IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4933

VERBUS: A Formal Model for Business Process Verification

Jesus Arias Fisteus
Telematic Engineering Department, Carlos III University of Madrid, Avda. Universidad, 30, 28911 Leganes (Madrid), Spain, jaf@it.uc3m.es

Andres Marin Lopez
Telematic Engineering Department, Carlos III University of Madrid, Avda. Universidad, 30, 28911 Leganes (Madrid), Spain, amarin@it.uc3m.es

Carlos Delgado Kloos
Telematic Engineering Department, Carlos III University of Madrid, Avda. Universidad, 30, 28911 Leganes (Madrid), Spain, cdk@it.uc3m.es

ABSTRACT

Business process management is a key issue in B2B. Different process modelling languages and workflow management tools and frameworks have appeared to aid the development, deployment and management of e-commerce solutions. Nevertheless, there is not yet a framework to compel with tasks such as guaranteeing: safety outcomes of the run of a composition of processes, the eventual execution of processes under some condition, or the soundness of a design with respect to the specification. This is the mission of formal methods, and they have been successfully applied in the fields of real-time software, hardware verification, and a growing list of application areas as the computation costs decrease. In this article we present VERBUS, a formal system for the modelling and verification of business processes. VERBUS allows a designer to specify properties for verification. The Finite State Machine theory is under the hood of VERBUS, providing the designer with a well-known and understandable approach, which offers a set of existing tools for verification or for compiling to other formats for the application of model–checkers.

INTRODUCTION

Business processes define the way in which an organization performs its business activities. From a design perspective a business process: is the container of a set of activities with a particular goal; is often carried out by a collaboration team; can overpass the bounds of an organization; can be initiated as a reaction to events external to the organization.

Modelling a business process consists of creating a conceptual system which reflects the more relevant characteristics of the real process under consideration. In [3], three different categories of purposes for business process modelling are given: *description*, *analysis*, and *execution*. The aim of our work is the analysis of processes. In [4] three different types of analysis are presented: *validation* (the system behaves as expected), *verification* (proof of correctness), *performance analysis* (resource consumption, timing and delays, throughput, etc.)

In the recent years, a number of languages have been developed for business process description, most of them adapted from the area of computer science. Most authors agree that modelling techniques can be divided in two: communication-based [5] and activity-based. Among them, activity-based techniques are the most used in current business process management (BPM) systems. Activity-based models describe a process by set of activities and their relations. Activities are the main components of a process and represent units of work. Petri Nets [4], UML activity diagrams [6,7], Role Activity Diagrams (RAD) [3], or Integrated DEFinition (IDEF) [8] are some of the languages used for activity-based modelling. Among them, only Petri Nets allow for the direct application of formal methods for process definition and analysis.

More recently, several XML languages have been developed to model inter-organizational processes. Their main objective is the modelling of loosely coupled business processes, executed by collaboration of different BPM tools in a heterogeneous and distributed environment. Many of them use web services as communication middleware. BPEL4WS, BPML, WSCI, XLANG and WSFL are some of these languages [14].

None of these languages can be formally verified directly. Although Petri nets and some other formalisms can be used for formal modelling of business processes, none of them has been applied to the verification of these XML languages. In this work we present VERBUS, a formal system that can be used as the basis for the verification of some of these languages. Business processes are defined in VERBUS by means of a formal specification, which is analyzed by verification of its requirements. The formal system is based on transition systems. Model-checking tools and refinement by transformation are the strategies of VERBUS to accomplish verification and to get to a running system.

The structure of the article is organized as follows. First the VERBUS formal model is introduced. Different techniques used in VERBUS for process verification are later presented. Then some related works and, finally, the conclusions and our intentions to future work are exposed.

THE FORMAL MODEL

This section begins with an informal description of the model to improve the understandability of the formal model.

Informal Description of the Model

The model is based on the concepts of activity–based modelling defined in [4]. Among them, entity and activity are the most important concepts in our model.

