

# The Role of Visual Tools in a Web Application Design and Verification Framework: A Visual Notation for LTL Formulae

Marco Brambilla<sup>1</sup>, Alin Deutsch<sup>2</sup>, Liying Sui<sup>2</sup>, and Victor Vianu<sup>2</sup>

<sup>1</sup> Dipartimento Elettronica e Informazione, Politecnico di Milano,  
Via Ponzio 34/5, 20133 Milano, Italy  
mbrambill@elet.polimi.it

<sup>2</sup> Computer Science and Engineering Dept., UC San Diego,  
La Jolla, CA 92093-0114, USA  
{deutsch, lsui, vianu}@cs.ucsd.edu

**Abstract.** As the Web becomes a platform for implementing complex B2C and B2B applications, there is a need to extend Web conceptual modeling to process-centric applications. In this context, new problems about process safety and verification arise. Recent work has investigated high-level specification and verification of Web applications. This relies on a formal data-driven model of the application, which can access an underlying database as well as state information updated as the interaction progresses, and a set of user inputs. Properties verified concern the sequences of events, inputs, states, and actions resulting from the interaction. For the purpose of automatic verification, properties are expressed in linear-time or branching-time temporal logics. However, temporal logics properties are difficult to specify and understand by users, which can be a significant obstacle to the practical use of verification tools. In the present paper, we propose two alternative visual notations for specifying temporal properties. One alternative is to restrict the sequences of events using existing workflow specifications, such as BPMN, describing the execution flow of tasks within the application. However, such workflow formalisms have limited ability to express temporal properties. Another alternative is to develop a visual approach for explicitly specifying temporal operators, thus recovering their full expressiveness.

## 1 Introduction

Since the Web is becoming the most popular implementation platform for complex B2B applications, supporting business processes becomes a priority for Web application design, and development lifecycles should explicitly consider this aspect. The spread of Web applications interacting with users and programs while accessing an underlying database has been accompanied by the emergence of tools for their high-level specification [1, 10]. A representative, successful example is WebML [4, 11], which allows to specify a Web application using a visual interactive variant of the E-R model augmented with a workflow and query formalism. The code for the Web application is automatically generated from the WebML specification. This not only allows fast prototyping and productivity increment, but also provides a new opportunity for the automatic verification of Web applications.

We focus here on interactive Web applications modeled by WebML, generating Web pages dynamically by queries on an underlying database. The Web application accepts input from external users or programs. It responds by taking some action, updating its internal state database, and navigating to a new Web page determined by yet another query. A run is a sequence of inputs together with the Web pages, states and actions generated by the Web application. We use a WebML-style formalism proposed in [6], which models the queries used in the specification as first-order queries (FO).

As discussed in [6, 7], verification of high-level WebML-like specifications concerns properties of the sequences of events, inputs, states, and actions resulting from the interaction, which range from basic soundness of the specification (e.g. the next Web page to display is always uniquely defined) to semantic properties (e.g. no order is shipped before the payment is received). Of special interest are workflow-based properties, describing the execution flow of the tasks within the application. Those properties can capture activity execution constraints and special process features like pro-activity, exception handling, errors compensation. Such properties can be expressed using an extension of linear-time temporal logic (LTL), called LTL-FO [8]. Properties of runs of a Web application are defined by formulae using temporal operators such as  $G$ ,  $F$ ,  $X$ ,  $U$ , and  $B$ . For example,  $Fp$  means that  $p$  eventually holds; and  $pBq$  holds if either  $q$  always holds, or it eventually fails and  $p$  must hold sometime before  $q$  becomes false. Classical LTL formulae are built from propositional variables, using temporal and Boolean operators. An LTL-FO formula is obtained by combining FO formulae with temporal and boolean operators (but no further quantifications). The remaining free variables in the resulting formula are universally quantified at the very end. For example, the LTL-FO formula

$$\forall x \forall y \forall id [(pay(id, x, y) \wedge price(x, y)) B \neg Ship(id, x)]$$

states that whenever item  $x$  is shipped to customer  $id$ , a payment for  $x$  in the correct amount must have been previously received from customer  $id$ . Results in [6] show that it is decidable in PSPACE whether a Web application specification satisfies a LTL-FO formula, under a restriction called input boundedness. Input boundedness requires that all quantified variables range over values from user inputs, in all formulae used in the rules of the specification. And in [7], the authors implemented a verifier for high-level WebML-style specification languages, based on the result in [6].

