RC23830 (C0512-005) December 14, 2005
Computer Science

IBM Research Report

BPSL Modeler - Visual Notation Language for Intuitive Business Property Reasoning

Xu Ke*, Liu Ying, Wu Cheng*
IBM Research Division
China Research Laboratory
HaoHai Building, No. 7, 5th Street
ShangDi, Beijing 100085
China

*Department of Automation
Tsinghua University
Beijing, China

BPSL Modeler - Visual Notation Language for Intuitive Business Property Reasoning

XU Ke [1]

Department of Automation, Tsinghua University
Beijing, China

LIU Ying [2]

IBM China Research Laboratory
Beijing, China

WU Cheng

Department of Automation, Tsinghua University
Beijing, China

Abstract

The urgent need for reliable business applications demands the emergence of a powerful yet easy-to-use language for business property reasoning. The Business Property Specification Language (BPSL) and its supporting tool (BPSL modeler) are presented to address the issue. BPSL modeler facilitates the specification and understanding of business properties by simplifying complex logics and common behaviors in business processes and exploiting intuitive notations for property representation. It also absorbs existing knowledge in the practices and researches in business domain so as to make BPSL modeler tailored for the application in business domains. Important ideas and features of BPSL modeler, together with the syntax and formal semantics are provided to help understand its effectiveness.

Key words: Business Property Specification Language, Temporal Logic, Business Property Template, Formal Verification

1 Introduction

Minimizing potential pitfalls in business applications is critical with the growth of complexity in existing business systems. Driven by the urgent need for en-

[1] Email: `xk02@mails.tsinghua.edu.cn`
[2] Email: `aliceliu@cn.ibm.com`

suring reliable business applications, it has been recognized to be a promising approach to integrate formal verification techniques like model checking[4] into business domains to help efficiently verify their business processes[13].

Specifying user desired properties is a critical step in the formal verification of business processes. These properties, as expressed in formalisms such as temporal logics of CTL (Computation Tree Logic)[4] or LTL (Linear Time Logic)[4], captures specific user requirements on business processes and are thus called business properties (in short, *BP*). However, lacking an intuitive and easy-to-use specification language for business properties is currently a serious obstacle for business analysts to write and understand their desired properties since their limited knowledge in logical reasoning makes it impossible for them to use the complex and rigid logics directly. Therefore, tool support for the intuitive specification of business properties, automatic generation of logical formulas is critical for applying formal verification techniques.

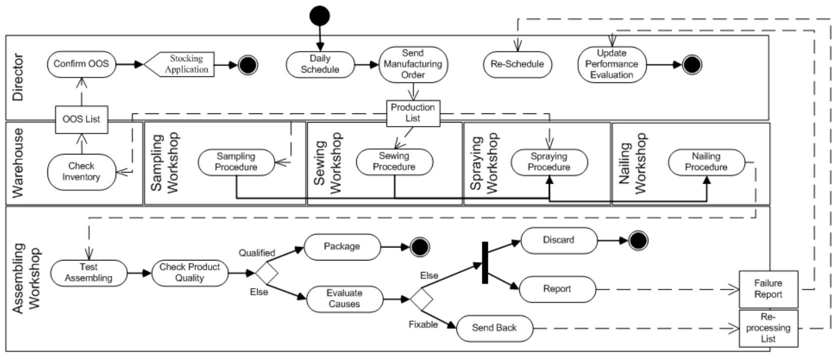


Fig1. Business Process for Sofa Manufacturing

To be more specific, take the following real scenario as an example. Figure 1 illustrates a partial model of a sofa manufacturing process in UML activity diagram. In the process, the factory director delegates manufacturing tasks to different workshops (with details omitted). The assembly workshop assembles the semi-finished products and checks their qualities. The final product will either be packaged or sent back for re-processing. During the design of the process, it is desired that the business process should comply with some expected properties in order to ensure its reliability. For example: "*A product must eventually be packaged if it is re-processed less than three times. However, a failure report must to be generated after its second sent back for re-precessing and before the product is re-precessed the third time*". This property can be captured with the temporal logic of LTL as shown in figure 2.

$$\begin{aligned}&\text{ReportGeneration}=F\,((\neg \text{sentback}\ U\,(\text{sentback} \wedge (X\,\neg \text{sentback}\ U\,(\text{sentback} \wedge \text{report})))) \vee \\ &(\neg \text{sentback}\ U\,(\text{sentback} \wedge (X\,\neg \text{sentback}\ U\,(\text{sentback} \wedge (X\,\neg \text{sentback}\ U\,(\text{sentback} \wedge \text{report})))))) \vee \\ &G\,(\neg(\neg \text{sentback}\ U\,(\text{sentback} \wedge (X\,\neg \text{sentback}\ U\,(\text{sentback} \wedge (X\,\neg \text{sentback}\ U\,(\text{sentback} \wedge \text{true})))))) \wedge F\,\text{package})\end{aligned}$$

Fig2. LTL Specification of Report Generation Property

The complexity of specifying and understanding business properties with traditional temporal logics can be easily understood from the example. Therefore, the primary contribution of this work is the proposal of an easy-to-use visual notation language called BPSL and its supporting tool BPSL modeler that are tailored for the application in business domains. BPSL modeler enjoys the advantage of maintaining good usability and understandability for specifying business properties while still preserving strict formal semantics for each property to support their reasoning. These advantages are enabled by tailoring the language of BPSL and its operators for the specifying common behaviors in business processes and absorbing existing business knowledge into the language. As a comparison, we'll illustrate in section 5 how the above property can be more intuitively specified with BPSL modeler.

2 Related Works

An important working direction for facilitating temporal property specification is to provide visual extensions for existing logics[3][5][10]. This benefits users by helping understand different semantics of temporal operators with their visualized formalisms. On the other hand the Property Specification Language (PSL)[6], an IEEE standard in digital circuit community, focuses on the reduction of the complexity in property specification by providing a more flexible and redundant choice of temporal operators.

When taking a close investigation in business domain, in REALM (Regulations Expressed as Logical Models)[7], a specific extension of propositional temporal logics (PTL) is contained to specify compliance rules in business models. A domain specific model checking language tuned for business applications named Strix is proposed in [1]. The property specification in Strix directly uses CTL connectives to explore the business process model in Strix.

