A Hybrid Approach for Business Process Verification

Bing Li and Junichi Iijima

Graduate School of Decision Science and Technology, Tokyo Institute of Technology
W9-66, 2-12-1 Ookayama, Meguro, Tokyo, Japan {li.b.ab, iijima.j.aa }@m.titech.ac.jp

Abstract. Business Process Verification (BPV) works as one of the important functions in the emerging Business Process Management Systems. Current proposed approaches are not yet well applied because of the gap between formal models defined in the academia and informal models used in the industry. This paper attempts to propose a hybrid approach to solve this problem. XPDL will be used to describe business processes and Situation Calculus will be employed as the formalism to perform the function of BPV. A typical order fulfillment process is exemplified to illustrate the approach and the demonstration system implements the automatic transformation from the XPDL-defined process and performs the logical verification.

Keywords: *Business process verification, XPDL, Situation calculus*

l. INTRODUCTION

As part of modern enterprise information systems, Business Process Management Systems (BPMS) are increasingly important and receive greater consideration from the enterprise's executives and IT engineers. BPMS can be defined as a generic software system that is driven by explicit process designs to enact and manage operational business processes from the perspective of IT system engineers [1]. Business process design is important in the emerging BPMS.

Many previous and current research efforts are related to business process design, also called workflow modeling or business process modeling. These approaches can be classified into two categories. Applications of formal methods in business process modeling fall in the category of formal approaches, which usually employ mathematical logic [2-4]. The obvious strength of these approaches resides in the precise and inferable process model that can be verified mathematically and automatically. But since these formal approaches emphasize mathematical notation and calculi, they are not yet well applied to the BPM industry. On the other hand, informal approaches are more supported by BPMS vendors. They usually define a process in graphical or text-based languages. Then the defined process is simulated and tested to uncover potential errors that have been existent in the design phase. The merit of these informal approaches is their friendliness to general users. But the function of business process verification is obviously insufficient and detection of design errors is possibly postponed to the simulation phase or even to the execution phase.

A hybrid approach integrating both informal and formal approaches in business process design, promises to combine the aforementioned separate strengths in BPMS. This paper attempts to elucidate such a hybrid approach for business process verification (BPV), which is especially important in dynamically designing business processes. A typical order fulfillment process will be used to explain the approach and implementation of model transformation from the XPDL-defined process. A general explanation and the detailed underlying formalization can be found in Li et al. [5] and [6].

2. EXAMPLE

2.l Order Fulfillment Process

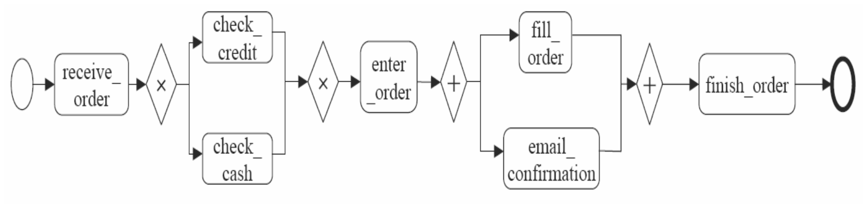


Figure 1. Order Fulfillment Process (BPMN)

As shown in Figure 1, a simple but typical example – order fulfillment process – is used to explain the approach in this paper. This process can include the five basic workflow patterns found in Havey [7]. For convenience, in this paper these workflow patterns will be referred to as XOR-Split, XOR-Join, AND-Split, And-Join and Sequence. To avoid displaying the whole lengthy process definition in XML syntax, Business Process Modeling Notation (BPMN) [8] is used to illustrate the process graphically and intuitively. From this BPMN-defined process model, the constituent activities and the transition routing can be clearly shown. BPMN can undoubtedly provide the communication convenience to some extent. However, it can not precisely represent a process model or allow for easy analysis. Therefore we prefer to use an XML-syntaxed language in this research.

