A Rigorous Methodology for Specification and Verification of Business Processes

Cristian Masalagiu, Wei-Ngan Chin,
Stefan Andrei, Vasile Alaiba

No. 4, October 2009
A Rigorous Methodology for Specification and Verification of Business Processes

Cristian MASALAGIU¹, Wei-Ngan CHIN², Ştefan ANDREI³ and Vasile ALAIBA¹

¹Faculty of Computer Science, "A.I.Cuza" University of Iași, Romania
²School of Computing, National University of Singapore, Singapore
³Computer Science Department, Lamar University, Beaumont, TX 77710, U.S.A.

Abstract. Both specification and verification of business processes are gaining more and more attention in the field. Most of the existing works in the last years are dealing with important, yet very specialized, issues. Among these, we can enumerate compensation constructs to cope with exceptions generated by long running business transactions, fully programmable fault and compensation handling mechanism, web service area, scope-based compensation and shared-labels for synchronization, and so on.

The main purpose of this paper is to present a semi-automatized framework to describe and analyse business processes. Business analysts can now use a simple specification language (e.g. BPMN [Obj06]) to describe any type of activity in a company, in a concurrent and modular fashion. The associated programs (e.g. BPDs [Obj06]) have to be executed in an appropriate language (e.g. BPEL4WS [ACD⁺03]). Much more, they have to be confirmed to be sound, via some prescribed (a priori) conditions.

We suggest how all the issues can be embedded in a unifying computer tool. We link our work with similar approaches and we justify our particular choices (besides BPMN and BPD): the TLA+ language for expressing the imposed behavioural conditions and Petri Nets ([EB87], [EB88]) to describe an intermediate semantics. In fact, we want to manage in an appropriate way the general relationship diagram (Figure 1). Examples and case studies are provided.

Keywords: business process, unifying computer tool, specification, verification

Correspondence and offprint requests to: Ştefan ANDREI, Computer Science Department, Lamar University, Beaumont, TX 77710, U.S.A., e-mail: Stefan.Andrei@lamar.edu
1. Introduction
1.1. Background

Business Process Management - BPM, is "gaining momentum" in the context of current economic growth. Although nobody denies the importance of BPM as a discipline, different organizations have chosen divergent approaches.

The lack of a consistent (provably sound) implementation for various business process models has already become a concern for computer scientists ([KIT02]). Under the auspices of BPMI/OMG, the Business Process Modeling Notation - BPMN, was developed and its latest standards issued on the market in 2006. Standard work is still in progress, for example BPMN 2.0 RFP was published in 2007 and the final version is expected to be released in 2008. In fact, “The primary goal of the BPMN effort was to provide a notation that is readily understandable by all...” businessmen ([Whi04]). In what follows we give a short description of BPMN, in fact of Business Process Diagrams - BPDs, which are, in fact, programs written in BPMN. Even if a BPD is a (multi-labeled, both on nodes and on arcs) directed graph with additional restrictions on the graphical layout, we tried for the beginning to avoid a more complicated abstract language.

Business analysts deserved to have and use a simple specification language to describe any type of activity in a company, in a concurrent and modular fashion. The programs in such a language have then to be executed and confirmed to be sound. The Temporal Logic of Actions - TLA, and the underlying specification language TLA+, together with the associated Temporal Logic model Checker - TLC, fits our purposes. We suggest how all the issues can be embedded in a unifying computer tool. We justify our particular choices (BPMN, BPD, TLA+, etc.) in Section 1.3.

1.2. Related Work

Ptolemy ([HLL+03]) and Kepler ([ABJ+04]) can be used instead of Business Process Execution Language for Web Services - BPEL4WS ([Bay04], [ACD+03], [Ohj06]) and some kind of an ontology-based semantics may be applied in view of integration instead of Petri Nets - PN.

For example, in the schemata in Figure 1 some nodes may be replaced with "equivalent" ones: BPEL4WS with Ptolemy/Kepler; Linear Time Temporal Logic - LTL and branching time temporal logic (Computational Tree Logic) - CTL ([CWA+96]) with any other type of logic which takes the time parameter into account (for example real-time logic, interval logic, duration calculus); TLA+/TLC ([Lam02]) with similar specification/model checker languages for temporal formulae, a new Symbolic Model Verifier - NuSMV ([CCG+02]). As mentioned in [CCG+02] NuSMV “...performs symbolic model checking of CTL formulae on networks of automata with shared variables”. As an alternative, SPIN, a model checker for LTL formulae written in Process Meta Language - Promela ([Hol04]), can be used. We can also add supplementary nodes (one for π-process, one for CCS, CSP, etc.) or arcs for expressing, as usual, existing or possible to be established links between different languages.

There exist many recent works for specification and verification of business processes. Business Process Execution Language (BPEL) is a de-facto standard for specifying the behavior of business processes. Many business-related features can be expressed in BPEL. For example, BPEL includes compensation constructs to cope with exceptions generated by long running business transactions. M. Butler, C. Ferreira and M.Y. Ng showed how a substantial subset of BPEL can be mapped to the so-called StAC language [BFN05]. The StAC language can be used to specify the orchestration of activities in long running business transactions.

Another important feature of BPEL is the fully programmable fault and compensation handling mechanism, which allows the user to specify the compensation behaviors of processes in application-specific manners. Z. Qiu, S. Wang, G. Pu and X. Zhao presented a formal operational semantics to a simplified version of BPEL, highlighting important concepts related to fault and compensation handling [QWPZ05].