However, the above works still suffer from the following deficiencies:

- When complex property is considered (e.g. the one in figure 2), providing visual notations alone is not enough for making a specification language easy-to-use and a property intuitive to understand.
- Existing knowledge in specifying different business entities (e.g. activity, resource) and their relations (e.g. Response, Exclusion,) are not fully exploited to facilitate business property specification and understanding.
- There lacks the tool support for automatically generating formal semantics from intuitive business property representations so as to enable the quick integration between business process modelers and formal verification tools.

The visual notation language of BPSL is extended from PSL by specifically tuning it for business domains. Distinct features of BPSL modeler include:

- BPSL provides an intuitive representation of business properties such that business people can easily understand them with their existing knowledge and experience.
- By concluding frequently used categories of business property templates and providing the "push button" generation of their BPSL definitions, BPSL

modeler absorbs existing business knowledge and significantly facilitates business property specification.

- BPSL modeler supports the auto-generation of underlying formal semantics of each property based on both the logic of CTL and LTL. Consequently, it not only ensures the preciseness of BPSL, but also facilitates the reasoning of business models with existing formal verification techniques.

The rest of the paper is organized as follows. In the next section the framework and basic concepts in BPSL modeler are introduced to help understand its ideas. In section 4, important features in BPSL modeler are explained in detail together with the analysis of corresponding characteristics in business property specification. In section 5, it is illustrated how BPSL modeler can be used to specify complex business properties. Section 6 concludes the paper.

3 BPSL Modeler - Framework and Basic Concepts

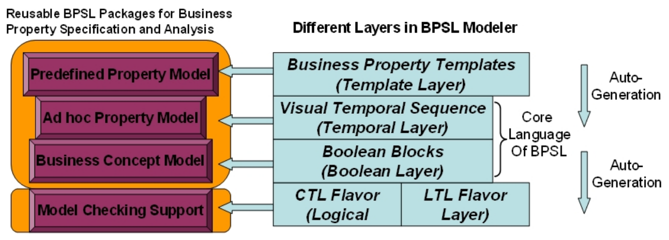

Reusable BPSL Packages for Business Property Specification and Analysis
Different Layers in BPSL Modeler
Predefined Property Model
Ad hoc Property Model
Business Concept Model
Model Checking Support
Business Property Templates (Template Layer)
Visual Temporal Sequence (Temporal Layer)
Boolean Blocks (Boolean Layer)
CTL Flavor (Logical
LTL Flavor Layer)
Core Language Of BPSL
Auto-Generation
Auto-Generation

Fig3. Framework of BPSL Modeler

To better understand the key ideas in BPSL modeler for business property specification, figure 3 provides an illustration of its framework. BPSL consists of a Boolean layer and a temporal layer. In Boolean Layer, Boolean Blocks (in short, *BB*) are basic elements for capturing the attributes of different business entities (e.g. activity, resource) and form a basic concept model for specific business domain. In temporal layer, Temporal Sequence (in short, *TS*) is the visual representation that specifies the logical relations between different business entities and the temporal constraints on specific business models. Above the two layers, BPSL modeler concludes and implements frequently used business properties or patterns from business experiences in the form of Business Property Templates. BPSL modeler supports the push button generation of the BPSL definition for each template and in turn the CTL/LTL definition for each *BP* as its formal semantics. Consequently, a direct benefit of the framework is that property specifications in BPSL modeler can be easily reorganized to form reusable BPSL packages for property generation in different business application domains. Besides, the CTL/LTL formal foundation also facilitates the integration of business property specification with existing formal verification techniques since the reasoning of the two temporal logics have a wide tool support such as RuleBase[2], etc. A full syntax, semantics and visual notations of BPSL are listed in Appendix 1-4.

In order to well understand the language of BPSL, figure 4 illustrates a sample specification of a "*MailChecking*" property built in BPSL modeler.

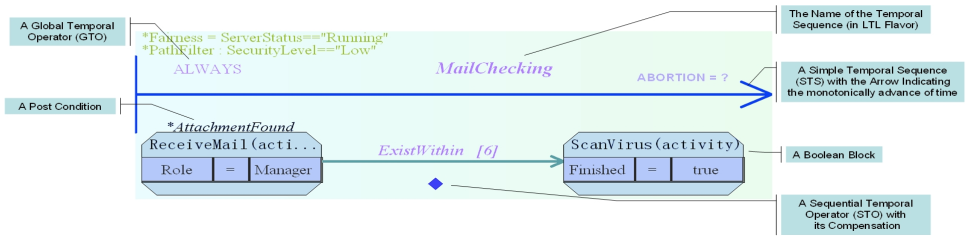

A Global Temporal Operator (GTO)
*Fairness = ServerStatus=="Running"
*PathFilter : SecurityLevel=="Low"
ALWAYS
MailChecking
ABORTION = ?
The Name of the Temporal Sequence (in LTL Flavor)
A Simple Temporal Sequence (STS) with the Arrow Indicating the monotonically advance of time
A Post Condition
*AttachmentFound
ReceiveMail(acti...
Role
=
Manager
ExistWithin [6]
ScanVirus(activity)
Finished
=
true
A Boolean Block
A Sequential Temporal Operator (STO) with its Compensation

Fig4. Example Property in BPSL Molder

Briefly speaking, the business property orders that whenever a manager receives an e-mail, virus must be eventually scanned to ensure its security. We will first investigate the basic concepts of BPSL modeler involved in the example, and leave the introduction of its advanced features in the next section.

