Architecture on a Hybrid Business Process Design and Verification System

Bing Li and Junichi Iijima
Department of Industrial Engineering and Management,
Graduate School of Decision Science and Technology,
Tokyo Institute of Technology, Japan
{li.b.ab, iijima.j.aa}@m.titech.ac.jp

Abstract—Recently Business Process Management System (BPMS) has become one of the hottest research topics related to enterprise information systems. Based on some requirements in the emerging BPMS, this paper proposes an original architecture on a hybrid Business Process Design and Verification System (BPDVS). This architecture attempts to build a system that integrates the informal and formal approaches in the BPM related fields. In this way it can obtain practicability and robustness, which are presently strengths separately from the informal and formal approaches. The industrial standard – XPDL – is used as the informal process description language while a logic language – Situation Calculus – is employed as the underlying formalism to precisely specify a process. The hybrid BPDVS will integrate these two different process models and attempts to provide a more integrative and dynamic design environment for business processes.

Keywords— BPDVS; Situation Calculus; XPDL

I. INTRODUCTION

Business Process Management System (BPMS) is increasingly important in enterprise information systems. It explicitly supports the design, enactment and monitoring of business processes. Despite of the increasing importance of BPMS, the related technologies are still in infancy. Currently there are three outstanding features [1, 2].

Standardization. Now standardization in BPM is driven by the maturing software industry and is greatly corresponding to the needs from enterprise information system integration. There exist several competing standards emerging and maturing in the BPM industry, which are competitive and integrative [9, 10, 11]. Standardization has reached the consensus in the BPM industry and will greatly improve the integration between different BPMS. Meanwhile, one standard alone may not provide perfect solutions and collaboration or transformation between standards is also necessary.

Service Oriented Architecture (SOA). Web services have shown the strength of enabling the flexibility of business processes [12]. SOA puts more emphasis on using software services to support the requirements of business processes. Service orchestration and choreography have emerged as the important issues in service-oriented applications. Some methods that enable automatic service composition are considered in order to responding to the increasing number of available services within and across enterprises. In the more developed ubiquitous environment, this service oriented computing model will make the business information system more flexible and satisfy the customers’ need in great extent.

Formal Methods. Business processes should be quickly developed or improved so as to adapt to the ever-changing business environment. This requires that BPMS should put more emphasis on process design and ensure correctness of business processes. The potential errors could be detected in the design phase, not in the simulation or execution phase. Application of formal methods will solve these problems. Besides, the process specification in a formal language is more precise and explicitly defines business logic, which can remove unambiguosity [1, 3, 4]. The function of business process verification is increasingly important and application of formal methods will be helpful to make BPMS reach higher degree of robustness.

II. RELATED WORK

There are some existing research fields related to business process design and verification, which are usually titled business process modeling or workflow verification. Generally these approaches can be divided into two categories – informal and formal approaches according to their obviousness of employing mathematics, especially mathematical logics.

The informal approaches usually lack the obvious support from mathematical theories. They design business processes with a graphical or text-based language and make some simple simulation to test the defined model. These approaches are driven by the industry and have gained some supports and application in the business systems. They are easily accessible to the general business users when there is a graphical interface. But only some syntax errors can be detected, which depends on the performance of simulation engines. Detection of design errors is postponed to the testing or even the execution phase of business processes. This will obviously increase the cost and risk for the development of BPMS.

The formal approaches are proposed by the academia and application of formal methods is the distinguishing feature [1, 3, 4, 5]. Benefiting from the underlying formalism, these approaches can verify process models mathematically and
reach higher degree of robustness. The potential reasoning capability such as automatic process composition can also be implemented since these methods are usually based on a specific logic language that inherently enables the logical inference. Application of formal methods in BPMS will enable intelligent functions and improve the degree of automation. But these approaches cannot be understood well without some background knowledge since they are more involved in the mathematical notation and logic calculus.