An entity represents a document, some data, or a material object that is acted upon by the process. We represent entities by the set of their relevant attributes, which are expressed by name–value pairs. The values of the attributes of all the entities of a process define the state of that process at a given moment. The process starts at an initial state and evolves changing its state. A change in the value of any attribute causes a change in the state of the process.

An activity is a unit of work of the process. In this model, each activity is composed by two or more transitions. A transition models an atomic change in the state of the process. Each transition is associated to the beginning, completion or an intermediate result of an activity. A transition is modelled by a condition-action pair. The condition is a Boolean predicate about the value of the attributes. The action is a set of assignment expressions that change the value of some attributes. If

the condition is evaluated to a true logical value for the current state, then its action can be executed, and so a new state is reached.

The execution of an activity is decomposed in a set of at least two transitions. The first of them models the beginning of the activity. Its condition can be viewed as the activity precondition (or requirements). The second models its completion. Its action represents the changes in the attributes as the result of the execution of the activity. In addition to initial and end transitions, more transitions can be used to model intermediate results in the execution of an activity.

The execution of an activity can also be non-deterministic. For a given state, the condition of two or more transitions could evaluate to true. In this case, one of them is non-deterministically selected. For example, suppose that an activity that can end either normally or with an error is modelled. Two different end transitions could be defined for it, both with the same condition and different action. One of them could represent the normal behaviour and the other the behaviour for the occurrence of an error. At execution time, only one of them would be selected.

The execution model presented in this work is inherently concurrent. At a given time, a set of activities is being executed concurrently if their start transitions have been executed, but not their end transitions. To impose the sequential execution of a group of activities, control attributes can be used. For example, suppose that activity B can be executed only after the completion of activity A. An attribute can be used to mark whether activity A has been completed. The condition of the initial transition of B might use this attribute to impose this requirement.

Formal Definition of the Model

In the following paragraphs the concepts of attribute, state, transition, path, process and functional transition are formally defined.

Let A be the finite set of attributes of a process. Each attribute has a given value in the finite set V_a. The set V is defined as $V = \bigcup_{a \in A} V_a$.

Definition 1 A state is defined as a mapping $s:A \rightarrow V$ such that $\forall a \in A$, $s(a) \in V_a$

The state is an application that associates each attribute with its value. Given a set of attributes A, the finite set of all the states is denoted by S. The process evolves from one state to the next state by *transitions*. A transition is a relation between the original state and a subsequent state, directly reached from the original state (i.e. there are no intermediate states between both states):

Definition 2 A transition is a pair (s_1,s_2) belonging to the product SxS

The set of all the transitions of a given process is a subset $T \in P(SxS)$, where P denotes the power-set of a set. This is the definition of a binary relation. Thus, the set T of all transitions can be represented as a binary relation R on the domain S. For each pair of states $s_1,s_2 \in S$, $(s_1,s_2) \in T$ is equivalent to $s_1 R s_2$.

The concept of transition allows the designer to define the dynamic behavior of a process P. Given a process in state $s \in S$ and the set $S'=\{s' \in S \ / \ sRs'\}$, "$s' \in S$' the process can evolve from the state s to the state s'. And P must evolve to a state $s' \in S'$ (i.e. whenever there are transitions with origin in the actual state of P, P must eventually evolve towards a destination state of some of the transitions). From a given state s, if there is no state $s \in S$ such that sRs', then s is a final state of P. $S_f \subset S$ denotes the subset of all the final states.

Definition 3 A path is defined as a sequence of states $e = <s_0,s_1,\ldots,s_{n-1}>$ such that $s_i R s_{i+1}$ for all $0 \leq i < n-1$.

Given a path $e = <s_0,s_1,\ldots,s_{n-1}>$, the i-th state of e is denoted as e^i. The last state of e is denoted as $last(e) = e^{n-1}$.

Definition 4 Given a set of initial states S_0, an execution is a path e such that $e^0 \in S_0$ and $final(e) \in S_f$.

