Formal Methods for Business Processes: A Survey

Sanjana Sahayaraj[1], S Sheerazuddin[2‡]

Abstract—**As service oriented architecture gains momentum, the business processes are composed of several modular web services which provide service to and take services from each other. As all communication between these web services happens over the internet in real time, it is very crucial that the way each web service participates will not affect the overall business process in any case. Web services description language is often used to describe the business processes. If formal models can be developed for these descriptions, the verification of various properties of the business process would be easier and accurate. This is what the paper focuses on.**

Keywords: BPEL-Business Process Execution Language; PN-Petri Nets; FSA- Finite State Automaton; SOA- Service Oriented Architecture

1 INTRODUCTION

As more business processes in organizations require the aid of web services, it is essential that these web services are no longer isolated and are able to collaborate and function as a single unit, profiting the organization. In such a scenario, the business process description should be fault-proof. This requires some formal methods of verifying the business process description for properties such as soundness, variability, the business compliance, process termination, compatibility between the various web services, et-cetera. BPEL is commonly used to describe the business processes. Web services are the basic modules that make up most of the business processes. When these modules are put together, just the BPEL description may not be sufficient in ensuring the proper behavior of the business process as a whole. This is where formal representations such as automatons and petri nets need to be explored. Once, the business processes are formally represented, the business organization can be sure of making profits by coalescing the web services together as a business process.

*

†[1]Sanjana Sahayaraj is a student in SSN College of Engineering, affiliated to Anna University, Chennai, India. sanjana.sahayaraj@gmail.com

‡[2]S Sheerazuddin is a faculty in the Department of Computer Science and Engineering, SSN College of Engineering, affiliated to Anna University, Chennai, India. sheerazuddins@ssn.edu.in

2 BUSINESS PROCESS EXECUTION LANGUAGE

Business Process Execution Language (BPEL) is used in organizations to specify the way in which a business process works. It is often associated with Business Process Modeling and Notation (BPMN). While BPMN is the visualization part, BPEL serves as the execution specification. BPEL is capable of describing the business processes as abstract process and as well as executable process. BPEL includes many features from other languages such as WSFL by IBM and XLANG by Microsoft.

An example BPEL process representing switch activity:

```
<process name="newSwitch"
targetNameSpace="http://ex.otn.com"
xmlns:services="http://exserv.otn.com">
 <switch name="s1">
  <invoke name="act1">
   <sources>
    <source linkname="tok1">
   </sources>
  </invoke>
  <invoke name="act2">
   <sources>
    <source linkname="tok2">
   </sources>
  </invoke>
 </switch>
</process>
```

There are two kinds of activities that can be included in a BPEL description. They are basic activities and structured activities. Basic activities are elementary activities like assign, reply, invoke, wait, etcetera and a few structured activities may be sequence, switch, while, flow, etcetera. Ever since, BPEL was introduced, several propositions have been made for its semantics. In this paper we shall discuss the semantics for BPEL based on finite state automaton and petri nets.

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

3 PETRI NETS TO REPRESENT BPEL

Petri Nets is a mathematical model that can be referred to as a kind of graph with two types of nodes namely places and transitions, between which arcs are present. Petri nets can represent basic and structured activities mentioned in the BPEL [18]. Representations of many structured activities in BPEL have been explored in [1]. Scope is one of the structured activities with comes along with its related concepts of event handlers, fault handlers, exception handlers and compensation handlers.

The event handlers can be associated with system occurrences or internal events or environmental occurrences or external events. Each event handler will be invoked if and only if the corresponding event occurs. An event handler can also have many instance operating simultaneously.

Petri nets also include control links which are facilities to depict the dependencies between activities. The link status can be true or false indicating whether the dependency condition has been satisfied or not. Let us say, a place in the petri net should have a specified number of tokens before a transition from it can be fired. Thus, the link status of each of the outgoing links from this place will be true only when all the tokens are present. In a similar way, join conditions are evaluated [18]. A join condition can be defined by a boolean expression where each variable of the expression corresponds to a single control link with the link status set to either true or false. In addition to verifying certain properties of the business process that is specified in the BPEL, petri nets can also be used to reveal any ambiguities if present in the BPEL specification. The concept of control links and structured activities has been clarified in [1]. WofBPEL is a tool that uses these petri nets to check for unreachable activities, conflicting message receipt actions, etcetera.