Although the current BPMS applies these approaches in some different extent, some problems are still not solved well. Firstly, although business process verification (BPV) is obviously important to decrease the development cost by detecting errors in the design phase, the function of BPV is not sufficient in the current products of BPMS. Only some simple errors can be detected in the current tools. But verification in a higher degree is necessary. For example, checking business processes along with the business logic is more meaningful when designing business processes in an enterprise with many related processes interconnected. Thus semantic check is required to integrate the enterprise-wide processes correctly. Furthermore, verification at the semantic level is potentially important in the networked environment where most business processes are encapsulated as web services and the semantic verification can ensure the correctness and enable the automatic process composition, which will be in great need when there exist great numbers of online services or processes to satisfy various customers’ needs.

Secondly, there exists a gap between informal and formal approaches [6, 7]. These two kinds of approaches have their separate strengths. The informal approaches are driven by industrial application and are close to the business world while the formal approaches are more mathematical and enable higher degree of robustness and automation. This gap is the reason why the formal approaches are not well applied to the BPM industry in a large scale. To bridge the gap between them and make integration will result in obtaining both practicability and robustness.

To solve the above problems, we propose an original architecture to integrate the formal and informal approaches. This paper focuses on the function of business process design and verification in BPMS, which is an obvious field that will benefit from application of formal methods. The formalism of Situation Calculus is employed for its strength of action reasoning and modeling capability in dynamical domains. In the following sections, the architecture of a hybrid business process design and verification system (BPDVS) will be proposed; then the implementation about this system will be explained in detail.

### III. Architecture on a Hybrid BPDVS

The architecture is illustrated in Fig. 1, which can be separated into two layers: Infrastructure Layer and Application Layer. The Infrastructure Layer consists of process base, action base, verification engine and its related knowledge base. The Application Layer includes designing and verification components in BPDVS. This architecture can possibly be extended to include other components in BPMS such as process monitor and control.

#### A. Application Layer

The application layer consists of two components: the Designer and Verifier. The system users – business process analyzers or designers will use these two components to design and verify processes.

The **Designer** is a process editor that can define or modify business processes expressed in the graphical or text based languages. BPMN, BPEL and XPDL can be used as standard languages to specify business processes while they emphasize different aspects of processes. The preferred way is to illustrate a business process graphically and store it in the XML syntax.

BPMN [9] is the most attractive standard in the current BPM industry. It attempts to provide a uniform format to represent a business process to the different users, including business analysts and IT engineers. Thus the communication gap will be removed and the various systems will be easy to integrate. BPEL [10] focuses on the execution of processes encapsulated as web services. It is more related to the invocation and implementation of business processes. XPDL [11] is the process description standard promoted by WfMC and concentrates on the specification about business logic in a process.

![Figure 1. Architecture on a hybrid Business Process Design and Verification System](image-url)
The **Verifier** is to perform the function of business process verification. Business processes defined in BPMN, XPDL or BPEL will be formally verified. Situation Calculus is employed to form the mechanism of BPV in this research.

Situation Calculus is usually considered as a dialect of First Order Logic (FOL) although it can also include some second-order features. This formalism is applied to dynamical domain modeling. The most important concepts of Situation Calculus are situations and actions. A situation is a world history represented by the sequence of actions. An action is specified by preconditions and successor state axioms, based on which the reasoning about actions can be performed [8].

There are several reasons why Situation Calculus is selected as the underlying logical language in this research. It has the strength of reasoning about actions; the semantic transformation from the informal process models such as XPDL is intuitive and uncomplicated. Furthermore, some former work of applying Situation Calculus in process modeling has been done [4]. The theories of Situation Calculus have been extended to include dynamical features such as concurrency and reactivenss and these theories will enable processing various perspectives of a business process.

**B. Infrastructure Layer**

The Infrastructure Layer includes Process Base, Action Base, Verification Engine and its Knowledge Base.

**Process Base** includes process definitions, which can be stored in various formats. These formats will be the standard description languages that are supported by the industrial vendors. In this way the process base can be shared and reused by various BPMS. The maturing industrial standards such as BPMN, BPEL and XPDL will be used. BPMN provides a graphical process model with a friendly interface to the general users. But it is not convenient to be analyzed. To keep high reusability, process models should be analyzable and thus BPMN should be complemented with an analyzable format such as in XML syntax. That is why BPEL or XPDL is also necessary in the Process Base. These models can collaborate to provide an integrative solution for business process modeling.