It is known that formal methods are helpful for many issues raised in the Web service area. A. Ferrara presented a framework for the design and the verification of web services using process algebras and their tools [For04]. A two-way mapping between abstract specifications written using these calculi and executable Web services written in BPEL was described.

A formal framework for studying the semantics of orchestration languages for Web Services was introduced in [Vir04]. Virolia described the syntax and semantics of a core language to derive the interactive behaviour of a business process out from a BPEL4WS specification. This was realised by developing a process algebra...
meant to focus on the notion of correlation, which is exploited by BPEL4WS to define a business process as the concurrent behaviour of several process instances.

Creating Web services manually is a difficult task. K. Turner defined a description of graphically and formally describing web services [Tur05]. These description are automatically translated into Lotos, permitting rigorous analysis and automated validation.

The interaction between essential facets of web services can be described using process-algebraic notations. G. Salaün, L. Bordeaux and M. Schaerf claimed that process algebras provide a very complete and satisfactory assistance to the whole process of web service development [SBS04]. They showed on a case study that available tools based on process algebra are effective at verifying that web services conform their requirements and respect properties.

Web services are also increasingly being applied in solving many universal interoperability problems. BPEL contains also several other interesting features, including scope-based compensation, fault handling and shared-labels for synchronization. H. Jifeng, Z. Huibiao and P. Geguang explored an observation-oriented model for BPEL-like languages, which can be used to study program equivalence [JHG07].

All these works are handling important and very specialized issues regarding business processes. Our paper describes a semi-automatized framework for analysing such processes.

1.3. Motivation

Our choice was not arbitrary, although arguments for other possible variants also exist. BPEL4WS is known in the business world more than Kepler. On the other side TLA+ uses a visible notion of state, being action-based more than event-based.

For those implied in business, there exist internal (to the enterprise, department, office, etc.) activities to be performed (works) and the flow (of control) that define the order of performing the activities. There are also other external requirements including, e.g., the use of specific protocols or Information Technology (IT) tools. Any sequence of works may thus be called a (business) workflow process - BWP. Any enterprise (part of a business world) may be viewed as a collection of concurrent, inter-related (business) workflow processes.

Main concepts in the enterprise architecture world are, among others: standardization, integration, general protocols, IT-tools (such as web services [Pel03]). Any person acting in the business world, i.e., “...business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes” (businessmen, for short - [Whi04]) will be certainly interested in (at least) the following requirements:

Rec1 Easily express any BWP (structure, behavior, external events, etc.) in a reliable language (call it business model).

Rec2 All his/her necessities have to be semi-automatically accomplished, including certified answers to questions like: “Does this system (BWP) do what he/she wants?”

Rec3 Use the (full) power of new IT-technologies in an integrated environment. To attend this goal we will build upon the relations described by the diagram in Figure 1. We shall call it the General Relationship Diagram - GRD.

If the businessmen are to use the full power of new IT-technologies they also have to be acquainted to some abstract concepts and methods. But, what happens if they do not want to have anything in common with

<table>
<thead>
<tr>
<th>BWP</th>
<th>← →</th>
<th>WFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD</td>
<td>← →</td>
<td>BPEL4WS</td>
</tr>
<tr>
<td>LTL/CTL</td>
<td>← →</td>
<td>automata</td>
</tr>
<tr>
<td>TLA</td>
<td>← →</td>
<td>TLA+/TLC</td>
</tr>
</tbody>
</table>

Fig. 1. General Relationship Diagram
“boring” abstract stuff? Then, the idea is to semi-automatically construct a formula \( SP \) describing a business process diagram (\textit{specification}) together with a formula \( SA \), describing the supplementary conditions to be satisfied by the \textit{BWP} (the \textit{safety assertions}). Then, our model checker will have to verify the “consistency” of the formula \( \Psi \triangleq SP \rightarrow SA \) for the given \textit{BWP}.

1.4. One motivating example

\textbf{Example 1 (Request for Quotation - RFQ).} We have chosen the below characteristics for the well known \textit{RFQ} problem ([KTK02]). The \textit{RFQ} process starts when a purchasing agent sends out a request for quotation to a set of selected vendors. After having received a quote from each vendor, or after a dead-line has elapsed, an evaluation process to select the best quote begins. The business process is composed out of the following activities:

- **Send** notification activity models the repetitive sending of the request for quotation from the purchasing agent to a preselected set of vendors.
- **Receive** quotes activity occurs when the purchasing agent receives a quote from a vendor.
- **Decide** disposition activity compares the various quotes with each other. It takes place after either a quote from each vendor has been received or a \textit{deadline has elapsed}. The corresponding activity takes place only if at least one quote has been received.

The two loosely coupled repetitive activities are, of course, \textit{Send} and \textit{Receive}. \textit{Send} is executed for each of the vendors the organization wants to involve in the \textit{RFQ} process. The \textit{Receive} activity, although coupled with \textit{Send}, is also repetitive but it may not occur the same number of times. Indeed, a vendor may choose not to respond in any way to the \textit{RFQ} document, thus a \textit{Send} is not always followed by a \textit{Receive}.

Many (if not all) business processes are specified together with real-time constraints. In our example, each \textit{Send} activity is either followed by a \textit{Receive} or meets a deadline. The time limit is usually imposed by a real time constraint, e.g., 48 hours after sending of the \textit{RFQ}.