- ***Boolean Block(BB):*** Represented by an octagon, a *BB* is a three-tuple consists of its name (e.g. *ReceiveMail*), stereotype (e.g. *Activity*) and a set of atomic attributes (e.g. the role of manager). It can thus be used to identify specific business entity that are qualified by the pre-defined conditions in *BB*. For example, the above *BB* of *ReceiveMail* indicates any activity that is responsible by a manager in a business process model.
- ***Simple Temporal Sequence (STS):*** A *STS* specifies the temporal relations among a sequence of *BB*s or business properties along paths where time advances monotonically. As can be found in Appendix 1-2, BPSL supports 8 stereotypes of Sequential Temporal Operators (*STO*s) with different semantics to specify these relations in different situations. The example in figure 4 is itself a *STS*. It specifies the sequential relation between two *BB*s so as to ensure that a virus scan activity should be finished within 6 operations after a manager receives an e-mail.
- ***Compound Temporal Sequence (CTS):*** Different from *STS*, a *CTS* specifies the different logical relations (e.g. *And, Or, Not, Imply, IFF*) and predefined temporal relations (e.g. *Before, After, Until*) between two *STS*s with Compound Temporal Operators (*CTO*). Notation and syntax of *CTO* can also be referred in Appendix 1, 2 respectively.
- ***LTL and CTL Flavor:*** In order to enhance the expressiveness of BPSL and obtain a wider tool support for formal verification, temporal sequences in BPSL can be interpreted in either LTL or CTL flavor. *TS*s in different flavors possess different available temporal operators in order to avoid the confusion of their semantics. For example, the temporal sequence of *MailChecking* in figure 4 is a LTL flavored specification. As shown in Appendix 1 and 2, different *STO*s are associated with different notations so as to distinguish their semantics in the context of CTL and LTL (e.g. *Next, Within, MultiWithinOnEvent*, etc in LTL flavor and *PossiblyLeadsTo, CertainlyLeadsTo, CertainlyLeadsToBeforeEvent*, etc in CTL flavor). These temporal operators can be used to express rather complex temporal rela-

tions in a simple and compact manner, e.g. some states should be reached within a certain range of steps in the process (*Within*); some states should be reached n times when certain event holds in the process (*MultiWithinOnEvent*); some states should be reached before certain event takes place in the process (*CertainlyLeadsToBeforeEvent*). It is then *BPSL* modeler's job to automatically generate the logical formalisms foreach property according to its semantics in different flavors and hide the complexity from users.

- ***Global Temporal Operator (GTO):*** Four types of *GTO*s are supported *"Always", "Eventually", "Repeat" and "Never"*. *GTO* specifies whether a *TS* should hold in all circumstances when a business process runs (*Always*, as is used in our example), or it is sufficient to only ensure that the *TS* can hold in some cases in the execution of the business process (*Eventually*), or to guarantee that the *TS* must hold at least n times or never holds in the business process model (*Repeat* and *Never* respectively).
- ***Abortion:*** The abortion condition in a *TS* indicates the circumstance in which the evaluation of a *TS* should be forced to stop. In the above case, the question mark indicates that no abortion condition is explicitly specified.
- ***Post condition:*** A post condition can be associated with *BB*s in *TS*. It specifies whether the rest of the *TS* after a *BB* is necessary to be further evaluated. For example, the post condition "*AttachmentFound*" associated with "*ReceiveMail*" in the example indicates that only when an attachment is found in the e-mail should the scan virus activity be eventually performed.

4 Features of BPSL and BPSL Modeler

This section introduces the advanced features in BPSL modeler in accordance with the analysis of some characteristics in business property specification.

4.1 Sequence Composition and Grouping

Business people may not be familiar with logical reasoning, but most of them are familiar with business process representations like DAG (Directed Acyclic Graph) or UML Activity Diagram. Therefore, in order to make property visualization in BPSL more acceptable to business people by exploiting their existing experience, BPSL properties are also presented in a "process" oriented form. That is, temporal sequences in BPSL not only specifies the temporal relations between two *BB*s or *BP*s, but can actually be composed to form a rather long chain of property sequence with And-Fork, Or-Fork and Join relations. Figure 5 illustrates such an example.

The fork, join operators specify the different and/or relations when evaluating each branches in the temporal sequence. Besides, in BPSL the grouping of a partial temporal sequence is also supported which corresponds to the concept of sub-process in business process model. In figure 5(b) the details of part of the temporal sequence "*subSTS*" are hidden by grouping the corresponding parts in the original *STS*. Another advantage of sequence composition and grouping is that it can abbreviate the specification of complex business properties in BPSL and flexibly adjust the granularity of property representation.

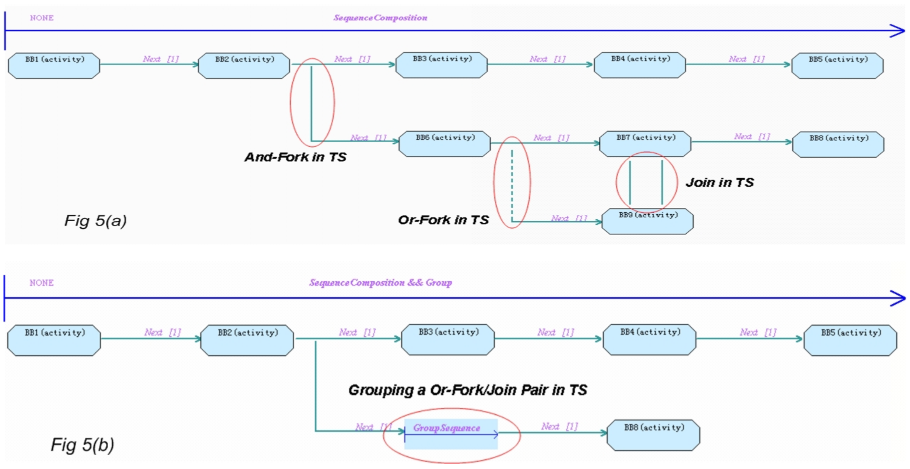


Fig5. Temporal Sequence Composition and Grouping

4.2 Property Compensation

Traditional logical operators support specifying the question of "If certain condition holds, what a model should satisfy?" (e.g. the *imply* operator). However, for a comprehend understanding of a business model, business analysts often tend to ask another half of the question, that is: "*But when that condition never holds, what the process should behave like then?*". For example, in the property of figure 4, what if there is never a sixth operation after a manager receives an email in the process? Therefore, compensation operators can be associated in BPSL properties to make business property specification in these unusual situations more direct and intuitive. Compensation operator refers to a user defined compensation property that must hold true when some compensation conditions (e.g. there is no sixth operation in the future) are triggered. These compensation conditions can be dynamically computed by function *compute()* as shown in Appendix 3 when the compensation operator is associated with different operators in the BPSL property. As a result, the compensation in figure 4 implies that there must be at least 6 additional operations after an e-mail is received in order to make the property hold because the associated compensation property is "false" (as denoted by a diamond).