3.1 Scope and the handlers

The main focus of scope is event handling for normal flow and exception handling or fault handling if anything goes wrong. The main activity has event handlers, fault handlers and compensation handlers.The exception handling is represented using flags depicting the

- normal execution of a process without any exception being raised *to_continue*.
- situation where an error occurs and all activities need to stop executing *to_stop*.
- *snapshot* to indicate that a snapshot has been preserved after successful completion of scope.
- *no_snapshot* if the scope has been compensated and no snapshot is available.

Extension of petri nets has been done in [1] to represent the above.
In a sunny day scenario, the *to_continue* flag will be set and when the execution completes without any error, the *snapshot* is set for the scope. On the other hand, if an exception occurs, then *no_snapshot* is set with the *to_continue* changing to *to_stop*.

The event handlers are present to handle normal events that occur while the scope is running. An event handler can be invoked once or multiple times. An event handler is ready to be invoked once the corresponding scope starts. When an event occurs, an instance of the event handler is created. As long as the scope is active, the event handler remains active. The normal event is also enabled till the main activity of the scope is running. The completion of the scope has to wait for the completion of event handlers. Even if an event handler starts in the absence of any active events, it is allowed to complete. An event handler can have several instances running at the same time based on the nature of the activity.

Unlike this character of event handlers, there can be only one fault handler running at a time for a scope. A default handler is always present to handle the fault if not caught by any other fault handler.These handlers in the scope can be represented as basic activities within an enclosing structured activity using petri nets. Analysis of such nets give us vital information about how many event handlers are allowed, mapping of fault handler, etcetera. During mapping fault handlers, the causes for a fault are taken as transitions (fault events) in the petri net. There can be an arc from such a transition to a place, denoting the normal flow after fault occurrence. Nested fault handlers can be present to aid the case of faults occurring within fault handlers.

Compensation handlers and exception handlers are yet another class where the compensation handler refers to application-specific activities (basic or structured) that undo the work of a completed scope. The compensation may also involve one of the sub-scopes of the concerned scope. The compensate activity can be called from within a fault handler or compensation handler. This can be shown by an arc from a transition labeled compensate to a place in the compensation handler subnet [1]. Faults can occur during compensation handling and such faults are handled by fault handlers associated with the scope of the compensation handler.

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Termination of a scope due to fault is yet another case which is shown in BPEL by the execution of an exit activity. Execution of exit activity requires all the activities in the scope to be terminated as soon as possible without any fault handling or compensation handling. In case of termination, the core activity of the corresponding scope has to be skipped. This can be shown in PN using a transition *bypass*.

3.2 Verifying properties

Given the representational power of PN, it can be used to verify many properties of the business process under consideration. Automated analysis is also possible with tools like Woflan. For this, the BPEL has to be modelled onto workflow nets and they are known for their soundness property [18]. As an example here, the following petri net clearly checks the reachability property and shows the unreachable activity which otherwise might go undetected with just a BPEL specification.

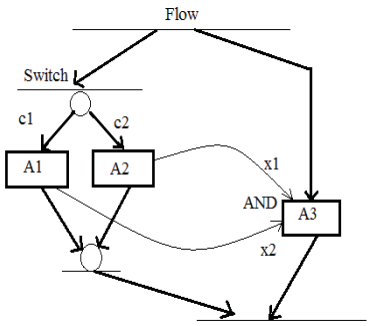

Figure 1. A flow where A3 is unreachable

The concept of presence of tokens and join condition evaluation in the above PN makes it easier to detect unreachability. WofBPEL tool is able to detect all runs of the PN starting from the initial state and going upto the final state [1]. For the above PN, WofBPEL would have reported that there is no possible run to reach A3. Two different methods are used for property verification. They are relaxed soundness and transition variants. Relaxed soundness keeps track of every possible run. Every transition in these runs is said to be relaxed sound and vice versa. Though relaxed soundness is complete, it involves expensive computation. Transition invariants includes a multiset of transitions, all of which cannot execute simultaneously.Every run from initial state to final state will correspond to a transition invariant.