Let \( p, q, d, t, r \) be atomic propositions, with the following meaning:

- \( p \)- send
- \( q \)- receive
- \( d \)- decide
- \( r \)- timeout received
- \( t \)- process finished

The characteristics describing our example may be expressed by the following \texttt{CTL} [CE82] formulae:

\begin{align*}
F & = F_1 \land F_2 \land F_3 \land F_4 \land F_5 \\
F_1 & = \mathit{AX} p \\
F_2 & = \mathit{AX}F_1 \rightarrow p \\
F_3 & = \mathit{AX}F_2 \rightarrow p \lor q \\
F_4 & = \mathit{AX}q \rightarrow q \lor d \\
F_5 & = \mathit{AX}(d \lor r) \leftrightarrow t
\end{align*}

We may notice that \texttt{CTL} can not model adequately the time constraints imposed by the above example. We consider that we need \texttt{TLA} in order to better express these real-time constraints. Verification of the \texttt{CTL} definition is possible using \texttt{NuSMV}, which imposes some restrictions on the \texttt{CTL} model. Instead, the \texttt{TLC} model checker does not limit the \texttt{TLA+} specifications.

\begin{itemize}
\item In the above example the formula \( SP \rightarrow SA \) (in the general view) is of the form \( \mathit{AGF} \rightarrow \mathit{AFt} \) (it represents \textit{termination}) and its validity can be easily checked. As we have previously mentioned, we have used some commonly used notations for \texttt{CTL} operators.
\item The \texttt{BPMN} specification of our problem is defined by a business process diagram (\textit{BPD}). \texttt{BPMN} is offering us the possibility to map the problem definition accurately: the \textit{BPD} has a \textit{Starting point}, three \texttt{activities} and an \textit{End point}. Both \textit{Send} and \textit{Receive} can execute several times and this behaviour is depicted on the diagram in \texttt{Figure 2}. The \textit{Decide} activity is conditioned by either the receiving of the same number of messages or by the deadline. The time condition is depicted by a timer set to trigger at the deadline.
\end{itemize}
The above example is relevant and suggests the importance of professional responses to questions regarding the whole set of the behaviours of the specified systems.

To summarize, the main contributions of our paper may be found in Section 3, Sections 5 and 7. We present a semi-automatic framework to describe and analyse the BWPs. The associated programs (the BPDs) have to be executed in an appropriate language and they have to be confirmed to be sound via some prescribed (a priori) conditions. We gave some suggestions how all the issues can be embedded into some unifying computer tools. We linked our work with similar approaches and we justified our particular choices, such as the TLA+ language for expressing the imposed behavior or conditions and the PNs to describe the intermediate semantics.

Structure of the paper. Section 2 is dedicated to a detailed and understandable presentation of the specification language BPMN and of its program (the BPDs) for the description of BWPs. Their semi-automatic translation to an executable (web services oriented) language (BPEL4WS) is presented in [Gro04] and [Obj06]. Section 3 really describes our view for verifying BWPs. In Section 4 we turn to the idea that these unformalized and semi-automatic treatments may have harmful impacts on the soundness of the implied real systems. We suggest a semi-automatically way to treat the global problem, based on some operational semantics (the PN approach may be essential). The examples provided may have important generalizations. Sections 5, 7 and 8 contain our most concrete work and results, together with the main ideas regarding future work.
2. Preliminaries

Any BPD is composed by elements in four basic groups (in the rest of this section we will rely mainly on [Whi04], including the figures):

- **Flow Objects** - FO (generally the activities/works, nodes in graph)
- **Connecting Objects** - CO (generally the usual flow of control, that is arcs in graph, which relate the FOs)
- **Swimlane Objects** - SO (determine some partitions of the intended graph into certain subgraphs and represent supplementary conditions for drawing it)
- **Artifact Objects** - AO (markings, i.e. special labels of the nodes and arcs in the graph).

The **flow objects**, depicted in Figure 3(a), are also divided into three distinct subcategories:

- **Flow Objects Events** - FOE (graphically represented by circles);
- **Flow Objects Activities** - FOA (rounded-corner rectangles);
- **Flow Objects Gateways** - FOG (diamonds).

The **COs** are of three types, all being represented by straight arrows in Figure 3(b):

- **Connecting Objects of type Sequence Flow** - COSF;
- **Connecting Objects of type Message Flow** - COMF;
- **Connecting Objects of type Association** - COA.

CO may also have additional marks (on, under or above the arrow).

**Example 2 (a simple workflow process).** It is the moment to display another BPD (apart of Figure 2), describing a simple BWP: the selling of a package to a client by a certain vendor (Figure 4). It contains not only the already given FO and CO but also some AO (more precisely, annotations which will be soon described, but are self-explanatory in the example). There exist an initial and a final event, some atomic activities and one (decision) gateway.

The **SO** (graphically depicted in Figure 5(a)) represent “a mechanism to organize activities into separate visual categories” in order to “illustrate different functional capabilities or responsibilities” in a process (similar as in the so called traditional swimlane process modeling methodologies). They can be of two kinds:

- **Swimlanes Objects of type Pool** - SOP;
- **Swimlanes Objects of type Lane** - SOL.

The situation captured with **SOs** is business to business - B2B.