While the results of [6, 7] on automatic verification are encouraging, describing formal models of applications and temporal logic properties is a very technical task, which many designers may not appreciate, since specifying even simple temporal properties can be complex and error-prone. Indeed, LTL properties are difficult for the average user involved in specification, design, development, and verification of Web applications since he is not a logic expert. To increase the likelihood of acceptance by users, a more user-friendly and easy to understand visual tool for specifying temporal properties is called for.

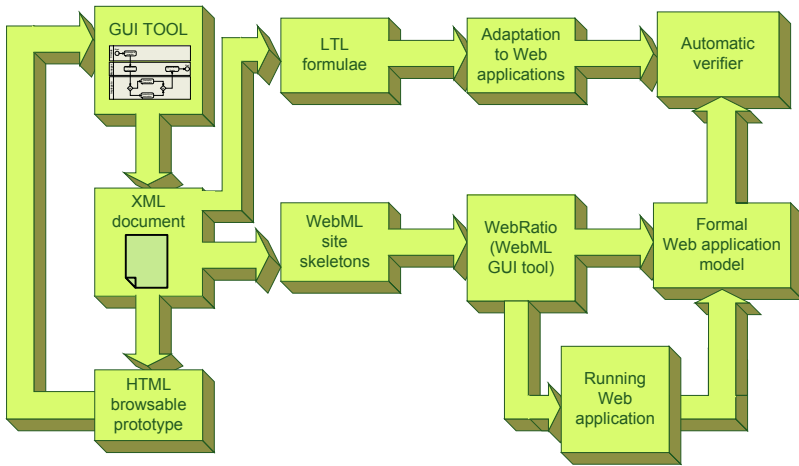
Existing workflow specification languages already provide a way to specify temporal constraints on the sequence of activities. Thus, they may be an appealing way to specify temporal properties. To investigate this possibility, we focus on BPMN, a well known notation for workflows. We begin by providing semantics to the BPMN notation in terms of LTL formulae. This has a twofold benefit: first, it allows compiling BPMN specifications into LTL formulas, which can then be passed on to a veri-

fier; second, it provides insight into the ability of BPMN diagrams to express LTL properties. In particular, it turns out that BPMN cannot express all LTL properties (for example, BPMN cannot express the  $X$  operator, or negation). Given such limitations of BPMN, we next consider an extension of this formalism with explicit temporal operators, which achieves full expressiveness relative to LTL. The extension is consistent with the workflow-oriented visual style of BPMN.

Other works use visual notations for model checking, but with a quite different flavor: in [5] lattices and other graph representations are used for multi-valued model-checking, useful for analyzing models that contain uncertainty or inconsistency; [9] uses LTL for automatic checking of diagrams representing architectural models.

## 2 Overall Framework

This section describes the general framework of our investigation, providing a comprehensive approach to the design and verification of workflow-based Web applications. We make use of several existing software tools, techniques and methodologies.



**Fig. 1.** Overall view of the proposed design and verification framework

The architecture we aim for is represented in Fig. 1: the central element is a visual CASE tool that allows the design of BPMN workflow diagrams and the automatic generation of LTL formulae to be verified on a given formal specification of a Web application. Since the tool produces a XML representation of the workflow, several other translations can be implemented, by simply programming new XSLT transformations.

For example, it is possible to exploit the workflow diagram to generate a browsable HTML or even JSP prototype. Another interesting transformation automatically generates Web application diagrams according to existing modeling languages for the Web. Some of these languages (e.g., WebML [2]), have been recently extended with primitives for business process management. To apply automatic verification, the Web application must be formally specified. This can be done by hand, or by implementing automatic translation. The verification itself can be achieved by

using an automatic verifier such as the one described in [7]. The formulae to be checked can be LTL rules, automatically extracted by the BPMN representation of the site.