2.2 XML Process Definition Language (XPDL)

XPDL[9] is an industrial standard which is supported by many BPMS developers and vendors. This approach selects XPDL as the source process model for its analyzability in XML syntax. XPDL focuses on the business logic and can specify transition relations in business processes. Constituents in a process are represented by using the concepts such as *Workflow Process*, *Activity*, *Transition* and so on. By employing this process specification standard, the approach will make it easy to integrate the industrial efforts and put them into practice. For example, the XPDL specification related to the activity "check_credit" is shown as follows. Referring to Figure 1 helps to understand the XML-based specification intuitively. The activity can be referred by using the id of "check_credit"; the activity will be implemented by an application, which can be a software application or another process; the input parameters are "CardNo" and "Rate" that refer to the NO. Of the credit card and the rating of the credit; the performer information can be provided. The control flows are represented by transitions. There are one incoming transition – from "xor_split" to "check_credit" and one outgoing transition – from "check_credit" to "xor_join"; the transition conditions can be represented by using equations or other mathematical expressions.

```
…
<Activity Id="check_credit">
   <Implementation>
            <Tool Id="check_credit_app" Type="APPLICATION">
                     <ActualParameters>
                              <ActualParameter>CardNo</ActualParameter>
                              <ActualParameter>Rate</ActualParameter>
                     </ActualParameters>
            </Tool>
   </Implementation>
   <Performer>orderProcessor</Performer>
   <ExtendedAttributes/>
</Activity>
…
<Transition From="xor_split" Id="order_fulfillment_tra2" To="check_credit">
   <Condition Type="CONDITION">
            PayWay= ="credit" CreditStatus= ="none"
   </Condition>
</Transition>
…
<Transition From="check_credit" Id="tra3" To="xor_join">
   <Condition Type="CONDITION">
            CreditStatus = = "valid"
   </Condition>
</Transition>
…
```

The above XML script specifies the activity from the perspectives of input parameters, performers and transitions. In particularly, transition relations construct the control flow of a business process and related errors can lead to deadlock or unreachable activities. This paper concentrates on the control flow perspective, but the approach is possibly applied to verify other aspects of business processes such as global constraints [2].

In XPDL, the complex transitions can be represented by routing activities that correspond to the gateways in BPMN. In the above example, the "xor_split" and “xor_join” are routing activities. A transition has some conditions and only when these conditions are satisfied, the transition can happen.

In respect to transition conditions, the repressiveness of the current XPDL specification is not so robust. This problem can be overcome by clearly defining the format of conditions or directly using some XML-syntaxed rule languages.

3. FORMAL VERIFICATION

3.l Motivation of Formal Verification

Formal verification is necessary in the emerging BPMS[10]. Expanding and fast changing business needs require that business processes should be designed and deployed quickly. Human-designed processes are prone to containing potential errors or bugs, which may increase development time and cost. But formal verification of a business process before execution can greatly reduce errors in the design phase.

In detail, formal verification can bring the following benefits. First, it can remove any ambiguity from a business process and make it more precise. Formal verification will employ a formal language, which is usually a mathematical logic. Based on such a formal language, business processes can be specified in a precise and concise way. Second, this formal process specification will enable inference functions, including automatic verification, process analysis, etc.

But a gap exists between the industrial standard process description language, such as XPDL, and formal languages that are used in the academic research. This is why the function of verification is still not sufficient in BPMS products. This hybrid approach attempts to bridge this gap between XPDL and a formal language – Situation Calculus. This strategy is also meaningful to other BPM languages such as Business Process Execution Language (BPEL).

3.2 Formalism of Situation Calculus

Situation Calculus was first introduced by John McCarthy and later extended by Ray Reiter. Much research work has been done in this formalism and it has become a formal language to model dynamical domains.

Situation Calculus has strength of reasoning about actions. This formalism can be applied to business process modeling including verification [4].The semantic transformation from XPDL to Situation Calculus seems intuitive and uncomplicated. Furthermore Situation Calculus is extensible to include some dynamic features such as concurrency and reactiveness. Some basic concepts including action, situation and fluent will briefly introduced. Detailed explanations can be found in Reiter [11] and Brachman et al. [12].