When many web services are involved, it is a must to ensure that the service provided by one is consumed by the correct recipient. While the correct recipient is identified by the port type, partner link, etcetera in the BPEL, it is necessary to ensure that this recipient has only receive activity associated with it. This again can be solved by using petri nets to represent the BPEL and using some automated tool which generates an exhaustive state space and checks for these properties.

4 AUTOMATA BASED FORMAL MODELS FOR BPEL

The BPEL constructs can be converted into finite state automaton. Much work has been done in the ways of converting rational expressions to automatons. For the construction of a finite state machine corresponding to BPEL description, information regarding the peers and the communication (messages) between them have to be extracted in the first place. This is known as the peer list P and the message type M. Once the automaton has been obtained, details of this can be represented in the language Promela which again is given as input to the model checker SPIN [16]. This way, the correctness of the business process made of web services, as described in the BPEL can be verified. Now let us, see how BPEL for basic activities and structured activities can be represented by finite state machines. The first example describes a basic receive activity. If the input actions do not block time progress, then they have a lazy urgency, otherwise an eager urgency. The second example describes a structured activity- a sequence which is made of two other basic activities. The final state is connected to the initial state through local transitions.

Example 1:

```
<receive partnerLink=" ">
```

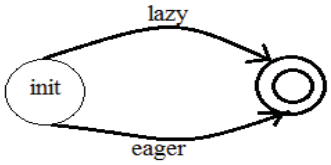

Figure 2. Receive activity

Example 2:

```
<sequence>
  <action_1 ...>
  <action_2 ...>
</sequence>
```

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

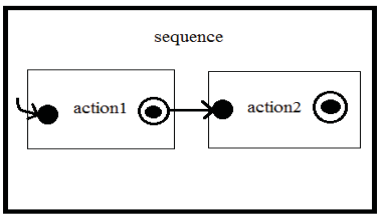

Figure 3. Automaton for a sequence activity

In promela, each automaton is taken as a process type *proctype* and the main process is called init. The translation of XPath expressions to Promela is a challenge and is a scope for research in the formal representation of business processes.

When many web services are involved, there might be an order of execution of these services. One typical scenario is that an answer message or reply message must follow a service request. Thus, the correctness properties in modeling these business processes, needs some temporal representation. They use Computational Tree Logic (CTL) in this paper because it not only shows the details regarding the present sequence of activities, but also what might happen in future in this business process which involves several services. In Computational Tree Logic, the changes with time are represented in a tree like model. The occurrence in the future can be the result of taking any of the path in the tree. Business process descriptions have been converted to CTL in [12] using XPath predicates. Since BPEL is an XML based language, XPath expressions can be used to select the nodes, some of which represent the partner service in the business process. Once the BPEL is converted into formal representation, verifying it should be the next focus. This formal model is converted into NuSMV language representation which is given as input to the NuSMV model checker. NuSMV is capable of analysing models that have properties expressed in Computational Tree Logic and Linear Temporal Logic. A framework is proposed in [12] in which, once the BPEL is input, all the information regarding each and every entity is extracted from the BPEL file. There can be a finite state machine for each entity in the business process. The result of this phase will be a set of graphs for each entity and a set of conditions for each message which may specify the order in which they have to be sent.

Another approach is to use Intermediate Format (IF) Model based on timed automata [13]. Each activity specified in the BPEL is represented here as an IF process. A structured activity can be represented by an IF process that dynamically creates its sub-activities. Most BPEL construct have a corresponding representation in the IF language. [16] proposes a conversation protocol which is a finite state automaton depicting the conversations between the various web services of a business process. Guarded automaton are also introduced showing the behavior of each peer in the composite web service, where a guard (XPath expression) is present for each transition.

Just like using petri nets to represent the join conditions, the IF timed automaton [13] can also be used. A special IF process called $linksManager$ is used. The $linksManager$ has knowledge all the sources and the targets (activities). When a source activity finishes, it evaluates the guards and sends a source message to the $linksManager$, which sends a target message with the guard to the target activity. When target receives message on all its inputs and the guard is satisfied, then the join condition evaluates to true. Otherwise, a join failure occurs. Just as in PN, scope can be represented by an IF process which creates two sub processes- one for its main activity and the other for its event handlers.