4.3 Filtering and Fairness

Business processes can be rather complex so as to provide a full view of the daily businesses in an enterprise. When reasoning a business process it is in some cases desired that redundant information (e.g. exception handling) which may not be the primary concern for business analysts can be neglected to avoid reaching wrong conclusions. Therefore, it will be of great help if the granularity of business reasoning can be flexibly controlled in the step of business property specification. BPSL provides this mechanism by supporting filter and fairness conditions. Borrowed from model checking[4], the fairness condition specifies that the evaluation of business property will only be done on process execution paths where it can hold infinitely often. On the contrary, the filter condition specifies that all the execution paths in the business process along which the filter condition may be satisfied will be neglected in the property evaluation. For example, recall the property in figure 4. Its filter condition implies that the mail checking property will not be evaluated in the cases where the security level of the business model is low. Its fairness condition, on the other hand, demands that such a property should only be evaluated in situations when the mail server is always correctly running.

4.4 Business Property Templates

Years of practices and researches in business domains form a considerable accumulation of relevant knowledge and experiences. Tuned for the application in business domain, BPSL modeler provides the capability of capturing these existing results in the form of business property templates to facilitate both property specification and understanding. Four pre-defined common categories of business property templates are concluded in BPSL modeler based on which frequently used business patterns can be automatically generated and reused. The definition of these business property templates can be automatically implemented within the expressiveness of BPSL and can in turn be formally interpreted in CTL / LTL with BPSL modeler. As illustrated below, the four categories of business templates are Soundness Templates, General Temporal Templates, Business Bug Templates and Functional Templates.

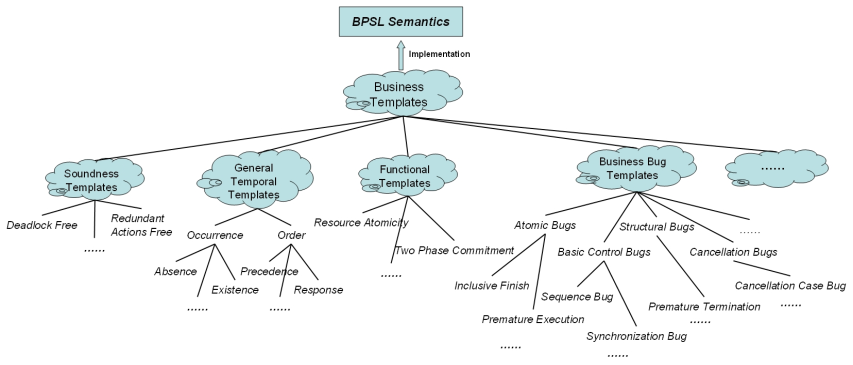
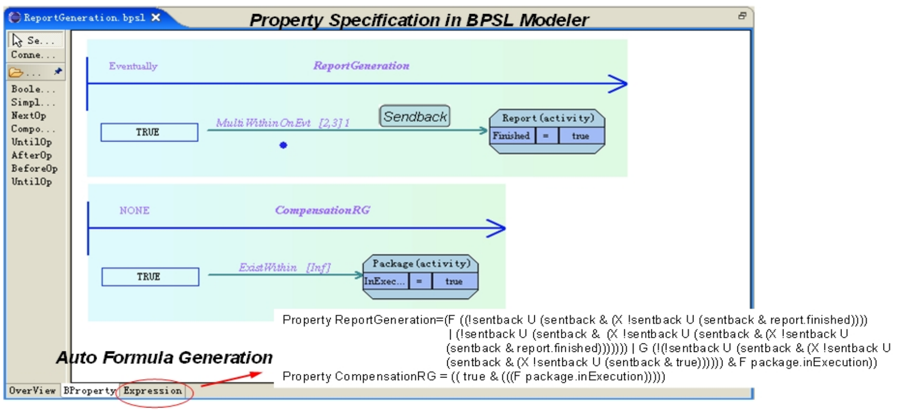


Fig6. Business Property Templates in BPSL

Soundness template concludes the commonly used workflow soundness[8] definitions (e.g. deadlock free, redundant actions free, etc) that each business process may satisfy; General temporal template implements the general temporal patterns that are based on the survey result in [9] and their semantic mapping[11] on CTL/LTL; Business bug template corresponds to traditional workflow patterns[12]. Each bug template can be used to falsify a true realization of a specific workflow pattern in the business process model; And functional template captures some useful functional requirements in business processes. For example, *ResourceAtomicity* can be used to impose constraints on the process such that "for certain kind of resource (like money), it should never be created or destroyed in the process".

5 Specifying Business Properties with BPSL Modeler



```
Property ReportGeneration=(F ((!sentback U (sentback & (X !sentback U (sentback & report.finished))))
                          | (!sentback U (sentback & (X !sentback U (sentback & (X !sentback U
                          (sentback & report.finished)))))) | G (!(!sentback U (sentback & (X !sentback U
                          (sentback & (X !sentback U (sentback & true)))))) & F package.inExecution))
Property CompensationRG = (( true & (((F package.inExecution)))))
```

Fig7. Re-specify ReportGeneration property in BPSL Modeler

Now that the concepts and features of BPSL modeler are introduced, it can be shown now how the complex property in figure 2 can be instead intuitively specified with BPSL modeler. BPSL modeler is developed as a plugin for Eclipse platform. Figure 7 illustrates the result built with BPSL modeler. The specification is simplified into a short LTL flavored *STS* which is more intuitive for understanding. The "*MultiWithinOnEvent*" sequential temporal operator specifies that within the second and the third occurrence of event "*sendback*", the Boolean Block for identifying the finished activity of Report must hold once. Besides, the compensation property for the operator specifies that in case there is never a third occurrence of event "*sendback*", a package activity will be eventually in execution. Note that this compensation property can be directly generated from business property template of *GlocalExistence* in figure 6 to minimize specification efforts. The auto-generated formal semantics of this specification with BPSL modeler shows that it is actually expressing the same formula as the one in figure 2.