The **AO** shapes are depicted in Figure 5(b).
Let us end the syntactical description of a BPD with two examples which reflect the use of some objects not yet emphasized here. First we show in Figure 6 a relevant example using pools, even if it is not, strictly speaking, part of the business world. By putting almost all concepts together, we present how a segment of a process using lanes looks like in Figure 7.

The examples describe some basic but not so complex real systems. Figure 6 and Figure 7 are self-explanatory.

3. Our view on verifying BWPs

We note that BPMN, a combined data and control flow language, may be viewed also as a graphical imperative, concurrent language and any BPD as a program in it. The analogy with usual programming languages may thus be expressed as

\[ \text{BWP} = \text{flowchart (or pseudocode)} \]
\[ \text{BPD} = \text{source program} \]

Up to now it is also known that any BPD can be semi-automatically transformed into an executable program written in BPEL4WS ([Obj06]). Thus, the analogy with usual programming ends with:

\[ \text{BPEL4WS} = \text{executable code} \]

The generation of an executable BPEL4WS from a BPD may be assimilated with an interactive compilation (interpretation) procedure, executed repeatedly in some distinct steps. BPEL4WS was created in a joint venture by IBM, Microsoft, e. a. and has finally emerged in 2003. The idea in BPEL4WS is to use
the power of world wide web i.e., following the paradigm of web services the companies are allowed to describe (simultaneously) the external environment of their processes (including the electronic or the abstract one).

**BPEL4WS** is the BPMN creator’s choice for an execution language ([Whi04]). The standard, based on XML technologies, provides a good orchestration platform for the miscellaneous processes a **BPD** may use. Although human readable and resembling with common concurrent (or distributed) programming languages, **BPEL4WS** introduces a new complexity level between the business process designer and the IT specialist (programmer, analyst, etc).

The automated translation of business process diagrams to a lower level, standards compliant specification language becomes in this context of paramount importance. The benefits of this translation are obvious:
closing the gap between the client (business specialist) and the provider (IT specialist) leads to creation of competitive advantages like better system integration, conformance to specifications and better adaptability. Unfortunately BPDs do not benefit now of a formal syntactic structural definition (see Section 5).

Any attempt to provide a well defined transformation algorithm is blocked by the lack of formalism in the definition of the diagram structure. Consequently, all we can do for the moment is to provide some kind of a mapping based on a tabular correspondence. As a consequence, BPMI/OMG suggests the opportunity of several mappings to other languages, for example BPEL4WS, ebXML, RosettaNet or W3C Choreography. The mapping from BPD to BPEL4WS, described in Chapter 6 of the BPMN specification ([Obj06]), is the only one available at the moment from BPMI/OMG. This mapping offers a human analyst the necessary information to translate a diagram but it is not enough to provide, by itself, a semi-automated (and sound) translation.

The automation of this process is even more difficult considering that a BPD does not embed always all the information needed to create an executable process. The BPDs can fall in at least three categories, corresponding to the three submodels of the BPMN standard specifications:

- **private (internal)** BWPs;
- **abstract (public)** BWPs;
- **collaboration (global)** BWPs.

Each of the above has its particularities. For example, abstract processes interact, by their nature, with external (relating to the enterprise) processes. While this kind of diagrams is not more complicated (in structure) as the private type, it needs to rely for its execution on external services, usually available through web services. Since the graphical representation is, for any non-trivial process, not sufficient to generate an executable program, the diagram must be decorated with additional information. Attributes are used in this respect to place information on each element of a BPD.

For example, any graphical object must be assigned at least an Id attribute and optionally Categories and Documentation. External and internal processes may be invoked via XML-based messaging using the Web Services Description Language - WSDL. Even a BPEL4WS program sometimes may not be directly executable (it may contain some abstract parts, e.g. referring to some unknown business protocols), it will eventually be so under WSDL.

Executable processes are actually process specifications built on top of existing web services and can be executed on an engine like Business Process Web Services for Java - BPWS4J ([CK04]). More about other business models and the BPMI/OMG complete stack of standards can be found at the web addresses listed at the end of the paper and in [vdA03], [Ark02], [Obj06], [RO03].

It is important to note that the third requirement listed at the beginning of the paper, regarding the use of most valuable IT tools, is also fulfilled. We cannot say the same about the second one. The problem is that we act in a very complex concurrent (programming) environment and even very simple patterns occurring in the business world may become "...a real challenge to modern workflow management systems" ([KTK02]).

In order to do a global study in advance for BWP, some appropriate specification abstract model is needed. These formal models are used to control the fact that a given process has properties such as reachability (a particular situation has indeed the chance to occur), liveness (under certain conditions, a particular situation will eventually occur) or termination (which is in fact a particular kind of liveness).

Any IT professional knows that the qualified use of these models usually requests a solid mathematical background and special computing skills. Thus, we have a contradiction between the desire to posses a simple, user-friendly unsophisticated tool for specifying a business system and the necessity to ensure that the specification is sound and safety confident.

There exists the possibility to integrate business models (such as BPEL4WS) and model checkers on-line (for the last topic [YML99], [CWA+96] or [GL94] may be consulted). There exists, for example, a huge work done on scientific workflows and the concrete results already obtained are impressive. We do not entirely agree with the somewhat discouraging opinion that "business workflows are the remote relatives (2nd - 3rd cousins) of scientific workflows" or "in business workflows data-flow and control-flow are often divorced while in scientific workflows they are often married".