**Action Base** consists of specifications about primitive activities or tasks in a process. There are two direct merits to build it up. One is to separate abstract activity specification from process definition. Usually to redesign a process will not affect general activity specification, and the action base can improve reusability of activities. The other merit is to make a process model in action base work as an intermediate model between the informal standard process model (e.g., XPDL) and the inferable logical process model (e.g., Prolog). Furthermore, XML Situation-calculus Specification Language (XSSL) is devised in this research [6, 7]. XSSL will extract necessary information from XPDL and restore in a compact and convenient format for transformation to Prolog files based on Situation Calculus. Further definition and development about XSSL will potentially promote the usage of Situation Calculus for reasoning in service-oriented applications. Then XSSL can be used to specify services in a compact and logical form that focuses on the service interface specification. Situation Calculus can be employed to automatically compose integrative services.

The **Verification Engine** is used to verify business process definitions. The reasoning mechanism is from Situation Calculus and the logic programming language – Prolog, provides the implementation. The used engine is from the constraint programming platform – ECLiPSe. This engine will compile the Prolog files and build up the knowledge base for querying. The business process verification is made by querying the engine manually or automatically.

The **Knowledge Base** consists of Prolog files that specify business processes in a logical syntax. Most parts of this knowledge base are specifications about actions with preconditions and successor state axioms. Moreover, primitive action declaration and initial situations are necessary to enable reasoning in Situation Calculus.

**C. Feature of Hybridness**

According to the degree of obviousness in applying mathematical theories, this system includes informal and formal parts. The front ends in the face of users are the informal industrial standards while the background includes the formal logical language that enables automatic verification and composition. From the human and machine processing, this architecture attempts to be friendly to human, such as business process designer and analysts, and to machine, such as analyzable and reasonable by employing Situation Calculus. The hybridness makes this approach meaningful and promising in enterprise computing with higher degree of intelligence.

IV. IMPLEMENTATION OF BPDVS

A prototype of BPDVS is implemented to demonstrate feasibility of the architecture. This section explains some important issues related to the implementation and gives a system illustration.

**A. Concepts of Model Transformation**

The dashed line in Fig. 1 shows the model transformation in the prototype system. Several standard process languages can be used in this architecture and XPDL is selected as the specification format in this research.

By comparing with BPMN, XPDL is in XML syntax, which is compliant with the trend of data storage and service oriented computing. It is easy to be analyzed and transmitted. By comparing with BPEL, XPDL concentrates on the business process logic and is independent from the system implementation. Thus this research focuses on the XPDL representation of business processes. But with the aid of some mapping tools between BPMN and XPDL, this system can be applied to processes defined in BPMN. The strategy of this approach is also applicable to BPEL.

The XSSL files in Action Base are automatically transformed from XPDL. This transformation is formally defined in [6]. XSSL is inspired from Situation Calculus while attempting to encapsulate the underlying logical calculus to the general users.
Prolog files are automatically transformed from XSSL and are used to build up the knowledge base for the verification engine. The Prolog-defined process models are inferable for the logical format that enables the automatic verification. These files can also be accessed by the process modelers who are familiar with logic programming and Situation Calculus. But the general users need not concern about them since they could be automatically generated by the system.

Here we also make some explanation about Golog. A Golog interpreter in Prolog [8] is used when we verify the business process definition. Golog is based on Situation Calculus and extends to differentiate primitive actions and complex actions. It is obviously useful when we consider the granularity of processes. The current focus is on the primitive actions but it is extensible to include the complex actions.

B. Process Verification

XPDL [11] can specify business processes in some aspects. In the current stage of this research, the control flow perspective is concentrated. The control flow, i.e., transition relation described in XPDL, will be finally specified in the Prolog syntax that implements Situation Calculus.

By querying the verification engine with the related knowledge base compiled from the Prolog files, the precondition and successor state axioms of each action can be checked. The result will show if the definition is correct or not. If there is something that makes deadlock or conflicts with the predicted results, the process definition could have errors and the related action could be detected.

1) Precondition Axiom

Satisfiedness of the precondition axiom of an action in Situation Calculus will ensure that the action is executable, that is, the transition route to this action is free from deadlock.

