An ECA-based Control-rule formalism for the BPEL Process Modularization*

Bang Ouyang, Farong Zhong**, Huan Liu

Department of Computer Science, Zhejiang Normal University
Jinhua 321004, Zhejiang Province, P. R. China, ouyb888@163.com, zfr@zjnu.cn, liuhuan04jsj@126.com

Abstract

Because businesses are constantly changing, the control flow of business process must be pluggable, i.e., it should be defined in a flexible manner so that any change in the flow can be easily accommodated. The problem of abstraction with respect to languages for Web Services Composition has gained a lot of attention lately. In this paper, An ECA-based control-rule formalism is introduced to modularize the monolithic BPEL process structure. Using such a formalism, business users not only can model executable cross-organizational business processes, but also can easily manage changing business environments.

© 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
Selection and/or peer-review under responsibility of the Intelligent Information Technology Application Research Association.
Keywords: component: formalism; BPEL Process; composition language abstraction; control-rules;

1. Introduction

Business Process Execution Language (BPEL) [1], short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS standard executable language for specifying actions within business processes with web services. Processes in Business Process Execution Language export and import information by using web service interfaces exclusively. BPEL has gained a lot of attraction and support from industry as well as academia. The momentum of BPEL is seen through case studies, and a large number of BPEL engine implementations in the industry suggest that BPEL will be the standard of choice for coordinated Web service composition implementations.

Service composition shares some common features with workflows [2]. BPEL is based on the concepts of workflows. However, businesses are constantly changing, the control flow must be pluggable, i.e., it should be defined in a flexible manner so that any change in the flow can be easily accommodated. Keeping this issue in mind, a concept of “control-rules” has been introduced to capture the control flow. In this approach all the activities in the control flow are ready to execute in parallel as soon as the process
starts, but the execution sequence is governed by links that propagate from activity to activity. Using rules
the modeler can specify the direction of flow from one activity to another based on an optional condition.†

2. Related Works

The problem of abstraction with respect to languages for Web services composition has gained a lot of
attention lately. Researchers have approached this problem from various angles having the same objective,
i.e., to facilitate the process of modeling service compositions.

Zeng [3] has developed a dynamic Web service composition system that is capable of dynamic
composition and modification of running workflows by using a business rule inference engine. Business
rule templates are used to define the business policies, which in turn are used to generate process schemas.
These rules must be defined by domain experts prior to modeling. An XML based language is developed to
map the process specifications into the composition engine which runs the process. Our approach on the
other hand, is build on top of industry standard language and captures rules directly from the modeler.

SWORD [4] is another developer toolkit for building composite Web services using rule-based plan
generation. To create a composite service, the service requestor only needs specify the initial and finial
states for the composite service, the plan generation then can be achieved using a rule-based expert system.
Again, this system also does not utilize BPEL or any other Web service standard, which renders the
approach unfeasible in practical use.

All the rule based approaches mentioned above apply pure rule based methodology. Charfi [5] claim
that pure rule based approach is not appropriate to capture all the aspects of Web services composition and
put forward a hybrid composition approach. The focus of this research is derived from Aspect Oriented
Programming (AOP) and the business rules are proposed to be implemented in an aspect-oriented extension
of BPEL called AO4BPEL [6]. The usage of such technique requires AO4BPEL complaint implementation,
which has not yet been provided.

Our approach differs from the above work as regards to supporting rule based Web services
composition in the following manner:

- Only one classification of rules is defined that handle the control flow part of the composition
  linking activities together.
- Instead of capturing control flow details using declarative if-then rules, which later on raises issues
  of refinement and transformation complexities, we use rule that are in BPEL native format.

3. Rule Based Composition

3.1. Overview

Business rules are pieces of knowledge about the business and it is not appropriate to bury that
knowledge deep in code where no one can identify it as such [7]. Recently there has been a trend of
introducing rules into Web services composition [8, 9]. These approaches follow a pure rule based
approach for the composition and encapsulate the whole development life cycle. This pure rule based
approach is not appropriate for developing Web service compositions [5]. A multitude of rules makes the
composition difficult to understand and maintain. We therefore advocate the use of rules for capturing the

*Supported by the National Natural Science Foundation of China under Grant No.60873234.
**Corresponding author.†
control and data flow only instead of specifying the whole composition process in terms of rules. Our partial rule approach allows for a more understandable service composition by reducing complexity and avoiding monolithic process definitions.

The authors in [5] also advocate that business rules cannot capture certain aspects of Web services composition. For instance, we cannot define a business rule that states that two services should be called in parallel. In fact, what is needed is a tight integration between rule-based languages and Web service composition languages in order to be able to use activities and variables as terms and facts in the rule language. By doing so we will have refined rules at a lower level of execution that can more easily define control flow semantics of the composition language. Therefore, we propose using BPEL activities as terms in our rule formalism instead of conventional declarative statements.

In BPEL, the core composition specification only defines the basic control and data flow between the services to be composed. If we break down the business logic underlying the composition into several parts or modules, the composition becomes more flexible since each of these parts can evolve independent of the rest and can be created, modified or deleted dynamically at runtime. The idea is to modularize the control flow into individual units, each represented as a rule, linking a single pre-activity with a single post-activity so that when business policies are subject to change, we only have to modify the corresponding units that modularize the implementation of the business process. Moreover, this modularization reduces the complexity of the composition and boosts the adaptability.