6 Summary

Business property specification is a major step in reasoning business process models and ensuring its reliability. In this paper, the visual notation language of BPSL and its supporting tool BPSL modeler are proposed to enable the intuitive specification of business properties and facilitate the reasoning of business process models. BPSL modeler simplifies the complexity of business property specification by taking different business characteristics into consideration, e.g. the visual representation, property compensation and filtering, etc. It exploits existing knowledge in the practices and researches in business domain to make property specification of minimum efforts. In BPSL modeler, both the logics of LTL and CTL are supported and formal semantics can be auto-generated for each business property template and business property so as to ease the integration between BPSL modeler and existing formal verification tools. Our future work will include extending the application of BPSL modeler into more real cases to verify its values.

References

[1] Bard, B. *Seeing by Owl-Light - Symbolic Model Checking of Business Requirements*, Technical Report, IBM T.J. Watson Research Lab, 2002.

[2] Beer, I.S., B. David, et al. *RuleBase: Model checking at IBM*, International Conference on Computer Aided Verification, 1997, 480-483.

[3] Brambilla, M., A. Deutsch, et al. *The role of visual tools in a Web application design and verification framework: A visual notation for LTL formulae*, Lecture Notes in Computer Science, **3579**, 2005, 557-568.

[4] Clarke, E.M., Jr. O G., D. A. Peled, *Model Checking*, MIT Press, 1999.

[5] Del, B., A. L. Rella, E. Vicario, *Visual specification of branching time temporal logic*, Proc. 11th IEEE Symp. on Visual Languages, 1995, 61-68.

[6] Geist, D., *The PSL/Sugar specification language a language for all seasons*, Lecture Notes in Computer Science, **2800**, 2003, 3.

[7] Giblin, C., Y. Liu, et al. *Regulations Expressed As Logical Models (REALM)*, 18th Annual Conference on Legal Knowledge and Information Systems, IOS Press, Accepted, 2005.

[8] Hofstede, A., M. Orlowska, J, Rajapaske. *Verification problems in conceptual workflow specification*, Data and Knowledge Engineering, **24(3)**, 1998, 239-256.

[9] Matthew, B.D., S. A. George, C. C. James. *Patterns in Property Specifications for Finite-State Verification*, Proc. 21st International Conference on Software Engineering, 1999.

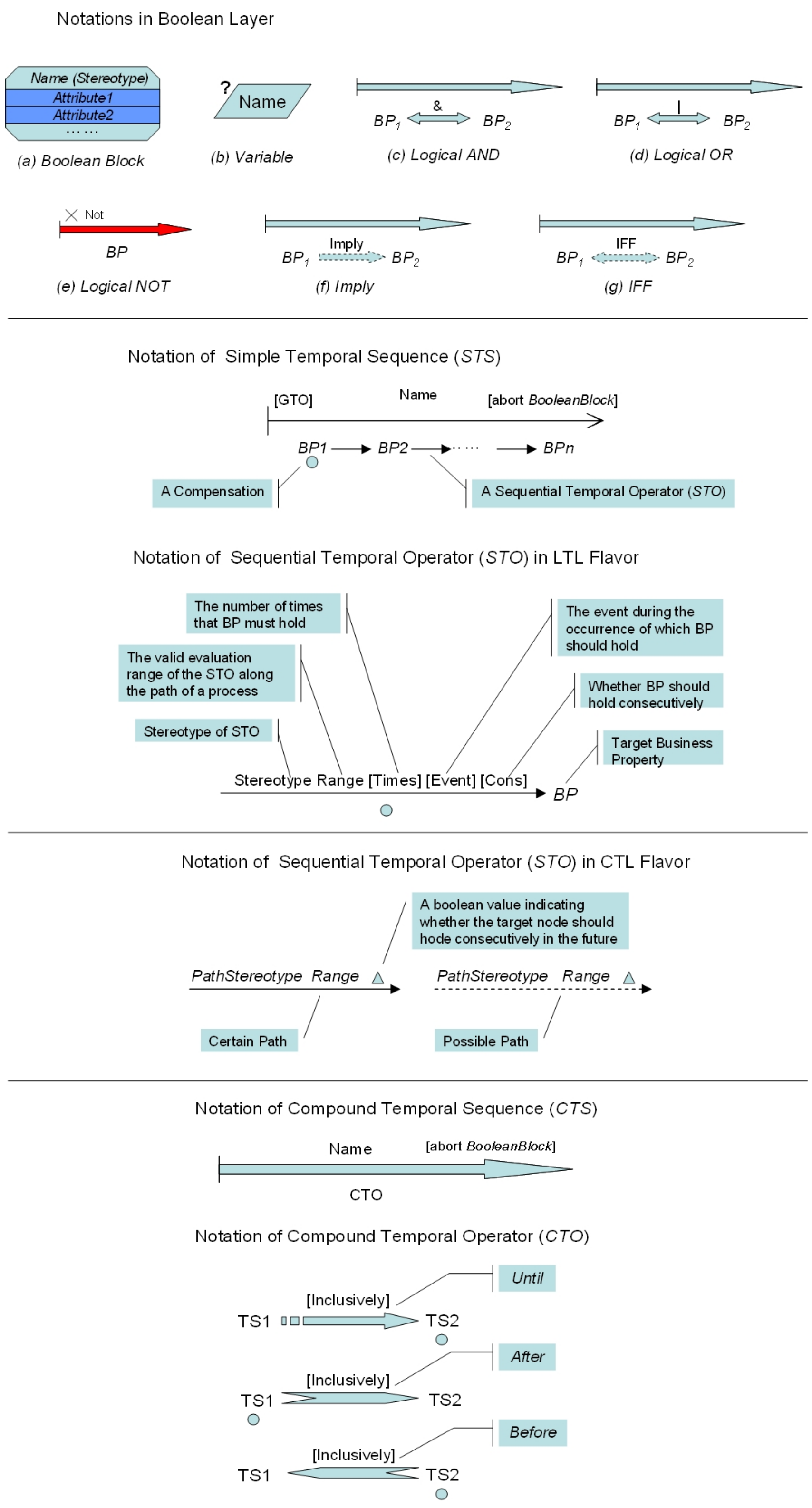
[10] Rao, A. C., A. Cau, H. Zedan, *Visualization of Interval Temporal Logic*, Proc. 5th Joint Conference on Information Sciences, 2000, 687-690.