### 3 Workflow Notations

Workflow design methods concentrate on notations capable of expressing process specifications. These notations capture activity execution constraints and special process features like pro-activity, exception handling, error compensation. In B2B Web applications, the process must be deployed on the Web, which raises novel issues due to the specific nature of Web interfaces. First, Web interfaces lead to the prevalence of hypertext-based navigation as a mean of user interaction with the process; this navigation has to be well formalized and incorporated in the very design of the process to enact, in order to guarantee correct application behavior. Second, the pull-based nature of Web applications (the HTTP protocol imposes that clients ask the server to perform some computations) lacks convenient means for interactions initiated by the server (typically known as notifications).


Processes can be pictorially represented with the Business Process Management Notation [3], which is adopted by the BPML standard, issued by the Business Process Management Initiative. The BPMN notation allows one to represent all the basic process concepts defined by the WfMC [12] model, and provides further constructs, more powerful conditional gateways, event and exception management, free combination of split/join points, and other minor extensions. BPMN events (messages, exceptions, and so on) can occur during the process execution. Gateways are process flow control elements; typical gateways include decision, splitting, merging and synchronization points. Table 1 briefly summarizes the main visual constructs provided by BPMN.

**Table 1.** BPMN main constructs


### Events



Start




End




Intermediate


### Gateways



Or gateway

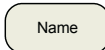


Xor gateway

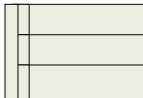


And gateway


### Activities and Flows




Activity




Pool and Lanes



Sequence flow



Message flow



Data Association

BPMN activities extend WfMC activities, as they can express various behaviors (looping execution, compensation, internal sub-process structuring, event catching, and so on). BPMN activities can be grouped into pools, and one pool contains all activities that are to be enacted by a given process participant. Within a pool, we use BPMN lanes to distinguish different user types that interact with the specific peer. The flow of the process is described by means of arrows, representing either the actual execution flow, or the flow of exchanged messages. Another type of arrows represents the association of data objects to activities; these are meant just as visual cues for the reader, and do not have an executable meaning.

## 4 BPMN Formalization Using LTL Formulae

BPMN appears to be a good and accepted notation for representing business processes. Since our target consists of verifying properties of process-based Web applications, BPMN is a good candidate as a visual representation of rules to be verified. For the BPMN formalization, we consider a significant subset of the full BPMN notation; indeed, BPMN comprises several particular symbols that are not interesting for the formalization.

The main actor in our solution is the concept of *activity*. An activity is a task to be executed, whose status is of interest. For sake of simplicity, we assume only two possible states for an activity: active and completed. In the following we adopt these abbreviations:

- $A1a = A1.status = \text{"active"};$
- $A1c = A1.status = \text{"completed"}.$

Obviously, the following holds:  $A1a \ B \ \neg A1c$ .


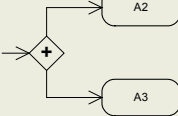
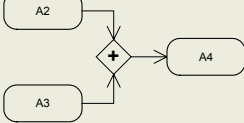
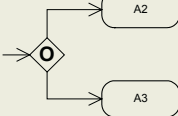
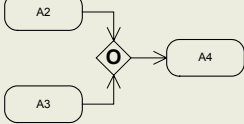
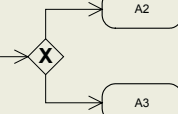
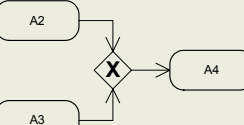
This section presents the temporal logic translation of the main BPMN visual primitives. For the translation, we do not consider a single element at time, but significant combinations of elements. In our proposal, we assume that temporal operators are connected through conjunction. This means that it's possible to translate single elements (or simple combinations) and then connect them with AND ( $\wedge$ ) connectors. Table 2 summarizes the proposed notation.

A **sequence** is a combination of two (or more) activities that can be executed only in sequential order. Its semantics can be naturally represented by the  $B$  temporal operator. The associated semantics is that activity  $A1$  must complete before activity  $A2$  can start. Because of the operator semantics, we introduce a negation on the second operand. The resulting LTL translation is:  $A1c \ B \ \neg A2a$ .

