specify how to handle them. Our strategy is that the modeling tool will indicate to the business analyst probable occurrences of exceptions, and as a result, the business analyst will consider the exceptions and decide how to handle them. We must be careful not to harass the business analyst with too many exceptions, as this would delay him, antagonize him, and clutter the diagram with redundant unnecessary exception flow.

3.1 Extend BPMN Activity

To achieve our goal, we propose to extend BPMN by adding three attributes to an Activity's meta-data: *Parameters*, *Errors*, and *Handlers*. *Errors* specify a list of expected exceptions (Error Events) for the Activity, and *Handlers* specify the exception handlers that are attached to the Activity. Using these two attributes, a static verification procedure can warn against unhandled Error Events (in the same way as a Java compiler performs such error handling analysis). Warnings are raised when an *Error* item occurs without a matching *Handler* item within a given BPM. The purpose of the *Parameters* attribute is to annotate the activity with semantic information that helps us infer potential errors.

The idea is simple — the business analyst, as he works on the happy path, will quickly add possible Error Events to the Errors attribute and continue his work on the happy path. Later on, the tool will warn against unhandled Error Events and the business analyst will return to the Activity and add appropriate Event Handlers at the appropriate structural level of the BPM, thus completing the missing exception paths.

3.2 Classify BPMN Activity

The next step, and our main objective, is to find ways to suggest a limited list of possible Error Events, relevant to a specific Activity. If we can produce a list of relevant potential errors when an activity is entered in a BPM, the business analyst will gain the benefits of the recommendation and of faster data entry. To this end, we encourage the business analyst to classify the Activity. Classification of a business activity characterizes its general run-time behavior. Such classification adds a **business semantics** information to the Activity's meta-data.

From the classification, one can infer possible run-time exceptions. We hypothesize that it is faster and easier to review an Activity's classification than its explicit list of exceptions. The selected classification will be saved in the activity's *Parameters* attribute.

To determine a valid limited list of standard classifications, we reviewed 9 business processes [21–23] with 76 business activities. Table 1 provides a sample

review of our findings. The sample details all the activity classes we encountered. The complete table can be found in our full research [6].

From this analysis, we derive business semantics classifications and their relations to Error Events, as detailed in Table 2.

The user should be able to choose from both Parameters and Errors. When specific activity parameters are selected, the matching exceptions are automatically chosen as well.

Table 1. Business activity classification sample

Business Activity	Classification	Probable exceptions
Find Order(s)	Query data	Query failed (order not found)
Approve return manually	Enter data (“approved”); Update data	Update failed (authorization of CSR)
Schedule for future release	Enter data (release version); Update data	Update rejected (invalid release)
Notify customer	Send and Receive data; Produce data (reply)	Send failed (unknown contacts)
Issue release document to QA	Produce data (document); Send data	Send rejected (QA rejected)
Receive Report State of Accounts	Receive notification	Receive reject (invalid state)
Release Form Sign-off (Legal Signatories)	Verify data; Enter data (signature)	Verification rejected (refuse to sign)
Look for RMA number in received box	Perform action; Produce data (number)	Action failed (number not found)
Perform System Test	Perform complex action; Produce data (results)	Timeout
Identify discrepancy between requirements and product	Request data; Analyze data; Produce data	Timeout

3.3 Inferring Potential Errors from Declarative Specification of Business Requirements

Note for example, in Table 1, that the Update activity may throw an exception due lack of authorization. This type of failure means the underlying action never started — it was rejected before having the chance to start. Although, technically, one could encode the authentication requirement as part of the business process, it is more common to specify this as a declarative, non-functional requirement, which can be handled using an appropriate communication protocol at the BPEL level.

More generally, it is a common practice in business process execution to ensure various requirements on the data transfer protocols between Web Services:

Security, Authorization, Authentication, Policy, Trust, Privacy, Non-Repudiation, and more. Such non-functional business requirements can be defined at the BPEL level (using recent standard recommendations in the WS-* family), but not at the BPMN level. At the BPMN level, the analyst does not know whether the Web Service, that will match the requested Activity, supports a required protocol.

We recommend to add, at modeling time, these business requirements as additional business semantics that describe the functionality of the business activity at execution time. Although such protocols are part of the IT level infrastructure, the requirement to exploit such tools is a business consideration, and is therefore under the responsibility of the business analyst.