The problem is that the real-time execution (viewed as an operational semantics) of a process will never reach (with the exception of some simple situations) the needed level of safety as any formal semantics can provide. This is due mainly to the inherently strong parallelism and non-determinism implied by process evolution and we may simply have much too many (even an infinite number of) behaviors. Any real-time
machine can provide only a small part of the whole set and it is possible not to have time to identify the “sick” ones. That is why a formal semantics is needed.

4. Semantics for BPMN

If our supplementary concern is not to radically change BPMN, then we have not an easy task, taken into account the fact that for the BPD-class we do not have a constructive (syntactical) definition. Much more, a lot of semantics is involved explicitly in the shape of a BPD and in the sets of its supplementary markings.

One approach to semantics is indirect and based on an “embedding” of BPDs into an existing language, the main candidate being the class of Petri Nets (\(\pi\)-calculus, CCS, CSP, etc.). They are somewhat similar in shape, including the graphical presentation and the lack of a structural definition. The workflow Petri Nets were already introduced and studied, important results regarding some of their properties have been obtained, etc. Although it may look tedious, the advantages of this approach are apparent: it will force a discipline in BPMN-programming and will allow getting further general results if we use some known Curry-Howard ([Tho91]) links between different formal models of concurrency (e.g., some special functors between different categories).

Let us focus on the direct semantic approach. This means to be able to give directly a formal (operational, algebraic, etc.) semantics:

\[
\vdash \colon \text{BPD} \rightarrow \mathcal{L}
\]

In (1), BPD represents the class of all diagrams and \(\mathcal{L}\) is an appropriate class of (infinite) trees (or paths). \(\mathcal{L}\) may represent, in the meantime, the set of trees (paths) in which a certain verification formula \(\Psi\) (in fact only its SP-part) is true.

As we have already mentioned, if we express the BWP directly in, say, TLA - temporal logic of actions (implemented as TLA+ and having a specific incorporated model checker - TLC [Lam94], [YML99], [Lam02]), \(\Psi\) may be written as

\[
\Psi \triangleq \text{SP (specification)} \rightarrow \text{SA (safety assertion)}
\]

The validity of \(\Psi\) can be shown, as many theorem provers do, by proving that \(\neg \Psi\) is unsatisfiable ([AC04]). Thus, we may refer to the schemata we depicted in Figure 1.

5. From languages to formulae

The translation of a BPD into a TLA+ formula can be made semi-automatically with the help of a Petri net, as an intermediate step or even directly. The idea behind is based on the fact that we can provide indeed a constructive definition of a BPD and the most reasonable (constructive) semantics for this kind of graphs are the Workflow Petri Nets - WPN.

The proofs of the following theorems are in fact sketches. Due to the ambiguities appearing in the definition of a BPD (see, for example, the concept of an artifact object), some suggested translations were skipped. Special computer tools should be developed in order to alleviate the processes of semi-automatic translation.

**Theorem 1.** Any BPD can be semi-automatically translated into a WPN.

**Proof.** A syntactical direct translation may be given if we note that any BPD is in fact a multi-labeled directed graph. The labels embed the semantical view of the designer. We start with a structural definition of the class of BPDs and after that we may give a corresponding list of associated WPNs, semi-automatically constructed. The base of the structural definition has to contain the BPDs in Figure 8. In this figure the rectangles represent FOA and/or FOE.

The diagrams above may be easily transformed into some graphs having exactly one **Start point** and one **End point**. Considering the **structural step** we have only to combine the initial elements. This can be done in a simple way by “putting together” the (uniques) **End point** and **Start point** of the two graphs involved. Then, in the resulting graph, this new node has to be cancelled. The additional semantic
information (represented by Swimlane Objects and Artifact Objects) may be captured in the corresponding WPN.

However to achieve a complete automatized translation, a more complicated work has to be accomplished. Some restrictions have to be imposed on the initial BPDs and more sophisticated Petri Nets have to be used (e.g. Colored Petri Nets or Jumping Petri Nets [TJM91], [TT02]). Much more, the categorial approach (illustrating the Curry-Howard principle) is necessary for introducing a formal semantics for the class of BPDs. This semantics must then related to the process-based formal semantics for Petri Nets ([Sas94]).

Think now that we have a WPN which is an image of a corresponding BPD. The suited behavior of this WPN can be satisfactorily expressed by a temporal formula. The next theorem states that such a formula may be again satisfactorily expressed by a TLA+ program.

**Theorem 2.** Any LTL/CTL formula can be semi-automatically implemented into a TLA+ program.

**Proof.** The TLA+/TLA language is closer to LTL than to CTL taking into account their associated semantics. Again, a list of suitable correspondences has to be used. The things are simpler because all the implied languages already have formal syntactical definitions. It is possible that some extra constraints be added to achieve a “clean” translation. □

The proof of the below theorem is immediate according to [YML99]. Some restrictions on the implied objects also exist.