**AND Split** represents the case in which the execution flow is spawn in two (or more) parallel branches, thus enabling mandatory parallel execution of two (or more) activities. The semantics of And split can be represented by saying that both the branches must eventually be executed. Notice that we do not impose any constraint on the actual temporal parallel execution: one of the two activities may start (and finish) before the other, or vice versa, or possibly they may be executed in a real parallel enactment. The important issue here is that both of them must be executed. The resulting LTL is:  $F A2a \wedge F A3a$ .

**AND Join** represents the case in which two (or more) parallel execution flow branches merge into a single flow, after all branches are completed. The semantics is

**Table 2.** BPMN symbols translation in LTL formulae

BPMN CONCEPT	BPMN VISUAL NOTATION	TEMPORAL LOGIC
Sequence		$(A1c \text{ B } \neg A2a)$
AND split		$(F A2a \wedge F A3a)$
AND join		$(A2c \wedge A3c) \text{ B } \neg A4a$
OR split		$(F A2a \vee F A3a)$
OR join		$(A2c \vee A3c) \text{ B } \neg A4a$
XOR split		$(F A2a \text{ xor } F A3a)$
XOR join		$(A2c \text{ xor } A3c) \text{ B } \neg A4a$

represented by the fact that both A2 and A3 must complete before the next activity (A4) can start:  $(A2c \wedge A3c) \text{ B } \neg A4a$ .

**OR Split** represents the case in which the execution flow is spawn in two or more parallel branches, thus enabling possible parallel execution of two (or more) activities. Its semantics is that an arbitrary (non-empty) subset of the branches can be executed. Again, we do not impose any constraint on the actual temporal execution. The resulting LTL translation is:  $F A2a \vee F A3a$ .

**OR Join** represents the case in which two (or more) parallel execution flow branches merge into a single flow. In this case, semantics implies that it is enough that one of the two activities ends for allowing the prosecution of the flow to the next activity (A4):  $(A2c \vee A3c) \text{ B } \neg A4a$ .

**XOR Split** represents the case in which the execution flow is spawn in two or more branches, thus enabling the execution of one and only one activity among the available set. The semantics is that one and just one branch can be executed among a set of branches. The resulting LTL translation is:  $F A2a \text{ xor } F A3a$ .

**XOR Join** represents the case in which two (or more) mutually exclusive execution branches merge into a single flow. Its semantics consist in allowing the continuation of the execution once one of the branches ends:  $(A2c \text{ xor } A3c) B \rightarrow A4a$ .

Notice that explicit negation is not allowed for activities within a workflow diagram. This limitation is meant for allowing coherence with the semantics of workflow modeling, in which capability of negating the execution of tasks is not usually provided.

The above specification allows compiling a BPMN specification into an LTL formula, which can then be passed on to a verifier. The translation also points out limitations in the expressive power of BPMN. Indeed, it is clear that BPMN cannot express all LTL properties. For example, the  $X$  operator cannot be specified, and neither can negation.

## 5 A Visual Notation for Full LTL Expressive Power

Since BPMN diagrams cannot express all LTL properties, we would like to develop an extension providing a complete visual representation of Linear-time Temporal Logic. For this purpose, we extend the BPMN notation with a few other primitives. We take as the basic building block of the diagram any generic property instead of a process activity. Indeed, at this point we no longer deal explicitly with workflows, but rather with generic temporal formulae. However, the proposal presented next is completely compatible with the BPMN semantics in Section 4. A property is assumed to be a logic proposition that does not contain any temporal operator. In this sense, we suppose that a simple Boolean logic formula does not need to be visually represented. Visual aid becomes fundamental for expressing complex temporal properties.

As mentioned earlier, from the expressive power point of view, the workflow primitives fall short in two main respects relative to full LTL: using BPMN operators (and in general any workflow notation) it is not possible to specify explicit negation and the concept of “next step” in the time scale. Indeed, workflow languages do not need to provide such primitives. We cover these aspects with our extended notation.