Actions are represented by action functions that consist of functional symbols and corresponding arguments. Situations are world histories represented by the sequence of actions. Fluents are functions and predicates that are dependent on the situation, which can represent the status and changes of the modeling world.

A domain model in Situation Calculus mostly consists of actions. An action is specified by precondition and successor state axioms. These axioms will be constructed by action functions, situations and fluents. Truth values of the fluents in these axioms will separately ensure executability of the action and satisfiability of the successor states.

The underlying concept is that the execution of an action will change the world states by making the related fluents become true; thus the new world state may satisfy the precondition of another action; then, this will result in the execution of an action sequence, that is, a situation starting from the initial world state.

3.3 Transformation from XPDL

XPDL is in XML syntax and has no obvious relationship with formal languages, which make it hard to be verified directly. Formal languages usually enable reasoning, including automatic verification, thanks to the underlying formal semantics. Thus transformation from XPDL is meaningful and Situation Calculus is selected for the strength explained in the section above. The specification in Situation Calculus will provide a precise and inferable process model for future analysis.

This research focuses on the control flow perspective of a business process and the transformation of transition relations in XPDL to Situation Calculus is most part of our work. As explained in Section 2.2 (XPDL) and Section 3.2 (Situation Calculus), the activities in XPDL correspond to the actions in Situation Calculus; the parameters correspond to the arguments in the action functions. Thus the obvious gap lies between the transition conditions in XPDL and the precondition and successor state axioms in Situation Calculus.

To bridge the gap between XPDL and Situation Calculus, we devise XML Situation-calculus Specification Language (XSSL), which attempts to represent some concepts in Situation Calculus by using XML syntax. In this format, the transformation will be convenient to introduce. Moreover, XSSL will enable the separation of activity specification from process specification, which can improve reusability of some common activities or processes.

Furthermore, it potentially improves the usage of Situation Calculus with more extension work to improve the expressiveness of XSSL. The key is to define XSSL more independently from XPDL, and represent more concepts of Situation Calculus in XML syntax.

The following XML script specifies the action of “check_credit”, which actually expresses the elements of XSSL. The XPDL-defined process specification can be automatically transformed into the XSSL-defined one. From the XSSL script, the important element is “Action”, which corresponds to the “Activity” in XPDL. An action is represented with its arguments, preconditions and postconditions. These concepts can be directly mapped into the formalism of Situation Calculus.

```
…
  <Action Id="check_credit">
          <args>
```

```
                <arg>CardNo</arg>
                <arg>Rate</arg>
        </args>
        <preconditions>
                <precondition>OrderStatus= ="received"</precondition>
                <precondition>PayWay= ="credit"</precondition>
                <precondition>CreditStatus= ="none"</precondition>
        </preconditions>
        <postconditions>
                <postcondition>CreditStatus= ="valid"</postcondition>
                <postcondition>OrderStatus= ="checked"</postcondition>
        </postconditions>
</Action>
…
```

When comparing this XSSL-defined activity with the XPDL-defined one, the main difference can be found to be in the transformation from the transition conditions in XPDL to the preconditions and postconditions in XSSL. This transformation is implemented based on the formal definition in Li et al. [6]. The defined mappings process different types of routing activities and calculate the preconditions and postconditions.

3.4 Logic Based Verification

The formalism of Situation Calculus can be implemented by a Prolog Interpreter [11]. Similarly, the logical process model – a formal specification in Situation Calculus – can be implemented by Prolog programs. Thus XSSL-defined process specification can be transformed into a Prolog format, which is an inferable model that can to be verified automatically. The following Prolog script specifies the action of "check_credit" based on the formalism of Situation Calculus.