4.1 Web Service Automaton

A web service automaton (WSA) has a signature M which is defined by input events, output events and internal events [20]. It also has a set of states, a set of transitions and a transition relation. In WSA each activity is abstracted as a *machine*. For example, an activity X is said to be $MachineX$. In WSA, the activities are represented by rounded rectangles and arcs are present to depict the control flow and data flow, that the BPEL wishes to convey. Using WSA, unreachable activities such as the one shown in the petri nets section can be detected. WSA also has the feature of hierarchical representation of BPEL activities. As stated already, each activity is mapped into a machine. Once the child machine (child activity) reaches its final state, it signals its parent machine (parent activity). This signal can be in the form of a message denoted by a labeled arc. Sometimes, even in fault handling, the fault message of the children can be forwarded to the parent.

Just as in PN, the concept of scope is found even in WSA and this also encloses fault handlers, event handlers and compensation handlers. The event handler runs along with the primary or main activity of the scope. Here the compensation handler can be invoked only after the normal completion of scope is done. But in the case of fault handler, the normal execution stops as soon as a fault is encountered. Fault handlers (FH) may be present for child activities. Once the FH of child is invoked, it completes execution and the main activity and event handler of child also complete. Even after this the fault may be rethrown to the parent and the parent scope can complete only after this rethrown fault is handled by its FH. All these handlers are again represented as machines made of states and solid arcs. When the scope starts,

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the primary machine (meant to represent the primary activity) starts and the event handler's machine also starts simultaneously. Once the primary machine's final state is reached, the event handler is disabled. The state of the primary machine is made known by *done* message.

Data flow analysis is an important feature of WSA, that serves to capture the relation between inputs and outputs of BPEL activities. The data flow analysis can be through a shared model or centralized data flow model and the other approach is through analysing the data dependencies between the various activities. In the first approach, a *linkcentre* is represented by a rectangle through which all the data exchanges must pass. In the second approach, the activities are allowed to directly exchange data among each other.

4.2 Annotated DFA

An annotated DFA just like any other DFA has a set of states, set of final states, set of messages, set of transitions and set of relation between states. It also supports logical formulae. Boolean entities like $true$ and $false$, symbols and variables are part of the logical formulae supported by annotated DFA. It is used to recursively define a formal representation for the BPEL description. The recursion starts with the first activity which is a child of $< process >$ element. Here, the recursion starts with the creation of the start and final states and passed to a partial structure which is used in the recursion as input and output places. The start state of this partial DFA is denoted by q_{in} and the final state by q_{out} as they are the places to enter and exit the partial structure. Finally a full structure for the annotated DFA is obtained from the partial structure.

In this DFA, the simple activities usually have only a single state and no transitions. Usually the annotations are given for the transitions and thus in the simple activity, the single state is annotated by $true$. The two simple activities are empty activity, which is represented by a partial annotated DFA with a single state and terminate activity in which the output state is an empty state and the input is marked final. When it comes to structured activities, a number of partial structures are combined together to get the corresponding annotated DFA. The communication activities in BPEL are shown by message exchanges in annotated DFA.

In case of message sending, the recipient must be able to handle what was sent to it. The sender will select and send only those messages that are compatible with its normal working. On the other hand, not all the incoming messages are suitable for the receiver. While giving annotations, the sending messages are represented as a conjunction of messages, since they are mandatory transitions. But, as a receiver needs alternative options, the receiving messages are represented as a disjunction of messages.

4.3 Constraint Automata

Constraint automata are used especially in verifying those business processes that involve orchestration. In orchestration, a central co-ordinator is present between the various web services making up the business process. The formal language used to represent the BPEL before converting into automaton is called Reo which guarantees possibilities for both model checking and verification [6]. Reo consists of primitive connectors consisting of two ends. One end can be source which will be the sender of messages and the other will be sink which will be the recipient. A constrain automata is based on timed data streams in which each element is a pair. Each pair is made of data and time. The time will give information regarding whether the data is being sent or received. The constraint automata has a set of states, initial state, set of final states and set of transition relations. The state of the automaton changes over time. At each point of time, there may be a data item in a port of the component. Based on the characteristics of this data item, a transition fires and the state changes accordingly. Any basic or structured activity in BPEL can be mapped onto Reo circuits where each component has a start port and an end port. A tool has also been designed to automatically convert BPEL specification to Reo circuits and from there to automaton. Further verification can be performed on the obtained automaton to study the possible occurrences in the course of the business process in detail.