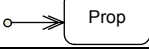
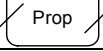
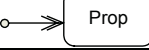
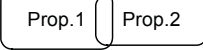




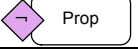

For representing generic LTL formulae we adopt the following visual elements: a *property* is represented with a rounded rectangle, which is the same symbol of activities within workflows; *parentheses*, which are essential for specifying evaluation priority in formulae, can be represented by dashed blocks surrounding properties (this choice is coherent with BPMN notation, which introduces the concept of *group* for representing grouping of activities); *Before* is represented with a simple arrow connecting two properties, thus allowing compatibility with the semantics of workflow sequences (for coherence, we impose the arrow symbol to comprise the semantics of *Before Not*); for *Next* operator we propose a symbol that recalls the concept of after/before in BPMN, and then adds the notion of “immediately” after (a double headed arrow, as depicted in Table 3); *Globally* has no direct counterpart in BPMN (although it can be simulated), therefore we propose a symbol represented by a rounded rectangle with two slashes on the sides, ideally representing the fact that the property has no time limitations; *Eventually* is a unary operator, that we represent with a simple arrow, with no starting point, similarly to the Next operator (notice that the before operator has a similar symbol, but the arrow always starts from a property

or a group); *Until* is represented by two properties that intersect on one side, to represent the fact that the first property must hold until the second one holds;

Classical Boolean operators (*And*, *Or*, *Not*, *Xor*, *Implication*) are represented by the diamond symbol of BPMN gateways: depending on the operator, the diamond contains the proper initial letter (e.g., *A* for *And*, *O* for *Or*, and so on). We decided to avoid using the symbol of “+” for *And* (like in BPMN) for coherence with the other symbols and because in Boolean logics the “+” symbol is often associated with the *Or* operator. In case of binary operators, the diamond directly attaches to the two operands. In case of unary operators (*Not*), the diamond attaches to the single property the operator applies to.

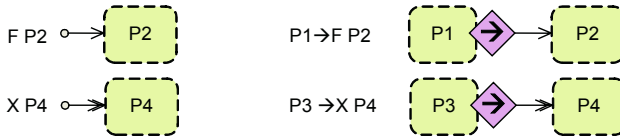
Again, we assume that temporal operators at the same level of nesting are connected through conjunction. Notice that unary temporal operators, like *Eventually F* and *Next X*, must be represented only by arrows with no starting point, while binary operators can be depicted as arrows with a starting element.

**Table 3.** Visual translation of LTL operators

OPERATOR	LTL FORMULA	VISUAL NOTATION
Property	Prop	
Before	$(\text{Prop1 } B \neg \text{Prop2})$	
Next	$X \text{ Prop}$	
Always	$G \text{ Prop}$	
Eventually	$F \text{ Prop}$	
Until	$(\text{Prop1 } U \text{ Prop2})$	
And	$(\text{Prop1 } \wedge \text{ Prop2})$	
Or	$(\text{Prop1 } \vee \text{ Prop2})$	
Xor	$(\text{Prop1 } \text{ xor } \text{ Prop2})$	
Implication	$(\text{Prop1 } \rightarrow \text{ Prop2})$	
Not	$\text{not Prop1}$	
Parenthesis	$(\text{Prop1 } \wedge \text{ Prop2})$	

A shortcut notation can be adopted for unary operators, which allows to connect a starting point of a unary temporal operator directly to a logic connector (*And*, *Or*, *Xor*, *Implication*), as represented in Fig.2. Notice also that the proper combination of sequence arrows and Boolean diamonds can produce the same effect as BPMN gateways.

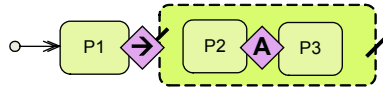




**Fig. 2.** Visual shortcuts for unary temporal operators

To illustrate the resulting diagrams, we provide some examples of visual notation corresponding to given LTL formulae.