[11] Property Specification Patterns, 2005, http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml

[12] van der Aalst, W.M.P, ter Hofstede, A.H.M, et al. *Workflow patterns*, Distributed and Parallel Databases, **14**, 2003, 5-51.

[13] Wang, W. L., Z. Hidvegi, et al. *E-process design and assurance using model checking*, **33(10)**, 2004, 48-53.

7 Appendix


Notations in Boolean Layer
Name (Stereotype)
Attribute1
Attribute2
……
(a) Boolean Block
?
Name
(b) Variable
&
BP1 BP2
(c) Logical AND
|
BP1 BP2
(d) Logical OR
Not
BP
(e) Logical NOT
Imply
BP1 BP2
(f) Imply
IFF
BP1 BP2
(g) IFF
Notation of Simple Temporal Sequence (STS)
[GTO]
Name
[abort BooleanBlock]
BP1 BP2 …… BPn
A Compensation
A Sequential Temporal Operator (STO)
Notation of Sequential Temporal Operator (STO) in LTL Flavor
The number of times that BP must hold
The valid evaluation range of the STO along the path of a process
Stereotype of STO
The event during the occurrence of which BP should hold
Whether BP should hold consecutively
Target Business Property
Stereotype Range [Times] [Event] [Cons]
BP
Notation of Sequential Temporal Operator (STO) in CTL Flavor
A boolean value indicating whether the target node should hode consecutively in the future
PathStereotype Range
PathStereotype Range
Certain Path
Possible Path
Notation of Compound Temporal Sequence (CTS)
Name
[abort BooleanBlock]
CTO
Notation of Compound Temporal Operator (CTO)
Until
[Inclusively]
TS1
TS2
After
[Inclusively]
TS1
TS2
Before
[Inclusively]
TS1
TS2

Appendix 1. Visual Notations in BPSL

```
BP::=TS | BT | BE
BE::=BooleanBlock | BE OR BE | BE AND BE| NOT BE | BE IMPLY BE | BE IFF BE
BooleanBlock::=TRUE | FALSE | (name, stereotype, attrs)
attrs::=    TRUE |FALSE |atom = atom |atom > atom |atom < atom |atom >= atom |atom <= atom
            |atomset ⊃ atomset |atomset ⊂ atomset |attrs & attrs|attrs | attrs |!attrs |(attrs)
atom::=     symbol | number                          atomset::= (atom)+
```

```
TS::= STS | CTS | TS OR TS | TS AND TS | NOT TS | BE IMPLY TS | TS IFF TS
STS::=[GTO] StartPoint (STO)* [abort BE] [Pathfilter] [Fairness] [Compensate BP]
StartPoint::=BP      Pathfilter::=BP      Fairness::=BE

Stereotype::=next | next_all | next_repeat | next_eventually | next_event |
             next_event_all | next_event_repeat | next_event_eventually
Range::=IntegerRange | IntervalRange | InfinityRange
IntervalRange::=BoundRange | BooleanRange
BoundRange::=LowerBound, UpperBound
LowerBound::=IntegerRange            UpperBound::=IntegerRange
IntegerRange::=number                BooleanRange::=BE
Times::=number                       Event::=BE

CTS::=CTO [abort BE] [Compensate BP]
CTO::=TS Until TS [Inclusive] |TS After TS [Inclusive] |TS Before TS [Inclusive]
```

```
STO::= STO_LTL | STO_CTL
STO_LTL ::= next IntegerRange BP
      |next_all IntervalRange BP        | next_all InfinityRange BP
      |next_repeat IntervalRange Times [Cons] BP
      |next_eventually BoundRange BP | next_eventually InfinityRange BP
      |next_event IntegerRange Event BP
      |next_event_all BoundRange Event BP
      |next_event_repeat BoundRange Times Event [Cons] BP
      |next_event_eventually BoundRange Event BP
      |STO [postcondition]
STO_CTL ::=[Possible | Certain]  PathStereotype [cons] [postcondition]
PathStereotype ::= LeadtoBefore IntegerRange | InfinityRange
                  |LeadtoBeforeEvent event
event ::= BE              cons ::= TRUE | FALSE
```

Appendix 2. Syntax of BPSL in both CTL and LTL flavor

Property Compensation (Notations):

- ▢ denotes true – Enables a property to hold weakly
- ◇ denotes false (default) – Enables a property to hold strongly
- ○ denotes customized compensation property

Computing Compensation Conditions for a Property:

$$Compute() :\ STO \mid CTO \mid Imply \rightarrow BP$$

$$Compute(STO) = \begin{cases} next\ IntegerRange\ true & \text{if '}Range\text{' is } IntegerRange \text{ in } STO \text{ and '}event\text{' is not specified} \\ next\ UpperBound\ true & \text{if '}Range\text{' is } IntervalRange \text{ in } STO \text{ and '}event\text{' is not specified} \\ next\ 1\ (Eventually\ BE) & \text{if '}Range\text{' is } BooleanRange \text{ in } STO \text{ and '}event\text{' is not specified} \\ next\_event\ IntegerRange\ event\ true & \text{if '}Range\text{' is } IntegerRange \text{ in } STO \text{ and '}event\text{' is specified} \\ next\_event\ UpperBound\ event\ true & \text{if '}Range\text{' is } IntervalRange \text{ in } STO \text{ and '}event\text{' is specified} \end{cases}$$

$$Compute(CTO) = \begin{cases} BP2 & \text{if } CTO = BP1\ Until\ BP2 \\ BP1 & \text{if } CTO = BP1\ After\ BP2 \\ BP2 & \text{if } CTO = BP1\ Before\ BP2 \end{cases}$$

$$Compute(BE1\ Imply\ BE2) = BE1$$

Appendix 3. Syntax and Semantics of Property Compensation in BPSL

The semantics of BPSL in either LTL or CTL flavor is defined as a relation K,pi|=∅, where pi / s is a path / state in Kripke Structure K. Note that similar semantics in CTL flavor that has already been clarified in the semantics in LTL flavor are omitted in the following

$K,\pi \models BooleanBlock \quad \text{if } \forall p \in props \text{ where } BooleanBlock = (name, stereotype, props),\ K, s_0 \models p$