According to the definition, an execution is a possible evolution of a process, from an initial to a final state, among all the possible runs (evolutions) conforming to the specification of the process. A process P is defined as the set of all the possible executions conforming to its specification:

Definition 5 Given a set of states S, a binary relation R in the domain of S and a set of initial states S_0, a process P is defined as the set of all the possible executions.

The explicit definition of transitions is natural for small-sized processes (reduced set of states, attributes and transitions). However, it is not practical for realistic processes, which have a large set of states. In VERBUS, an implicit representation of states and transitions is proposed. States can be grouped in sets such that a certain property is true for all of them. Consider a process with a group of states which share in common the property *document D has been reviewed.* It is easier to define the property than to enumerate all the states satisfying it. Following this approach, transitions can be grouped in *functional transitions*:

Definition 6 A functional transition is a function $f : S' \rightarrow S$ such that $S' \subseteq S$ and $\forall s \in S'$ sRf(s).

The functional transition f establishes a transition between each state of its domain (S') and a state of S. It is just a more compact notation to represent multiple transitions between states. This function can be defined by two predicates: a first predicate defining the function domain (the condition); and a second predicate defining the equation of the function (the action). Note that a state s can be in the domain of more than one functional transition. This models the non-determinism of the process.

The set of all the functional transitions that define a process is denoted by F. Given the set F, it can be proved that there is a binary relation R defining exactly the same transitions. So the proposed implicit representation of transitions is equivalent to the explicit representations of binary relations. The main points in introducing the implicit definition of states are: to use a compact notation to describe the processes; and to handle efficiently the complexity of specs in the verification process.

Figure 1: Fragment of a VoD example process. Each entity is a grouping of attributes. For reasons of space, only one activity is declared.

```
process VoD {
   enttype Request {
           state: enum (init, received, reviewed, responded, confirmed,
                        ready, cancelled);
           movie: abstract;
           quality: abstract;
           date: abstract;
           alternative: boolean;
           acceptable: boolean;
           accepted: boolean;
   }
   enttype NetworkResources {
           state: enum (no_reservation, provisional_reservation,
                 confirmed_reservation);
   }
   entity req: Request;
   entity network: NetworkResources;
   ...
   activity NetworkNegotiation {
           state negot: enum (init, processing, finalised);
           transition begin {
                 domain: {act.negot=init & req.state=reviewed &
   req.acceptable
                           & network.state=no_reservation}
                 action: {act.negot=processing}
           }
           transition acceptable_end {
                 domain: {act.negot=processing}
                 action: {act.negot=finalised
                           & network.state=provisional_reservation}
           }
           transition no_acceptable_end {
                 domain: {act.negot=processing}
                 action: {act.negot=finalised & network.state=no_reservation
                       & !req.acceptable}
           }
   }
   …
```

A fragment of an example business process for a Video on Demand service is presented in Fig.1. It is defined using a textual language very closed to the formal definitions above. The language is described in [9].

VERIFICATION OF BUSINESS PROCESSES

When the modeller defines the business process, it is useful for him to check that the defined process is formally correct. This is the main goal of VERBUS. The modeller specifies, in addition to the process definition, some properties that must be true for it. The system checks whether these properties are true or false. If any property is false, the system can give a counterexample. VERBUS can verify both safety and liveness properties. In this section we explain how verifications can be done in VERBUS.

It can be proven that the VERBUS formal model is equivalent to a non-deterministical finite state machine (ND-FSM). So, many existing tools can be used to perform the verification of a VERBUS process specification, provided that an automatic mapping is defined from the VERBUS language to its equivalent specification expressed in the tool's language.

To test the system we have developed two mapping tools for VERBUS business process definitions. One of them maps it into a CLIPS program. The other one maps it into a PROMELA program. To perform the verification, the generated programs have to be executed in the appropriate environments.