Such declarative non-functional requirements belong to the *Parameters* specification of our extended BPMN activities. We associate the following annotations to the Send and Receive classes of activities: Security, Authorization, Authentication, and Non-Repudiation.

For each such annotation, we specified inference rules that trigger potential error conditions. For instance, if I send data to a third party, then I might be told I am not authorized.

We further added execution settings that may be part of non-functional business requirements for an Activity: expected maximum duration, required resources, number of retries. Each such parameter can similarly trigger expected exceptions: for example, if the expected maximum duration of an activity is specified, then one should consider what happens if the activity times out.

3.4 Constructs

Activities in BPMs are now annotated by a rich set of semantic features: classification and requirements, on the one side — united under the “Parameters” attribute — and Errors on the other side.

To reduce the cognitive load required to handle the parameters and improve usability in the modeling tool, we introduce a new element in the palette of BPMN constructs. We define three *Constructs*, classified according to their relevant Parameters. Constructs correspond to different structural roles for activities within a BPM: activities internal to the business; reactive activities (receive event, process, reply) and activities relying on external services (send query / receive reply). Different parameters are relevant for each of these 3 types of activities.

Internal Activity (IA) The first construct type corresponds to an activity internal to the organization and must be executed by one of the organization’s (Web) services. Whether the activity is “analyze data”, “update data”, or a physical action (“perform action”), the activity has a primitive reason to fail.

Receive Process Reply (RPR) The second construct type is called *Receive Process Reply* (RPR). It corresponds to a business process, internal to the organization, that reacts to an external event. The process most likely includes a Start Event — that receives data — a sequence of Activities, and an End Event — that

returns a reply. It can fail due to any of its underlying Activities or due to missing/illegal data in the triggering event/data.

Send Receive (SR) The last construct type is called *Send Receive* (SR). Such a construct consists of invoking an Activity to be performed outside of the organization or, at the very least, outside of the department. We know, at design time, that the construct will send data and wait for a response.

Only a subset of the general Parameters and Errors are relevant for each type of construct. Table 2 shows the mapping between construct types and relevant parameters. This classification improves the usability of the modeling tool.

Table 2. Parameters and Errors, per Construct

Construct	Parameter	Error
RPR, IA	Perform action	→ Action failed
RPR	Complex action	→ Timeout
RPR	Query	→ Query failed
RPR	Update	→ Update rejected
RPR	Notification	
RPR	Analyze data	→ Analysis rejected
RPR	Verify	→ Verify rejected
RPR, SR	Send	→ Send failed
RPR, SR	Send	→ Send rejected
RPR, SR	Receive	→ Receive rejected: format
RPR, SR	Receive	→ Receive rejected: authentication
RPR, SR	Receive	→ Receive rejected: authorization
RPR, IA, SR	Duration	→ Timeout
IA	Resources	→ Unavailable Resource
IA	External Event	→ External Event
RPR, SR	Number of retries	
RPR, SR	Security	→ Non-compliance: Security
RPR, SR	Non-repudiation	→ Non-compliance: Non-repudiation
RPR, SR	A/synchronous	
RPR, IA	Commitment level	→ Commitment violation
RPR	Log update required	→ Receive rejected

4 Implementation

We implemented the proposed exception handling method as part of the Prosero business process modeling tool and semantic activities repository.

The business analyst populates the semantic data attributes using wizard forms. Figure 4 shows the main form of the wizard, relevant to the RPR Construct. The forms for the AI and SR Constructs are quite similar.

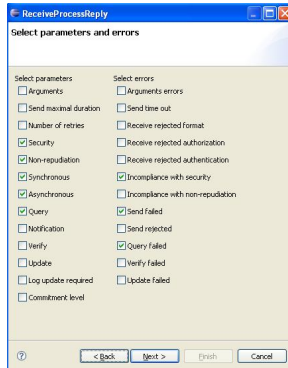


Fig. 4. Wizard form: Select parameters and errors.

The wizard is initiated when a Construct is added to the diagram. It connects to the Prosero repository and receives a complete list of Constructs and their associated Parameters, and Errors. According to the construct type, the wizard prompts for relevant parameters of the activity and infers the potential errors. The analyst can then override the relevant exceptions potentially triggered by this specific activity from the list of potential errors suggested by the wizard.