$K,\pi \models BPCompensate\ BP' \quad \text{if } (K,\pi \models Compute(BP) \wedge K,\pi \models BP) \vee (K,\pi \not\models Compute(BP) \wedge K,\pi \models BP')$

$K,\pi \models BP1\ Imply\ BP2* \quad \text{if } K,\pi \models BP1 \wedge K,\pi \models BP2$ (Note that further compensation can be considered when $BP1$ does not hold)

$K,\pi \models BP1\ IFF\ BP2 \quad \text{if } (K,\pi \models BP1 \wedge K,\pi \models BP2) \vee (K,\pi \not\models BP1 \wedge K,\pi \not\models BP2)$

$K,\pi \models STO1\ post\ STO2 \quad \text{if } K,\pi \models STO1 \wedge K,\pi \models (post- > STO2)$ where $post$ is the post-condition of STO1

$K,\pi \models STS\ filter \quad if\ K,\pi \models (!G(b)- > STS)$ where $filter$ is the path filter

$K,\pi \models Eventually\ BP \quad \text{if } K,\pi \models F\ BP$

$K,\pi \models Possibly\ Eventually\ BP \quad \text{if } K,\pi \models \neg(G\ \neg BP)$

$K,\pi \models Always\ BP \quad \text{if } K,\pi \models G\ BP$

$K,\pi \models Possibly\ Always\ BP \quad \text{if } K,\pi \models \neg(F\ \neg BP)$

$K,\pi \models Repeat\ i\ BP \quad \text{if } K,\pi \models \underbrace{F\ (BP \wedge (X\ F\ BP \wedge (\ldots\ldots \wedge (X\ F\ BP))))}_{i\ times}$

$K,\pi \models Never\ BP \quad \text{if } K,\pi \models G\ \neg BP$

$K,\pi \models Next\ i\ BP* \quad \text{if } K,\pi \models \underbrace{X\ X\ \ldots X\ BP}_{i\ times}$

$K,\pi \models AllWithin\ i,j\ BP* \quad \text{if } K,\pi \models (Next\ i\ BP) \wedge (Next\ i+1\ BP) \wedge \ldots \wedge (Next\ j\ BP)$

$K,\pi \models AllWithin\ b\ BP* \quad \text{if } K,\pi \models X\ (BP\ U\ b)$ where b is a $BooleanBlock$(and so is the following)

$K,\pi \models AllWithin\ \text{inf}\ BP* \quad \text{if } K,\pi \models G\ BP$

$K,\pi \models MultiWithin\ i,j\ t\ BP* \quad \text{if } \exists k_1,\ldots,k_t \in [i,j],\ k_p \neq k_q,\ p \neq q,\ s.t.\ K,\pi \models (Next\ k_1\ BP) \wedge (Next\ k_2\ BP) \wedge \ldots \wedge (Next\ k_t\ BP)$

$K,\pi \models MultiWithin\ i,j\ t\ cons\ BP* \quad \text{if } \exists k,\ k > i \wedge k+t\text{-}1 < j,\ s.t.\ K,\pi \models AllWithin\ k,k+t-1\ BP$

$K,\pi \models MultiWithin\ b\ t\ BP* \quad \text{if } \exists k,\ s.t.\ (K,\pi \models AllWithin\ 0,k\ \neg b) \wedge (K,\pi \models MultiWithin\ 0,k\ t\ BP)$

$K,\pi \models MultiWithin\ b\ t\ cons\ BP* \quad \text{if } \exists k,\ s.t.\ (K,\pi \models AllWithin\ 0,k\ \neg b) \wedge (K,\pi \models MultiWithin\ 0,k\ t\ cons\ BP)$

$K,\pi \models ExistWithin\ i,j\ BP* \quad \text{if } K,\pi \models (Next\ i\ BP) \vee (Next\ i+1\ BP) \vee \ldots \vee (Next\ j\ BP)$

$K,\pi \models ExistWithin\ \text{inf}\ BP* \quad \text{if } K,\pi \models F\ BP$

$K,\pi \models NextOnEvent\ 1\ b\ BP* \quad \text{if } K,\pi \models (\neg b\ U\ b \wedge BP)$

$K,\pi \models NextOnEvent\ i\ b\ BP* \quad \text{if } K,\pi \models \underbrace{NextOnEvent\ 1\ b\ (NextOnEvent\ 1\ b\ (\ldots(NextOnEvent\ 1\ b\ BP)))}_{i\ times}$

$K,\pi \models AllWithinOnEvent\ i,j\ b\ BP* \quad \text{if } K,\pi \models (NextOnEvent\ i\ b\ BP) \wedge (NextOnEvent\ i+1\ b\ BP) \wedge \ldots\ldots \wedge (NextOnEvent\ j\ b\ BP)$

$K,\pi \models MultiWithinOnEvent\ i,j\ t\ b\ BP* \quad \text{if } \exists k_1,\ldots,k_t \in [i,j],\ k_p \neq k_q,\ p \neq q,\ s.t.\ K,\pi \models (NextOnEvent\ k_1\ b\ BP) \wedge$
$(NextOnEvent\ k_2\ b\ BP) \wedge \ldots\ldots \wedge (NextOnEvent\ k_t\ b\ BP)$

$K,\pi \models MultiWithinOnEvent\ i,j\ t\ b\ cons\ BP* \quad \text{if } \exists k,\ k \geq i \wedge k+t \leq j+1,\ s.t.\ K,\pi \models AllWithinOnEvent\ k,k+t-1\ b\ BP$

$K,\pi \models ExistWithinOnEvent\ i,j\ b\ BP* \quad \text{if } K,\pi \models (NextOnEvent\ i\ b\ BP) \vee (NextOnEvent\ i+1\ b\ BP) \vee \ldots\ldots \vee (NextOnEvent\ j\ b\ BP)$

$K,\pi \models BP1\ Until\ BP2* \quad \text{if } K,\pi \models BP1\ U\ BP2$

$K,\pi \models BP1\ Until\ BP2\ Inclusive* \quad \text{if } K,\pi \models BP1\ U\ BP1 \wedge BP2$

$K,\pi \models BP1\ After\ BP2* \quad \text{if } K,\pi \models F\ (BP2 \wedge X\ F\ BP1)$