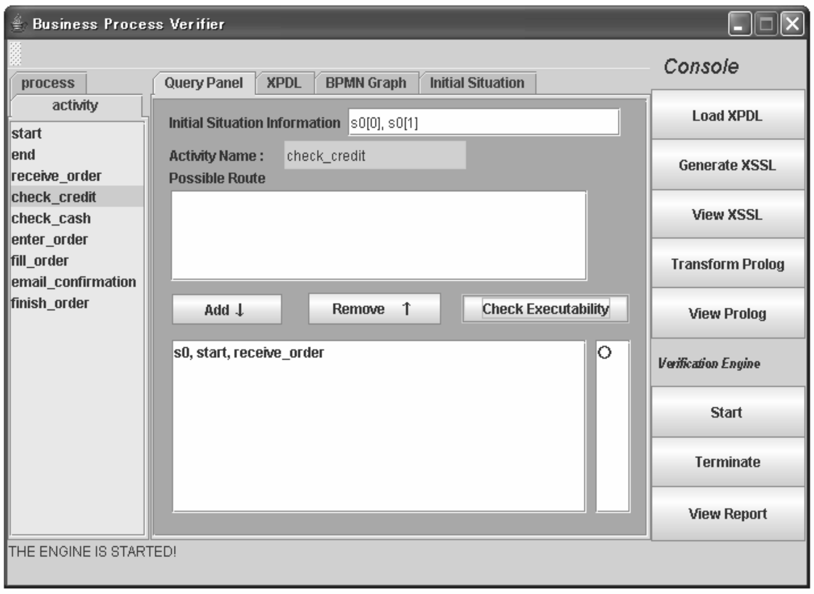
```
…
poss(check_credit(PID,CardNo,Rate),S):-
    order_status(PID,received,S),
    pay_way(PID,credit),
    credit_status(PID,none,S),
    card_no(PID,CardNo),
    rate(PID,Rate).

credit_status(PID,valid,do(A,S)):-
    A=check_credit(PID,CardNo,Rate);
    credit_status(PID,valid,S).

order_status(PID,checked,do(A,S)):-
    A=check_credit(PID,CardNo,Rate);
    order_status(PID,checked,S),
    not A=enter_order(PID,OrderInfo).
…
```

The above Prolog-defined process specification can be automatically generated from the XSSL-defined one. Moreover, some extra processing work should be done such as the introduction of the process id (PID), which enables the process concurrency and instances, and recovery of data relations. This kind of information is expressed in XPDL and can also be extracted into XSSL. It will be our extension work to study on how to represent extra information such as data relation in XSSL while keeping the independence of XSSL from XPDL.

In order to improve the performance of verification in Prolog, some extra processing work is introduced. For example, backtracking on situations will lead to memory overflow from our development experience. To solve this problem, situations are constructed from the transition routing information in XPDL. That is, the possible routes can be extracted from XPDL, enabling the construction of action sequences – situations.

4. DEMONSTRATION OF THE BPV SYSTEM

The interface of the implemented demonstration system is shown in Figure 2.



Initial Situation Information s0[0], s0[1]

Activity Name : check_credit

start
end
receive_order
check_credit
check_cash
enter_order
fill_order
email_confirmation
finish_order

s0, start, receive_order

THE ENGINE IS STARTED!

Figure 2. User Interface of the Demonstration System

A business process is defined in XPDL. This XPDL-defined business process can be defined with the aid of some XML editors or directly transformed from some graphical process model such as a BPMN-defined process one, which is currently not the focus of this research.

Load XPDL will parse the XPDL file and show the constituent activities in the left panel and other related information such as initial situations for testing. **Generate XSSL** will automatically generate the XSSL file from the XPDL file, and **Transform Prolog** will automatically transform the XSSL file into the Prolog file. These Prolog files will finally be used to build up the knowledge base for the background Prolog engine to make verification – to check the queries from the users.