The CLIPS based verification is described in [9]. It has two main problems: scalability and expressiveness. It has scalability problems because it can't deal with medium-sized process definitions, because of the state explosion problem. It lacks also of expressiveness because it can only verify simple properties. It isn't expressive enough to define some types of properties that involve temporal orderings of things. For example, it doesn't allow the modeller to say that *if an affirmative response is sent to the entity A, it must exist at least one path in which sometime in the future the requested service will be prepared.*The PROMELA based verification provides a good solution to these problems, using algorithms successfully applied to other domains such as digital circuits, communication protocols and software development. It is based on *model-checking* algorithms and temporal logics [12]. Fig. 2 shows three simple example properties.

A Verification Tool Based on PROMELA Specifications

Spin is a well–known open source software model-checker that can be used for the formal verification of distributed software systems, developed at Bell Labs. Spin uses a high level language to specify system descriptions, called PROMELA. Spin has been used to find design errors in operating systems, switching systems, concurrent algorithms, etc. It can be used to verify liveness and safety properties like deadlock detection, invariants or code reachability. It can also be used as a LTL (Linear Time Logics) model-checking system. PROMELA and Spin are further described in [13].

We have developed a tool that automatically translates VERBUS process definitions to equivalent PROMELA specifications. So each process defined in VERBUS can be efficiently verified with Spin. The tool was developed in C++. It parses the VERBUS specification and generates the PROMELA specification from it. The parser was generated using the GNU tools Flex and Bison. Each element of the VERBUS specification is mapped to PROMELA following these rules:

- Attributes: mapped as global variables. Boolean attributes are represented as *Boolean* variable types. Enumerated attributes are represented as *byte*, *short* or *int* variable types. A number is assigned to each possible enumerated value of each attribute. A constant is defined with the *#define* preprocessor directive to represent this number with a word (the textual name assigned by the workflow designer to this value). This makes the PROMELA specification easier to read.
- Processes: represented as PROMELA process types. The *init* process of PROMELA starts all these processes concurrently.
- Activities: no explicit implementation in the PROMELA speci fication, because each functional transition is defined indepen dently
- Functional transitions: included in a repetition statement (*do*). In it, the domain of each functional transition is represented as a conditional statement. All the statements are mutually exclu sive: at a given moment, only one can be non–deterministically selected from all the statements that evaluate to true. There is also a default statement that ends the *do* loop when no domain statement is true. The action of each functional transition is mapped to a statement following its domain.

The correctness properties defined in the VERBUS specification are also automatically mapped to the PROMELA specification and verified with Spin:

- Invariants: properties that must be true for all the reachable states of the process. To check invariants the method recommended in [13] is followed. A monitor process type is defined and run in parallel with the other processes. The content of this process type is an assertion statement with the logic conjunction of all the defined invariants. So if an invariant is not satisfied, Spin will detect it and give the counterexample.
- Goal reachability: the modeller can establish a Boolean predicate that must be true for all the final states of the process, called *goal predicate*. If the process reaches a final state for which this predicate is false, then the goal reachability property is not satisfied. For example, this property can be used to detect deadlocks and paths ending in incorrect states. In the PROMELA specification this condition is mapped as an assertion, placed after the do statement that represents the transitions.
- Functional transition reachability: when one or more functional transitions can not be reached in any possible path of the process, there is probably a design error. The code reachability proofs of Spin can directly detect this.
- Temporal properties: the designer can check temporal proper ties expressed with LTL logic or Spin never claims, specifying them in the user interface.

Figure 2: Example properties to verify in the process presented in Figure 1.

```
invariant inv_1 {
        !req.state=responded |!req.acceptable
         | network.state= provisional_reservation
}
invariant inv_2 {
        !req.state=cancelled
        | !network.state=confirmed_reservation
}
goal goal_1 {
        req.state=cancelled
        | req.state=ready
}
.
```

RELATED WORK

Many modelling systems had been developed for workflow and business processes, but few of them allow the modeller to formally verify properties to check the correctness of the defined processes. From the latter, almost all are based on Petri Nets.