**Theorem 3.** Any TLA+ program can be verified using TLC (on the basis of model checking strategy).

The previous three theorems were needed to support the correctness of the translations BWP → WPN, LTL/CTL → TLA, and TLA → TLC. In this way, our GRD (Figure 1) is strongly motivated. For instance, the application of **Theorem 1** and **Theorem 2** is exemplified in the next section.
Note that we may try to directly transform a BPD into a TLA (TLA+) formula (program). Although this translation seems very natural it can generate some complex practical issues. For example, the final TLA formula (TLA+ program) is complex. That is why we choose to look to different ways of traversing the GRD in order to get an efficient result. Generally, the business community is not concentrating on the effort to avoid this problem. We preferred to look to direct implementations, such as jBoss jBPM [http://www.jboss.com/products/jbpm]. Directly using a specification language called JPDL a BPEL4WS program can be obtained. From a BPEL4WS program, an execution graph (useful for the implementation of a model checking method) can be semi-automatically generated ([Law87]). The translation from JPDL to BPEL4WS is done using a one-to-one mapping method that is easy to implement, as the former language has a much simpler syntax than BPMN.

6. A new incremental approach for theorem debugging

Theorem proving is a challenging task for formal verification of systems. To prove that a formula \( \phi \) is a theorem, a common technique [CL97] is to show that its negation (that is, \( \neg \phi \)) is unsatisfiable. Most of the techniques reduce to show that a corresponding propositional formula \( F \) is unsatisfiable. If \( F \) is unsatisfiable, then the formula \( \phi \) is a theorem.

The theorem debugging problem deals with the case when \( \neg \phi \) is satisfiable. The re-design and debugging approaches refer to do some allowed changes (additions or removals) in \( \neg \phi \) such that the new formula \( \neg \phi' \) becomes unsatisfiable.

Before describing our technique to solve the theorem debugging, we highlight some of the existing techniques to solve the theorem proving. There exist many efforts to efficiently solve the theorem proving problem, based for example on rewriting rules and/or SAT-based techniques. Since 50’s, mathematicians started implementing theorem provers by constructing truth tables for statements in propositional logic. There is still a competition between these known methods as well as a need for new efficient techniques. Given a particular class of systems, a method may work better than others. For example, considering systems expressed in modal logics, it seems that SAT-based decision procedures are more efficient than decision procedures based on translation methods [GGST00].

Motivation: The motivation of this section is to describe a new technique based on incremental counting SAT, as an alternative for the SAT-based technique. We show that our technique may outperform the SAT-based approaches, when considering the re-design and debugging of systems. The experiments from [AC04] show that this is a promising technique as an alternative to SAT approaches.

Related Work: The SAT problem (Given a propositional formula \( F \), is there a truth assignment for \( F \)?) was the first discovered \( \mathcal{NP} \)-complete problem [Coo71]. We propose an alternative of SAT-based techniques by using instead a counting SAT-based technique (denoted also \#SAT). A SAT solver tests if a propositional formula \( F \) has at least one truth assignment, while a \#SAT solver returns the number of truth assignments of \( F \). For efficiency reasons, many of the existing SAT-based techniques are applied incrementally, that is, using the satisfiability of some sub-formulas to determine the satisfiability of a given formula. Given a boolean formula \( F \), an algorithm using the satisfiability of some sub-formulas of \( F \) to determine the satisfiability of \( F \), is said to be incremental. Such incremental algorithms are efficient because when checking the satisfiability of \( F \), only the provided final results for the sub-formulas of \( F \) are used, and not the re-computation of the whole \( F \). The basic incremental satisfiability problem of propositional logic has been introduced in [Hoo93] as follows: \textit{“Given a propositional formula \( F \), check whether \( F \cup \{ C \} \) is satisfiable for a given clause \( C \)”}. The algorithm presented in [Hoo93] solves the SAT problem using the Davis-Lemmon-Loveland’s procedure [DLL62] combined with a backtracking strategy that adds one clause at a time. For example, a SAT solver able to handle non-conjunctive normal form constraints and incremental satisfiability was presented in [WKS01].

As stated in [GJ90], the counting (enumeration) problems are another type of interesting problems, but they might be intractable even if \( \mathcal{P} = \mathcal{NP} \). A counting problem \( P \) determines how many solutions exist, not just an answer “Yes/No” like a decision problem. The same concept of completeness can be defined for counting problems. A counting problem \( P \) is in \#\( \mathcal{P} \) if there is a non-deterministic algorithm such that for each instance \( I \) the number of guesses that lead to the acceptance of \( I \) is exactly the number of distinct solutions of \( P \) regarding \( I \) and such that the length of the longest accepting computation is bounded by a polynomial in the length of \( I \) [Val79]. The \#\( \mathcal{P} \)-complete problems are at least as hard as \( \mathcal{NP} \)-complete problems, but probably much harder. The problem of counting the number of truth assignments (denoted by \#SAT) is: Given a propositional formula \( F \), how many truth assignments exist for \( F \)\. It was proved to be
Obviously, an algorithm for solving #SAT problem can also solve the SAT problem. There exist already algorithms for solving the #SAT problem [Iwa89, Dub91, And95, Zha96, And04]. In this paper we are addressing to a more challenging problem, namely the incremental #SAT problem: “Knowing the number of truth assignments of $F$, what is the number of truth assignments of $F \cup \{C\}$, for any arbitrary clause $C$?” To be best of our knowledge, there is no similar work dealing with this particular problem. Because our algorithm deals with the incremental side of the problem, this is definitely more efficient than existing algorithms, so there is no need to count again the number of truth assignments for the whole $F$. Our investigation was occasioned by efforts to solve timing constraints verification for real-time systems [JM86, JM87, AC04, ACCL06], but it has application whenever one wishes to check again for logical inferences after enlarging a propositional knowledge base. In fact, the technique described in this paper is useful not only for verification and debugging of real-time systems but for any kind of systems in general.