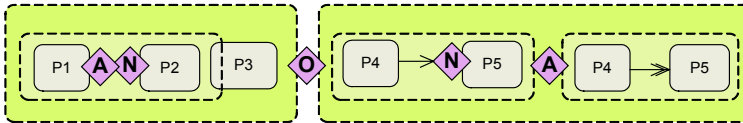
**Example 1.**  $(X P1) \rightarrow G (P2 \wedge P3)$



**Fig. 3.** Visual diagram representing the formula of Example 1

**Example 2.**  $((P1 \wedge \text{not } P2) U (P3)) \vee (P4 B P5 \wedge P6 \rightarrow X P7)$

Evidently, increasing the complexity of formulae results in increasingly complex diagrams. There is a reasonable complexity beyond which the visual notation becomes unpractical.



**Fig. 4.** Visual diagram representing the formula of Example 2

## 6 Implementation

This section presents the implementation of a prototype tool that allows to design BPMN diagrams and to automatically generate the corresponding LTL formulae. This tool has been developed to automate the generation of LTL formulae and to implement other automatic translations of BPMN diagrams. The implemented prototype allows designing workflow diagrams according to the BPMN standard. The designer can create, save and reload projects. At the moment, each project can contain only a single diagram.

The example shown in Fig. 5 is the BPMN specification of the process for the validation of an online loan request. The process takes place within a single pool, consisting of three parallel lanes, one per type of user. The process starts with an application request issued by an applicant, which is submitted for validation to a manager of the loan company. The manager may either reject it (if the application is not valid), which terminates the process, or assign it in parallel to two distinct employees for checking. After both checks are complete, the manager receives the application back and makes the final decision.

The tool allows top-down design of the application, because it provides also sub-process primitives, according to the BPMN specifications. This allows the designer to specify the workflow schema “in the large”, and then he can drill down in the design, by detailing each single activity in more specific sub-processes. This multi-level representation of the workflow, can be automatically flattened in a single level workflow schema, from which the LTL formulae can be extracted.

The user interface of the tool is organized as follows: the *main panel* of the tool consists of a board for drawing, zooming and browsing the diagrams, provided with a set of buttons that enable the user to insert the proper visual primitives; on the top-left corner, a *bird's eye view panel* always shows the complete diagram (this is particularly useful in case of big projects); if the project includes sub-processes, at any level it is possible to have the bird's eye view of any super-level; on the bottom-left corner, a *property panel* provides the description of the currently selected object; the *menu bar* allows to execute automatic transformations of the diagram (e.g., to generate LTL formulae) and to set some preferences.

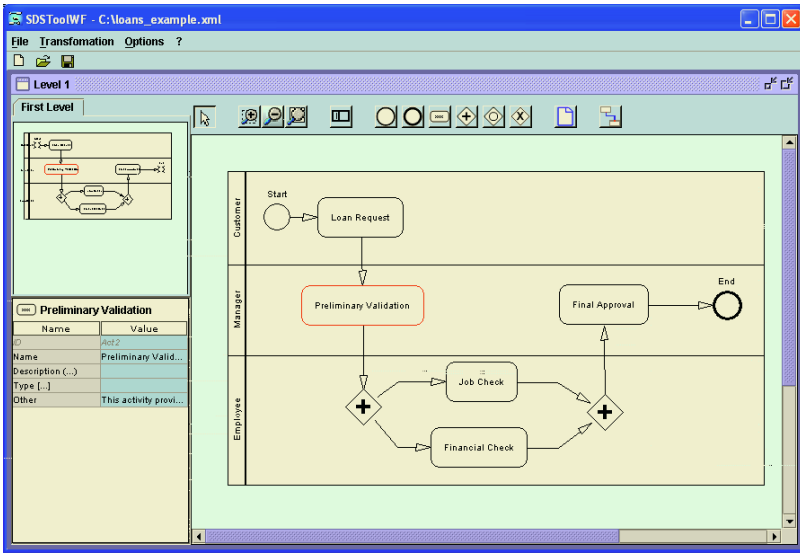


Fig. 5. CASE tool GUI for designing BPMN processes

The tool is designed to be flexible and extensible. It is able to manage user-defined properties of objects, and to dynamically add XSLT diagram transformations. The project is stored as an XML document and LTL formulae are generated using XSLT technology. Generation rules have been built based on the translation table presented in Section 4. To facilitate the translation, some assumptions have been made: gateways are considered as particular activities, thus allowing to insert them within precedence rules; in the transformation, it is enough to consider a pair of BPMN elements at time for defining the basic rules; the rest of the transformation is obtained through composition of such rules. These assumptions do not affect the generality of the transformation approach.