5 Empirical Evaluation

There is an expected trade-off in encouraging the specification of additional error handling in BPMs: on the one hand, more error handling will yield more robust specifications; on the other hand, it will require more time from the analyst, and yield potentially less readable diagrams.

We performed an empirical evaluation of the proposed method to verify the following hypothesis: we assume that the more complex the BPM, the more benefit there will be in using our error elicitation method.

We start by quantifying the notion of BPM complexity. We then present our experiment design and its results.

5.1 Business process complexity

How can we measure the complexity of a BPM? A thorough review [24] of different complexity metrics investigates the following parameters: the size of a model (activities and gateways), number of possible control flows, nesting depth, comprehensiveness (cognitive weight), anti-patterns, and modularization. The review concludes that “due to the number of factors that contribute to the complexity of a BPM, we cannot identify a single metric that measures all aspects of a model’s complexity” and proposes various metrics associations.

The “size of the model” metric is sufficient to identify business processes that will become challenging for a business analyst, including both Activities

and Gateways [25]. We use this variant of the metric to define three levels of complexity: Simple, Medium, and Complex, based on the average number of graphical elements in the diagrams.

Our assessment of 302 of business process workflows, from IBM [22], Defense Department [26], and Tibco [21], found the following “size of the model” measurements:

- Number of Activities: Max. 19, Avg. 5
- Number of Sub-Processes: Max. 9, Avg. 1
- Number of Gateways: Max. 9, Avg. 2
- Number of Lanes: Max. 3, Avg. 1

5.2 Experiment Design

We tested our enhanced modeling tool on a dataset of 11 business process models. These are detailed in Table 3. The business processes are based on Tibco [21], Smartdraw [27], and IBM [22] modeling tools’ samples, and are categorized into simple, medium, and complex complexity levels.

The complete list of the input business processes, their activities, and their output of exception handling choices can be found in [6].

Table 3. Experiment Input

Complexity level	Business process names
Simple	Deliver item, Return goods, Issue credit
Medium	Receive returned goods, Perform disposition, Administer return request
Complex	Update return, Approve return, HR Hiring, Software defect management, Software release product

In the experiment, a business analyst receives a business process diagram on paper, modeled using a regular modeling tool. He then remodels the BPD, using the tool enhanced with Constructs.

Both original BPD and the newly created BPD are compared with an expert opinion of the business process. The expert determines what the expected Error Events are that should be handled. Using the expert’s evaluation as a gold standard, we can measure whether the enhanced tool yields better results than using a regular tool, which produced the original diagram.

Comparing with the expert’s opinion produces True/False–Positive/Negative results for each error event and handler in the BPD. We compute the recall (REC), accuracy (ACC), precision (PRE), and F-Measure (F) of the original and of the outcomes, compared to the expert’s advice. The first and foremost important result is the accuracy of the specified Error Events — are there missing or redundant exception handlers.

We also measure the time it took the analyst to model the process - both using the enhanced tool and the original tool with no error handling.

Table 4. Results per process

Business process	Source	TP	TN	FP	FN	Time
<u>Simple business processes</u>						
Deliver item	original	2	6	0	3	08:16
	enhanced	4	5	1	1	09:14
Return goods	original	0	2	0	2	05:04
	enhanced	2	1	1	0	08:28
Issue credit	original	1	5	0	1	03:28
	enhanced	1	4	1	1	04:00
<u>Medium business processes</u>						
Receive returned goods	original	0	4	0	2	06:26
	enhanced	2	4	0	0	11:00
Perform disposition	original	0	1	0	2	05:20
	enhanced	0	1	0	2	10:43
Administer return request	original	1	3	0	0	06:10
	enhanced	1	2	1	0	08:00
<u>Complex business processes</u>						
Update return	original	1	3	1	2	16:06
	enhanced	1	2	2	2	17:30
Approve return	original	0	9	0	7	07:40
	enhanced	4	8	1	3	10:36
Software defect management	original	1	8	0	8	15:30
	enhanced	6	8	0	3	19:10
Software release product	original	1	4	0	1	15:40
	enhanced	1	3	1	1	19:50
HR Hiring	original	0	15	0	3	17:00
	enhanced	1	14	1	2	21:00