Concepts and Notations: In the following we introduce some concepts and notations [And95, And04] to allow the text to be self-contained. Let $\mathbb{L}_P$ be the propositional logic over the finite set of atomic formulae (known also as propositional variables) denoted by $V = \{A_1, A_2, ..., A_n\}$. A literal $L$ can be an atomic formula $A$ (positive literal), or its negation $\neg A$ (negative literal). We put $oL = \neg A$ if $L = A$ and $oL = A$ if $L = \neg A$ and we denote $V(L) = V(oL) = A$. Any function $S : V \rightarrow \{0,1\}$ is an assignment and it can be uniquely extended in $\mathbb{L}_P$ to $F$ (this extension will be denoted also by $S$). A formula $F$ is called satisfiable iff there exists a structure $S$ for which $S(F) = 1$. A formula $F$ is called unsatisfiable (or contradiction) iff $F$ is not satisfiable. Any propositional formulae $F \in \mathbb{L}_P$ can be translated into the conjunctive normal form (CNF): $F = (L_{1,1} \lor ... \lor L_{1,n_1}) \land ... \land (L_{k,1} \lor ... \lor L_{k,n_k})$, where $L_{i,j}$ are literals. In this paper, we shall use a set representation $F = \{\{L_{1,1}, ..., L_{1,n_1}\}, ..., \{L_{k,1}, ..., L_{k,n_k}\}\}$ to denote CNF. Any finite disjunction of literals is a clause. A formula in CNF (finite set of clauses) is called a clausal formula. Given $F = \{C_1, ..., C_s\}$, where $s \geq 1$ an arbitrary clausal formula over $V$, we denote by $\text{det}_V(F)$ or $\text{det}_V(C_1, ..., C_s)$ the determinant of $F$. This represents the number of truth assignments of $F$. Given $C$ is an arbitrary clause over $V$, then $\text{inc}_V(C, F)$ is called the increment of $F$ with clause $C$. This represents the number of truth assignments supposed to be subtracted from the determinant of $F$ when adding clause $C$.

Comparison between incremental counting SAT and SAT problem: Since an incremental SAT solver finishes its execution after detecting the first truth assignment, it is expected that an incremental SAT solver to be faster than an incremental counting SAT solver. In contrast, an incremental counting SAT solver will count how many truth assignments exist for a given clausal formula. However, there exist large classes of problems where an incremental counting SAT solver outperforms a SAT solver. Re-design and debugging problems are such examples. A successful example is a re-design approach for timing constraints verification of real-time systems [AC04].

The Main Idea: As we mentioned early, to prove that a formula $\phi$ is a theorem, a common technique is to show that its negation (that is, $\neg \phi$) is unsatisfiable. Most of the techniques reduce to show that a corresponding propositional formula $F$ is unsatisfiable. If $F$ is satisfiable, then we need to do some changes (add or delete some subformulas). For simplicity, suppose we need to do only additions of clauses (removal of clauses can be done similarly, [AC04]). We assume that we have to choose one clause out of all possible clauses $C_{1,1}, ..., C_{1,k}$, which is nothing else but the set of allowed clauses to be added according to the system specification (Figure 11). In other words, the goal of this approach is to provide a (minimal) set of clauses $C_1, ...., C_l$ such that $F \cup \{C_1\} \cup ... \cup \{C_l\}$ is unsatisfiable. Note that $F \cup \{C_1\} \cup ... \cup \{C_l\}$ is also called a solution. The solution will correspond to a new formula, say $\neg \varphi'$, which implies that $\varphi'$ is a theorem for the new system specification. We may say that $\varphi'$ is a correction of $\varphi$ after some allowed changes.

![Figure 11. Counting SAT versus SAT](image-url)
Proof Sketch: A typical incremental SAT solver can only check whether formulas $F \cup C_{1,j}$, ..., $F \cup C_{1,k}$ are satisfiable or not. But it cannot predict which of these $k$ clauses lead to the solution. Instead, an (incremental) counting SAT solver is able to precisely decide which of the formulas $F \cup C_{1,j}$, ..., $F \cup C_{1,k}$ lead faster to a solution. This can be done efficiently by considering $j$ from $\{1, ..., k\}$ such that $\text{inc}_V(C_{1,j}, F)$ is the maximum among all increments $\text{inc}_V(C_{1,1}, F)$, ..., $\text{inc}_V(C_{1,k}, F)$. In other words, the sub-tree of root $C_{1,j}$ will be selected for the next iterations (the one drawn with ticker lines, Figure 11). We get that $C_{1,j}$ is the best candidate which maximize $\text{det}_V(F \cup C_{1,1})$, ..., $\text{det}_V(F \cup C_{1,k})$. That is, $\text{det}_V(F \cup C_{1,j}) = \text{det}_V(F) + \text{inc}_V(C_{1,j}, F)$ is the maximum possible determinant. So, $C_{1,j}$ represents the optimal solution for $C_{1,j}$. The iteration can continue similarly with $C_{2,j}$, ..., $C_{l,j}$. This procedure continues until the determinant becomes zero, which corresponds to the unsatisfiability of the formula. In conclusion, a SAT solver needs to visit an exponential number of nodes in the worst case (that is, the total number of nodes of the tree in Figure 11), while our #SAT solver needs to visit only a linear number of nodes in the worst case. The above technique has a finite number of iterations according to the monotony of the determinant and the increment. At any iteration, the determinant of $F$ is decreasing because the increments of value zero are not taken into consideration. Moreover, the maximum increment is chosen at any iteration.