5 CONCLUSIONS

Thus, the mapping of BPEL onto PN and finite state machines, not only gives a clear visual representation of the business process but also provides a formal technique of verifying the validity of the service composition. Since the service composition involves several individual modular units, it is highly essential to make sure that they work together perfectly fine by designing a formal model. This is one important step before deploying any business process as a collection of web services. Given that, mappings onto Petri nets and finite state machines are tried and tested for most of the BPEL constructs, following these procedures benefits the business organization as they can be sure of the behavior of the web services when they try to collaborate and work together.

References

[1] Formal Semantics and Analysis of Control Flow in WS-BPEL-Chun Ouyang, Eric Verbeek, Wil M.P. van der Aalst, Stephen Breutel, Marlon Dumas and

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Arthur H.M. ter Hofstede - 2007.

[2] Transforming BPEL to Petri Nets Sebastian Hinz, Karsten Schmidt and Christian Stahl- 2005.

[3] Workflow Management Principles for Interactions between Petri Net-based Agents- Thomas Wagner and Daniel Moldt- Application and Theory of Petri Nets and Concurrency Lecture Notes in Computer ScienceVolume 9115,2015.

[4] Analyzing BPEL processes using Petri nets - H.M.W. Verbeek, W.M.P. van der Aalst-2005.

[5] Petri Nets- Properties, Analysis and Applications Tadao Murata -1988.

[6] Web Services and Formal Methods- S.Tasharofi et al; 4th International Workshop, WS-FM 2007.

[7] A Comparative Study of Web Service Composition via BPEL and Petri Nets Mohammad Salah Uddin, Member IACSIT, S. Ripon, Nakul C. Das, and Orin Hossain-2014.

[8] Petri Net Transformations for Business Processes A Survey Niels Lohmann, Eric Verbeek, and Remco Dijkman - Transactions on Petri Nets and Other Models of Concurrency II -2009.

[9] A Petri Net Semantics for BPEL- Informatik-Berichte 188, Humboldt-Universitt zu Berlin-Christian Stahl- July 2005.

[10] Translating Workflow Nets to BPEL Wil M.P. van der Aalst and Kristian Bisgaard Lassen - 2006.

[11] Analyzing Web Service Based Business Processes. In M. Cerioli, editor, Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering - A. Martens- 2005.

[12] Modelling and verification of BPEL business processes -Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software -2006.

[13] Transforming BPEL into Intermediate Format Language For Web Services Composition Testing - Mounir Lallali, Fatiha Zaidi, Ana Cavalli - 2008.

[14] Formal analysis of BPEL workflows with by model checking- Mate Kovacs, Daniel Varro and Laszlo Gonczy 2008.

[15] Elementary Net Systems - Grzegorz Rozenberg and Joost Engelfriet 1996.

[16] Analysis of Interacting BPEL Web Services - Xiang Fu, Tevfik Bultan, Jianwen Su - Systems and Information Theory(E.4) formal models of communication ACM- 2004.

[17] Timed automata: Semantics, algorithms and tools- J. Bengtsson, W. Yi, W. Reisig, G. Rozenberg (Eds.)- Lecture Notes on Concurrency and Petri Nets, Lecture Notes in Computer Science, vol. 3098, Springer-Verlag, 2004.

[18] Using Petri Nets in the formal representation of Business processes- Sanjana Sahayaraj, S Sheerazuddin, Seventh International Conference on Computational Intelligence, Modelling and Simulation 2015- accepted.

[19] Automata Semantics and Analysis of BPEL- Yongyan ZHENG, Paul KRAUSE- IEEE International Conference on Digital Ecosystems and Technologies- 2007.

[20] Transforming BPEL into annotated Deterministic Finite State Automata for Service Discovery- Andreas Wombacher, Peter Fankhauser - 2005.

ISBN: 978-988-19253-6-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)