Table 5. Results per process category

Complexity level	Source	REC %	ACC %	PRE %	F %
Simple	original	16.67	55.6	33.3	22.2
	enhanced	50.0	58.3	38.9	43.8
Medium	original	50.00	84.1	66.67	57.14
	enhanced	83.3	82.14	66.7	74.1
Complex	original	16.89	64.5	50.0	25.3
	enhanced	54.1	73.1	50.0	52.0
Total	original	25.9	67.4	50.0	34.1
	enhanced	61.0	71.5	60.0	60.5

Table 6. Enhanced tool vs regular tool (original)

Complexity level	Actual results					
	REC	ACC	PRE	F-M	Time	Ease
Simple	≫	>	>	>	>	=
Medium	≫	<	=	>	>	≈
Complex	≫	>	=	>	>	≤

5.3 Results

Table 4 presents the entire outcome by business process and Table 5 by level of complexity. Finally, in Table 6, we compare the results of using the enhanced Prosero modeling tool with using the regular modeling tool in terms of error specification quality and modeling time.

Recall The resulting recall was outstanding — on average, 61% for the enhanced tool versus 26% for the regular. This means the tool’s suggestions help identify probable exceptions in the business process.

Accuracy The accuracy is very good for the Complex scenarios — 73% for the enhanced tool versus 64% for the regular, which confirms our hypothesis. While the accuracy was not as good as expected for the Medium case, 82% for the enhanced and 84% for the regular, it is not much lower, either. This may indicate that the business analyst should use the Constructs only for Complex business processes.

Precision The fact that the precision and F-measure are in favor of the enhanced tool, on average, is suspicious. We would have expected many more False-Positive cases, i.e., redundant exception handling, and the lack of them makes us suspect that the expert was too generous with his exceptions or that the business analyst too careful with his.

Diagram Complexity The added Exception Flows overload the diagram, making it less readable. This is likely to happen in high-level business processes that may receive accumulating Error Events. Furthermore, every exception handling at the BPMN level translates to another scope at the BPEL level, wrapping the Activity. That makes the BPEL diagram less coherent, as well, and is a burden on the BPEL engine, which executes the process. We believe this aspect must be addressed by more research.

6 Conclusions

Exception handling enforcement strategies have developed and matured in most programming languages. We explored the notion of transferring such enforcement strategies to the world of business processes. We also investigated how a semantic repository of activities can suggest likely exceptions at modeling time.

Our approach relies on inferring likely errors from semantic meta-data attached to business activities. We expanded BPMN’s activities descriptions with semantic meta-data. Semantic data on the activities — the characterization and run-time behavior of an Activity — is saved in a new meta-data attribute, Parameters. These parameters are then used to infer possible run-time exceptions, which are saved in the Errors attribute. The same semantic data is also mapped to declarative behavioral parameters, which can be enforced by orchestration

languages such as BPEL at runtime, by using standards such as the WS-* family.

Our next step was to prepare generic lists of Parameters and Errors. We divided these lists into three high-level categories which we called Constructs: Internal Activity (IA), Send Receive (SR), and Receive Process Reply (RPR), and associated relevant semantic parameters to each such Construct.

We implemented the method in Prosero's BPMN modeling tool, relying on the Prosero semantic repository of business activities. We evaluated the usability of the implementation.

Our enhancements to the tool have been tested by modeling 11 business processes of different complexity levels. Using the enhanced tool produced more accurate business processes, according to an expert's opinion. If we consider the expert's model to be robust, i.e., has lesser chance of failing at execution time; "covers all its bases", then the enhanced tool assists in producing robust business processes.

Even with contrasting feedback, regarding the usability of the current prototype, we see that semantic data has great potential in improving business process modeling techniques. Future implementations, which will employ heuristics, perspectives, and stronger static verification, will make the job of the business analyst even easier and will produce safer business processes.