Conclusion of the incremental counting SAT: Another advantage is the re-use of the old value of the determinant to the next iteration. The former computations are not repeated. The incremental approach is much faster than the non-incremental approach. A comparison between these approaches was presented in [AC04]. Due to the extensive computation of this approach, we do not provide an example in this section, but we refer instead the interested reader to [AC04].

This technique showed how that incremental counting SAT approach is a better alternative to an usual SAT solver when considering the challenging problems of re-design or debugging of systems. Previous works [AC04, ACCL06] have demonstrated that this approach is indeed a promising technique. A #SAT solver can identify deterministically the solution, while a SAT solver needs to compute non-deterministically all the possible candidates for the solution (Figure 11).

7. Model checking

This section begins with the direct translation of the RFQ business process (see Example 1) into a TLA+ program (representing $\Psi$, i.e., $SP$ and $SA$).

The TLA+ specification of the RFQ problem follows the same pattern as its translation into a BPD form, but in a more rigorous fashion. The set of valid initial states is specified by the $\text{Init}$ state predicate. A transition is specified by the $\text{Next}$ action, that can be either a $\text{Send}$, a $\text{Receive}$ or a $\text{Decide}$ step depending on their respective guard conditions.

-------------------- MODULE RFQ ------------------------
EXTENDS RealTime, FiniteSets

CONSTANTS VENDORS, TIMEOUT, EVAL(_)
VARIABLES notifiedVendors, offers, startTime, winner

ASSUME IsFiniteSet(VENDORS)

Init ==
\ 
/\ notifiedVendors = {}
/\ offers = {}
/\ winner = {}
/\ startTime = now

----------------------------- MODULE RFQ --------------------
EXTENDS RealTime, FiniteSets

CONSTANTS VENDORS, TIMEOUT, EVAL(_)
VARIABLES notifiedVendors, offers, startTime, winner

ASSUME IsFiniteSet(VENDORS)

Init ==
\ 
/\ notifiedVendors = {}
/\ offers = {}
/\ winner = {}
/\ startTime = now

Send(V) ==
/\ V \in VENDORS
/\ V \notin notifiedVendors
/\ notifiedVendors’ = notifiedVendors \cup V

----------------------------- MODULE RFQ --------------------
Receive(V, O) ==
\[ V \in \text{notifiedVendors} \]
\[ \text{offers'} \equiv \text{offers} \cup \langle \langle O, V \rangle \rangle \]

TimeOutCondition ==
\[ \text{now} - \text{startTime} > \text{TIMEOUT} \]

SelectWinner ==
\[ \text{winner'} = \text{CHOOSE vendor} \in \text{notifiedVendors} : \]
\[ (\forall x \in \text{notifiedVendors} : \text{EVAL}(\text{vendor}) > \text{EVAL}(x)) \]

Decide ==
\[ \neg \ \text{TimeOutCondition} \]
\[ \neg \ \text{Cardinality(notifiedVendors) = Cardinality(VENDORS)} \]
\[ \neg \ \text{Cardinality(offers) = Cardinality(notifiedVendors)} \]
\[ \text{SelectWinner} \]

Next ==
\[ \forall V \in \text{VENDORS} : \text{Send}(V) \]
\[ \forall V \in \text{notifiedVendors} : \text{Receive}(V, O) \]
\[ \text{Decide} \]

Spec ==
\[ \text{Spec} \]
\[ \text{Init} \]
\[ \text{TNow}(<\langle \text{notifiedVendors}, \text{offers}, \text{startTime}, \text{winner} \rangle>) \]
\[ \text{[]}[\text{Next}].<\langle \text{notifiedVendors}, \text{offers}, \text{startTime}, \text{winner} \rangle> \]

The above program can be used to verify, using the TLC model checker, that the formula \( \Psi \) is true, where \( SP \) is represented by the formula \( \text{Spec} \) and \( SA \) is any safety assertion, such as \( \Box(\text{ENABLED Next}) \) (which describes the absence of deadlock).

8. Conclusions and future research

There is much to do in the future to achieve the proposed goal: the creation of commercial integrated, automated tools for specification and verification of business processes. The steps to be followed were suggested in the paper. Currently, we are able to do the translation steps in a systematic way (by combining tools with manual work), but we expect to develop a complete automatic implementation platform in the future.

An incremental approach may be used as an efficient way to avoid the recomputation of former parts of specification ([AC04]). We have to find efficient ways to translate BPDs into executable languages simultaneously with the verification of the behaviour of a given BPD versus some a priori specification (SP, SA, etc.). The idea is to go from manual to systematic (semi-automatic) and then to automatic approaches ([ACCL06]). Heuristic verification and debugging are not excluded.

If we look at the General Relationship Diagram (Figure 1), future research have to include general studies to relate the class of BPDs, BWP, WPNs, etc. in order to obtain a concrete semantics, useful both for implementing and verifying business processes.

The suggested methodology for specification and verification of (workflow) business processes has to be applied mainly to safety risky (mission critical, safety critical, etc) systems.

References