Petri Nets is the formalism used in many commercial BPM systems and research prototypes. In [4] the most relevant publications in this field are cited, and the usage of Petri nets for workflow modelling is explained and justified. It explains how definition and analysis of business processes can be done with this formalism. Petri Nets is a powerful tool for the definition of complex processes, and provides suitable formal methods for correctness verification of processes. In [7] formal semantics are added to UML activity diagrams to allow the verification of business processes defined with such formalism.

CONCLUSIONS AND FUTURE WORK

VERBUS is a new and powerful environment for the verification of business processes. It supports the verification of intra–organizational and inter–organizational business processes. It uses a formalism based on transition systems, and is equivalent to FSMs. So a lot of tools and theoretical results developed for these formalisms and their variants can be applied to business processes. The VERBUS mission is to connect high level business process definition languages to formal verification frameworks. The designer would never define processes in VERBUS directly: he would define them using other graphical languages. Then the specification could be automatically transformed to a VERBUS specification. Its main advantages are:

- It is based on a very general formal model (FSMs). So, many business process definition languages can be transformed to an equivalent VERBUS specification.
- Its specifications can be automatically transformed to other formal systems. So other verification tools can easily be used.
- The isolated specification of the requirements of each activity makes easier their reutilization for other processes.

VERBUS specifications can be verified using different formalisms. Particularly, we have developed tools that translate these specifications to CLIPS and PROMELA programs. The PROMELA version is verified using Spin. It is scalable, efficient, and can check more expressive properties defined with LTL. To complete the VERBUS system, we are developing an automatic translator from BPEL4WS process specifications to VERBUS specifications. BPEL4WS [15] is a business process definition language that models inter-organizational processes based on web services.

ACKNOWLEDGMENTS

This work is partially supported by the Spanish Science and Technology Ministry, in the project TIC2003-07208 “Infoflex”.

REFERENCES

[1] Heitmeyer, C., Mandrioli, D., eds.: Formal Methods for Real-Time Computing. Trends in Software. John Wiley Sons (1996)

[2] Clarke, E., Grumberg, O., Hiraishi, H., Jha, S., Long, D., McMillan, K., Ness, L.: Verification of the futurebus+ cache coherence protocol. Formal Methods in System Design **6** (1995) 217–232

[3] Ould, M.A.: Business processes: modelling and analysis for re–engineering and improvement. John Wiley & Sons (1995)

[4] Aalst, W.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8** (1998) 21–66

[5] Winograd, T., Flores, R.: Understanding Computers and Cognition. Addison–Wesley (1987)

[6] Dumas, M., Hofstede, A.H.M.t.: UML Activity Diagrams as a Workflow Specification Language. In: Proc. of the International Conference on the Unified Modeling Language (UML), Toronto, Canada, Springer Verlag (2001)

[7] Eshuis, R.: Semantics and Verification of UML Activity Diagrams for Workflow Modelling. PhD thesis, University of Twente (2002)

[8] Hanrahan, R.P.: The IDEF process modeling methodology. Crosstalk–The Journal of Defence Software Engineering **8** (1995)

[9] Fisteus, J.A., Delgado, C., Marín, A.: Modelo formal para la verificación de procesos de negocio: aplicación a un servicio de VoD. Proceedings of the III Congreso Iberoamericano de Telemática, Montevideo, Uruguay (2003)

[10] Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. QUT Technical report, FIT-TR-2002-02, Queensland University of Technology, Brisbane (2002) (Also see http: //www.tm.tue.nl/it/research/patterns.).

[11] Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison–Wesley (1979)

[12] Clarke, E.M., Grumberg, O., Long, D.: Model Checking. In Broy, M., ed.: Deductive Program Design. Volume 152 of NATO ASI Series. Springer (1996) 305–349

[13] Holzmann, G.J.: The model checker spin. IEEE Transactions on Software Engineering **23** (1997) 279–295

[14] Aissi, S., Malu, P., Srinivasan, K.: E-business process modeling: the next big step. IEEE Computer **35** (2002) 55–62

[15] Andrews, T., Curbera, F., Dholakia, H., et al. Business Process Execution Language for Web Services. Version 1.1 Specification.