References

1. OMG: Business process modeling notation (bpmn) specification v1.0. URL: <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf> (2006)
2. zur Muehlen, M., Recker, J.: How much language is enough? theoretical and practical use of the business process modeling notation. In Bellahsene, Z., Léonard, M., eds.: CAiSE. Volume 5074 of Lecture Notes in Computer Science., Springer (2008) 465–479
3. Silver, B.: Can business analysts model exception handling? BPMS Watch blogs. URL: <http://www.brsilver.com/wordpress/2006/09/08/can-business-analysts-model-exception-handling/> (2006)
4. Shang, Z., Cui, L., Wang, H.: A collaborative framework for exception handling in business process execution. In: 11th International Conference on Computer Supported Cooperative Work in Design, IEEE (2007) 914–919
5. Adams, M.J.: Facilitating Dynamic Flexibility and Exception Handling for Workflows. PhD thesis, Queensland University of Technology, Queensland, Australia (2007)
6. Ben-Eliahu, Z.: Exception handling enforcement in bpmn. M.Sc. thesis. Ben-Gurion University of the Negev, Beer-Sheva, Israel. (2008)
7. Elhadad, M., Balaban, M., Sturm, A.: Effective business process outsourcing: The prosero approach. International Journal of Interoperability in Business Information Systems **3**(1) (2008)
8. Silver, B.: Process modeling with bpmn: Six-part elearning series. SAP Community Network. URL: https://www.sdn.sap.com/irj/sdn/wiki?path=/x/2_E (2008)
9. White, S.A.: Introduction to bpmn. IBM Software group tutorial. URL: <http://www.bpmn.org/Documents/OMG%20BPMN%20Tutorial.pdf> (2005)

10. Blain, T.: BPMN Diagrams — Make It Right With Intermediate Compensation Events. Tyner Blain blog page. URL: <http://tynerblain.com/blog/2006/09/05/bpmn-intermediate-compensation/> (2006)
11. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.: Exception handling patterns in process-aware information systems. Technical report, BPMcenter.org (2006)
12. Christos, K., Costas, V., Panayiotis, G.: Enhancing bpel scenarios with dynamic relevance-based exception handling. Web Services, IEEE International Conference on **0** (2007) 751–758
13. Ghafoor, M.A., Yin, J., Dong, J., u Rehman, M.M.: π -rbt-calculus compensation and exception handling protocol. In: 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. (2006) 39–47
14. Bonatti, P.A., Festa, P.: On optimal service selection. In: WWW '05: Proceedings of the 14th international conference on World Wide Web, New York, NY, USA, ACM (2005) 530–538
15. Tessier, J.: Impact of exception handling patterns on web services. TheServerSide.com. URL: http://www.theserverside.com/patterns/thread.tss?thread_id=43752 (2007)
16. Bhumana, K., Kennedy, R., Pickett-Gordon, S.: Fault handling using bpel. Java.net project: BPEL BluePrint 3. URL: https://blueprints.dev.java.net/bpcatalog/ee5/soa/BP3_ProblemSolution.html (2006)
17. Fu, X., Bultan, T., Su, J.: Wsat: A tool for formal analysis of web services. In: Computer Aided Verification. Springer, Berlin (2004) 510–514 URL: <http://www.cs.ucsb.edu/~su/papers/2004/CAV2004.pdf>.
18. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in bpmn. Inf. Softw. Technol. **50**(12) (2008) 1281–1294
19. Zisk, S.: Business process semantics — an opportunity for convergence. DM Review Magazine. URL: http://www.dmreview.com/issues/2007_51/10001871-1.html (2008)
20. Wong, P.Y., Gibbons, J.: A process semantics for bpmn. Preprint, Oxford University Computing Laboratory, 2007. Available from: http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmn_extended.pdf. (2007)
21. TIBCO: Tibco iprocess suite. URL: <http://www.tibco.com/> (Accessed on 2008)
22. IBM: Websphere commerce version 6.0.0.6 documentation: business processes. URL: http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.business_process.doc/concepts/processBusiness_processes.htm (2006)
23. OASIS: Universal business language v2.0. URL: <http://docs.oasis-open.org/ubl/os-UBL-2.0/UBL-2.0.html> (2006)
24. Gruhn, V., Laue, R.: Approaches for business process model complexity metrics. Technologies for Business Information Systems (2007) 13–24
25. Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A discourse on complexity of process models. In: BPM 2006 Workshops, Workshop on Business Process Design BPI 2006, Berlin, Germany, Springer (2006) 117–128
26. U.S. Department of Defense, Business Transformation Agency: Ov-6c business process diagrams. URL: http://www.defenselink.mil/dbt/products/2008_BEA_ETP/bea/iwp/bealist_ov-6cbusinessprocessdiagrams_na.htm (Accessed on 2008)
27. Smartdraw.com: Smartdraw. BPMS Watch blogs. URL: <http://www.smartdraw.com/examples> (Accessed on 2008)