The system is currently implemented at the activity level, that is, the whole process is verified after checking each activity involved. E.g., to check the activity of "check_credit", there is only one possible route according to the XPDL definition (referring to Figure 1). First, select it to construct the situation to be verified and also set the initial situation or use the default settings (**Add ↓**); second, **Start** will start the prolog engine and build up the related knowledge base; **Check Executability** will query this engine and show the result as "○" for success and "×" for failure. The current result show that the route will succeed under the default initial situation settings – that is, there is an order paid by credit card in the initial situation; and after executing the activity of "receive_order", the activity of "check_credit" can be executed.

The successful result shows that the checked activity is executable in a certain situation. After each activity involved in a process is verified, the whole process is actually ensured to be executable. It is direct to make the whole process verification if we encapsulate the checking for each involved activity and just check the last activity in the transition route. When there is an error, it is necessary to backtrack and find where the problem occurs – that is, the transition condition can not be satisfied.

The verification employs action reasoning in Situation Calculus, which enables automatic verification at the semantic level. The precondition of each activity is verified to ensure that there is no deadlock in the process. The successor state condition interconnects the activities and represents the state changes in the process, thus making the whole process verification possible.

5. CONCLUSIONS AND FUTURE WORK

In this paper we proposed a hybrid approach for business process verification and explained it focusing on the model transformation from an XPDL-defined process. The underlying formalization was explained in Li et al. [6] that provided the theoretical foundation for this paper. The implementation of the prototype system and the internal automatic transformation demonstrated the feasibility of the approach.

This hybrid approach integrates the informal language – XPDL, and the formal language – Situation Calculus. By linking them to cooperate in business process verification, we can obtain some meaningful results. Practicability and robustness are two direct benefits. Besides, the formalized process specification is more precise and becomes inferable, enabling more potential analysis of business processes.

Much work still needs to be done in the future research. Only some concepts in XPDL are currently mapped to Situation Calculus. In order to put this approach into large-scale industrial application, some further extension of the transformation should be done. Furthermore, this approach is currently only applied to business process verification and it could be also used to dynamically aid process design such as providing some recommendation for process composition.

REFERENCES

1. M. Weske, W.M.P. Van Der Aalst, and H.M.W. Verbeek, Advances in business process management, *Data & Knowledge Engineering.* Volume 50, pp.1-8, (2004).
2. S. Mukherjee, H. Davulcu, M. Kifer, P. Senkul, and G. Yang, Logic Based Approaches to Workflow Modeling and Verification, *Logics for Emerging Applications of Databases* (Springer, 2003).
3. G.K. Janssens, J. Verelst, and B. Weyn, Techniques for Modeling Workflows and Their Support of Reuse, in *Business Process Management*, LNCS1806 (Springer, 2000), pp.1-15.
4. M. Koubarakisa, and D. Plexousakis, A formal framework for business process modeling and design, *Information Systems.* Volume 27, (2002), pp.299-319.
5. B. Li, and J. Iijima, Bridging The Gap Between XPDL And Situation Calculus: A Hybrid Approach For Business Process Verification, in *Proc. of the 5th International Workshop on Modeling, Simulation, Verification and Validation of Enterprise Information Systems – MSVVEIS 2007* (INSTICC, 2007), pp. 151-156.
6. B. Li, and J. Iijima, Formal Verification of XPDL-based Business Process Definition, *The International Journal of Business Process Integration and Management* (submitted, 2007).
7. M. Havey, *Essential Business Process Modeling*, 1st edition (O'Reilly: 2005).
8. OMG, *Business Process Modeling Notation Specification* (2006).
9. WfMC, *Process Definition Interface – XML Process Definition Language*, Version 2.0 (2005).
10. W.M.P. Van Der Aalst, and A.H.M. Ter Hofsede, Verification of Workflow Task Structures: A Petri-Net-Based Approach, *Information Systems*. Volume 25, Number 1, pp.43-69, (2000).
11. R. Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems* (MIT Press: Massachusetts, 2001).
12. R. Brachman and H. Levesque, *Knowledge Representation and Reasoning* (Morgan Kaufmann, 2004).