The LTL formula generated from the diagram shown in the picture is the following:

$$(F \text{ LoanReq}.a) \wedge (\text{LoanReq}.c \text{ } \mathbf{B} \neg \text{PreValid}.a) \wedge (\text{PreValid}.c \text{ } \mathbf{B} \neg \text{And1}) \wedge (\text{And1} \text{ } \mathbf{B} \neg (\text{JobCheck}.a \wedge \text{FinCheck}.a)) \wedge ((\text{JobCheck}.c \wedge \text{FinCheck}.c) \text{ } \mathbf{B} \neg \text{And2}) \wedge (\text{And2} \text{ } \mathbf{B} \neg \text{FinApp}.a) \wedge (F \text{ FinApp}.c).$$

Its interpretation is quite straightforward: for each process instantiation, LoanRequest will eventually be active, and it will complete before PreliminaryValidation can start. PreliminaryValidation must complete before the And split is enabled, and the And split will be evaluated before both JobCheck and FinancialCheck can start. These two activities must complete before the And join gateway, which in turn must precede the FinalApproval activity.

## 7 Conclusions

The proposed approach allows representing temporal formulae in a visual fashion. Our visual notation is inspired by workflow notations and concepts, since they appear to be the visual models that best fit the description of temporal properties. We extended such notations to yield the full expressive power of Linear-time Temporal Logic, thus enabling non-expert designers to tackle the verification of Web applications. We stress that we do not advocate the need of a new approach for verification of Web applications: traditional verification results still apply. The main contribution of this paper stands in the contribution of a visual notation for LTL formulae representation, which dramatically increase acceptance of verification approaches by the Web engineering community.

The implementation of a tool that allows to visually design models and to automatically generate LTL formulae greatly improves the usability of the approach. Future work will address semantic specification of the BPMN multi-level feature (i.e., the capability of structuring processes in sub-processes). The tool currently supports only BPMN the diagrams. The next task is to implement the complete library of symbols proposed in Section 5 for covering full LTL expressive power. Other extensions will include: an XSL transformation towards WebML diagram skeletons for helping the designer to specify the hypertext of the Web application; an XSL transformation towards HTML browsable prototypes, and a more refined automatic JSP prototype generation.

## References

1. Atzeni, P., Mecca, G., Meriardo, P.: Design and Maintenance of Data-Intensive Web Sites. EDBT 1998: 436-450.
2. Brambilla, M., Ceri, S., Comai, S., Fraternali, P., Manolescu, I.: Specification and design of workflow-driven hypertexts, Journal of Web Engineering, Vol. 1, No.1 (2002).
3. Business Process Management Language (BPML) and Notation (BPMN): <http://www.bpml.org>
4. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications, Morgan-Kaufmann, December 2002.
5. Chechik, M., Devereux, B., Easterbrook, S., Gurfinkel, A.: Multi-valued symbolic model-checking. ACM TOSEM, Volume 12, Issue 4 (October 2003), pp. 371 - 408

6. Deutsch A., Sui L. and Vianu V.: Specification and Verification of Data-driven Web Services. PODS 2004: 71-82.
7. Deutsch, A., Marcus, M., Sui, L., Vianu, V., and Zhou, D.: A Verifier for Interactive, Data-Driven Web Applications. SIGMOD 2005, Baltimore, June 13-16,2005.
8. Emerson, E.A.: Temporal and modal logic. In Leeuwen, J.V., editor, Handbook of Theoretical Computer Science, Vol. B, pages 995-1072. North-holland Pub. Co./MIT Press, 1990.
9. Muccini, H.: Software Architecture for Testing, Coordination and Views Model Checking. Ph.D. Thesis, 2002.
10. Schwabe, D., Rossi, G.: An Object Oriented Approach to Web Applications Design. TAPoS 4(4): (1998).
11. WebML Project Homepage: <http://www.webml.org>
12. Workflow Management Coalition Homepage: <http://www.wfmc.org>