2) Successor State Axiom

From the specification format of Situation Calculus, the satisfiedness of the successor state axioms of an action will be automatically ensured if the precondition axiom of this action could be satisfied. The obvious verification is not needed. But the successor state axioms of an action will reflect the effect changes caused by the execution of this action. Thus the successor states of an action will (partially) be the precondition of the successive action and such precondition will be verified.

3) Effective Transition Route

In order to avoid the memory overflow and improve the performance of Prolog reasoning, the checked situation is composed from the possible transition routes in the XPDL. This way will escape from backtracking on possible situations, which will easily cause overflow when there are many actions or transition branches. The designed possible transition routes are verified to ensure that the transition conditions, i.e., precondition of each action in the transition route, are satisfied.

From the above properties, the control flow perspective of a business process is verified at the semantic level. With further development, other perspectives such as data flow can also be verified.

C. Example of Business Process Models

When developing the prototype system, an example of order fulfillment process is used, as illustrated in Fig. 2. This can be considered as a typical process since it includes the five basic workflow patterns [2]. BPMN is used to show the process graphically and intuitively so as to avoid displaying the lengthy XPDL file.

XPDL and XSSL are both in XML syntax. As the following 1) XPDL Script and 2) XSSL Script show, these two scripts specify the activity of check_credit.

1) XPDL Script

```
<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>
```

Figure 2. Example Process of Order Fulfillment (BPMN)
2) **XSSL Script**

```xml
<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>
```

The above XSSL script attempts to specify the action of `check_credit` in a compact format that is convenient to be transformed to Situation Calculus specification in Prolog. It includes the arguments for action function and the precondition and postcondition to construct the precondition axioms and successor state axioms.

3) **Prolog Script**

```prolog
poss(check_credit(PID,CardNo,Rate),S):-
  order_status(PID,received,S),
  payWay(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).
```

The above Prolog program specifies the action of `check_credit` in a logical syntax, which implements the Situation Calculus specification. This Prolog-defined process model can be automatically generated from the XSSL-defined process model.

Moreover, some extra processing work should be done such as introducing the process id (PID) and recovery of data relations. This kind of information is expressed in XPDL and can also be extracted into XSSL, which will be the extension work.

### D. System Illustration

Fig. 3 shows the prototype system from the view of input and output. Fig. 4 shows the user interface of the implemented prototype system. To verify an XPDL-defined business process is the main function of the system. There are several steps to make the verification.

To **Load XPDL**, will parse the XPDL file and show the constituent activities and other related information such as initial situations for testing. To **Generate XSSL** can automatically generate the XSSL file from the XPDL file, and to **Transform Prolog** will automatically transform the XSSL file to the Prolog file.

After generating the Prolog-defined process model, the process can be verified. E.g., to check the activity of `enter_order`, there are two possible routes according to the XPDL definition (seeing Fig. 2). Firstly select them to form the situation to be verified and also set the initial situation or use the default setting; secondly, to **Start** the verification engine will start the prolog engine and build up the related knowledge base; to **Check Executability** will query this engine and show the result as $\bigcirc$ for success and $\times$ for failure. Current illustrated results show that the route including `check_cash` will fail if the initial situation only consists an order paid by credit card.
The verification employs the 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 causal relation about the state changes in the process.

The prototype system is implemented at activity level. The whole process is verified after checking each involved activity. It is prompt to automate the verification for the whole process if only we set the system to verify the last activity of the whole process and make some extension development.

Furthermore, to verify a process including subprocesses or complex activity block, the current system needs extending the transformation from XPDL to Situation Calculus. From the current theories about Situation Calculus, this is possible to be implemented and will be our next work.

V. Conclusion And Future Work

This hybrid BPDVS has the following advantages. Firstly, it is applicable to BPM industry. It employs the maturing standards as process description languages and enables integration among various BPM systems. Secondly, it obtains higher degree of preciseness and intelligence. These characteristics are implemented by applying the formalism of Situation Calculus. Based on the inferable model transformed from the standard process model, business process definition is verified semantically and syntactically. Thirdly, it bridges the gap between the informal industrial language and the formal logic-based language. The implemented prototype system demonstrates the feasibility of this approach.

There is still much work for further research. This BPDVS system only implements basic transformation between XPDL and Situation Calculus. Most of the XPDL concepts can be mapped to Situation Calculus, and thus the XSSL syntax should be further defined or extended.

REFERENCES