$K,\pi \models BP1\ After\ BP2\ Inclusive* \quad \text{if } K,\pi \models F\ (BP2 \wedge F\ BP1)$

$K,\pi \models BP1\ Before\ BP2* \quad \text{if } K,\pi \models \neg BP2\ U\ (\neg BP2 \wedge BP1)$

$K,\pi \models BP1\ Before\ BP2\ Inclusive* \quad \text{if } K,\pi \models \neg BP2\ U\ BP1$

$K,\pi \models BP\ abort\ b \quad \text{if } K,\pi \models BP \vee K,\pi \models BP1\ After\ BP2\ Compensate\ true$

(* indicates that potential compensation is available for the evaluation of the property)

$K,\pi \models SimpleTemporalSequence(STS)$ where $STS = BP\ STO_1,\ STO_2,\ \ldots\ldots$

if $K,\pi_0 \models BP \wedge K,\pi_{start(STO_1)} \models BP_1 \wedge K,\pi_{start(STO_1)+start(STO_2)} \models BP_2 \wedge \ldots\ldots$

$$\text{where start(STO}_k\text{)=}\begin{cases} i \\ \quad \text{if } 'Range' \text{ is } IntegerRange(i) \text{ in } STO_k \text{ and } 'event' \text{ is not specified} \\ j \\ \quad \text{if } 'Range' \text{ is } IntervalRange(i,j) \text{ in } STO_k \text{ and } 'event' \text{ is not specified} \\ d,\ \text{where } d \text{ is the next occurrence of } BooleanBlock\ b \text{ from the current state} \\ \qquad \text{and can be computed by formula } next\_event\ 1\ b\ true \\ \quad if\ 'Range' \text{ is } BooleanRange(b) \text{ in } STO_k \text{ and } 'event' \text{ is not specified} \\ d,\ \text{where } d \text{ is the } i\text{th next occurrence of } event\ b \text{ from the current state} \\ \qquad \text{and can be computed by formula } next\_event\ i\ b\ true \\ \quad if\ 'Range' \text{ is } IntegerRange(i) \text{ in } STO_k \text{ and } 'event' \text{ is specified} \\ d,\ \text{where } d \text{ is the } j\text{th next occurrence of } event\ b \text{ from the current state} \\ \qquad \text{and can be computed by formula } next\_event\ j\ b\ true \\ \quad if\ 'Range' \text{ is } IntervalRange(i,j) \text{ in } STO_k \text{ and } 'event' \text{ is specified} \end{cases}$$

$K,s \models AX\ b \quad if\ K,s \models \neg EX\ (\neg b)$

$K,s \models EF\ b \quad if\ K,s \models E(true\ U\ b)$

$K,s \models AG\ b \quad if\ K,s \models \neg EF\ (\neg b)$

$K,s \models AF\ b \quad if\ K,s \models \neg EG\ (\neg b)$

$K,s \models A(b1\ U\ b2) \quad if\ K,s \models \neg E(\neg b2\ U\ (\neg b2 \wedge \neg b1)) \wedge \neg EG\ (\neg b2)$

$K,\pi \models STS\ filter \quad if\ K,s \models (!AG(b)- > STS)$ where $filter$ is the path filter

$K,\pi \models Eventually\ BP \quad \text{if } K,s \models AF\ BP$

$K,\pi \models Possibly\ Eventually\ BP \quad \text{if } K,s \models EF\ BP$

$K,\pi \models Always\ BP \quad \text{if } K,s \models AG\ BP$

$K,\pi \models Possibly\ Always\ BP \quad \text{if } K,\pi \models EF\ BP$

$K,\pi \models Repeat\ i\ BP \quad \text{if } K,s \models \underbrace{AF\ (BP \wedge (AX\ AF\ BP \wedge (\ldots\ldots \wedge (AX\ AF\ BP))))}_{i\ times}$

$K,\pi \models Never\ BP \quad \text{if } K,\pi \models AG\ \neg BP$

$K,s \models Possible\ LeadstoBefore\ i\ node \quad \text{if } K,s \models EX(node) \vee EX(EX(node)) \vee \ldots \vee \underbrace{EX(EX\ldots EX(node))}_{i\ times}$

$K,s \models Certain\ LeadstoBefore\ i\ node \quad \text{if } K,s \models AX(node) \vee AX(AX(node)) \vee \ldots \vee \underbrace{AX(AX\ldots AX(node))}_{i\ times}$

$K,s \models Possible\ LeadstoBefore\ Infinity\ node \quad \text{if } K,s \models EF(node)$

$K,s \models Certain\ LeadstoBefore\ Infinity\ node \quad \text{if } K,s \models AF(node)$

$K,s \models Possible\ LeadstoBeforeEvent\ b\ node \quad \text{if } K,s \models (\neg(A(\neg(node\ U\ b)))) \wedge EF\ b$

$K,s \models Certain\ LeadstoBeforeEvent\ b\ node \quad \text{if } K,s \models (\neg(E(\neg(node\ U\ b)))) \wedge AF\ b$

$K,s \models Possible\ LeadstoBefore\ i\ cons\ node \quad \text{if } K,s \models EX(node) \wedge EX(EX(node)) \wedge \ldots \wedge \underbrace{EX(EX\ldots EX(node))}_{i\ times}$

$K,s \models Certain\ LeadstoBefore\ i\ cons\ node \quad \text{if } K,s \models AX(node) \wedge AX(AX(node)) \wedge \ldots \wedge \underbrace{AX(AX\ldots AX(node))}_{i\ times}$

$K,s \models Possible\ LeadstoBefore\ Infinity\ cons\ node \quad \text{if } K,s \models EG(node)$

$K,s \models Certain\ LeadstoBefore\ Infinity\ cons\ node \quad \text{if } K,s \models AG(node)$

$K,s \models Possible\ LeadstoBeforeEvent\ b\ cons\ node \quad \text{if } K,s \models E(node\ U\ b)$

$K,s \models Certain\ LeadstoBeforeEvent\ b\ cons\ node \quad \text{if } K,s \models A(node\ U\ b)$

$K,s \models Certain\ LeadstoBeforeEvent\ b\ cons\ node \quad \text{if } K,s \models A(node\ U\ b)$

Appendix 4. Semantics of BPSL in both CTL and LTL flavor