### 3.2. Control-Rule Formalism

Among various types of business rules identified by researchers [10, 11], one type is the “Reaction Rules” also known as ECA (Event-Condition-Action) rules that are concerned with the invocation of actions in response to events based conditions under which actions must be taken. The “control-rules” that we use are similar to ECA rules, in that an event is considered to be completion of an activity, condition specifies the transition condition, and action defines another activity that will be ready to execute after the first activity completes its execution.

The structure of a control-rule is as follows:

```
ControlRule {
  From (activity)
  [TransitionCondition (Boolean expression)]
  To (activity)
  Context (flow)
}
```

A control-rule links two activities together as a unit/module. “From” specifies a source activity upon the execution of which the “To” or target activity should be executed. “TransitionCondition” is an optional Boolean expression that functions as a guard for following this specified link. If the transition condition is omitted, it is deemed to be present with the constant value true. And if the transition condition evaluated to false, the link is marked as negative and the target activity is not executed. The links following this target activity are also marked as negative, i.e., the consequence of this decision is to propagate a negative status to the links downwards. The significance of transition conditions (used along with join conditions) is actuated in the case of choice and decision patterns.

When joining the activities defined by control-rules, the whole control flow is produced in the form of directed graph. This graph based approach does not require the structured constructs of BPEL. Thus constructs like sequence, flow, and switch are abstracted away from the modeler. The control flow is organized using nodes that perform work and lines of control between them describing the flow of execution. Here an issue arises, i.e., when the processes grows in size and has iterative tasks then maintaining the process becomes difficult [12], because control lines in such cases may have many tangled inter-dependencies. Changing the flow also becomes difficult as the modeler needs to be aware of
dependencies of a particular node across the entire flow. In order to get rid of this problem, we propose to use the while structured construct as an iterative activity. Another structured activity that we use is the pick activity which fundamentally is more an event management activity than a structured activity. To model a special case of simultaneously arriving messages we propose another structured activity called MultipleReceive that is internally represented with a flow construct and can contain more than one receive activities. All these structured activities may constitute a control flow of their own. Thus the control-rules can be hierarchically nested within these structures and the level at which the rule occurs can be identified by the Context attribute of the control-rule. In summary the activity of a control-rule can be anyone of the primitive activities of BPEL; assign (for data transformations); while (for looping); pick (differed choice, or event); or our own defined MultipleReceive activity.

4. Patterns analyst

It is important to analyze the approach regarding various control flow patterns that may be encountered during modeling. This is to ensure that any possible scenario can be handled using the control-rule formalism. In this chapter we analyze how the graph oriented structure of our approach can handle the most commonly occurring workflow patterns.

The control flow perspective provides an essential insight into business process modeling effectiveness and expressiveness. A conceptual model should describe all relevant static and dynamic aspects of the problem at hand. This implies that a process modeling technique should have sufficient expressive power in order to model all possible patterns [13]. As described in [14], a pattern “is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts”. Therefore it is important to analyze common patterns encountered in business process modeling. We evaluate our approach using a set of workflow patterns documented in [15] and communication patterns documented in [16].

4.1. Workflow Patterns

The workflow patterns have been compiled from an analysis of existing workflow languages and they capture typical control flow dependencies encountered in workflow modeling. These patterns are arguably suitable for analysis of Web services composition modeling, since the situations they capture are also relevant in this domain. We shall consider the patterns that are supported by BPEL only [2].

- **Sequence**: An activity in a workflow process is enabled after the completion of another activity in the same process. This pattern can be handled by using control-rule without any transition condition, originating from an activity a to another activity b (see Fig.1(a)).
- **Parallel Split**: A point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in
any order. This pattern can be handled by using control-rules without transition conditions, originating from one activity a and targeting at different activities b, c, d to be executed in parallel (see Fig.1(b))

- **Synchronization**: A point in the process where multiple parallel branches converge into one single thread of control, thus synchronizing multiple threads. It is an assumption of this pattern that after an incoming branch has been completed, it cannot be completed again while the merge is still waiting for other branches to be completed. This pattern can be handled by control-rules with no transition condition, originating from one activity a and targeting at different activities b, c, d. An AND join condition must be specified at the target activity (see Fig.1(c)).

- **Exclusive Choice**: A point in the workflow process where based on a decision or workflow control data, one of several branches is chosen. This pattern is handled by control-rules originating from one activity a and targeting different activities b, c, d. Each rule should be accompanied with a disjoint transition condition c1, c2 and c3 (see Fig.1(d)).

- **Simple Merge**: A point in the workflow process where two or more alternative branches come together without synchronization. It is an assumption of this pattern that none of the alternative branches is ever executed in parallel. The pattern can be handled by using control-rules originated from different activities b, c, d and targeting at one activity a. An OR join condition must be specified at the target activity (see Fig.1(e)).

- **Multi-Choice**: A point in the process, where, based on a decision or control data, a number of branches are chosen and executed as parallel threads. This pattern can be handled by control-rules originating from an activity a and targeting at different activities b, c, d. A joint transition condition specified will result in multiple threads of execution (see Fig.1(f)).

- **Synchronizing Merge**: A point in the process where multiple paths converge into one single thread. Some of these paths are “active” (i.e., they are being executed) and some are not. If only one path is active, the activity after the merge is triggered as soon as this path completes. If more than one path is active, synchronization of all active paths needs to take place before the next activity is triggered. This pattern is handled in the same way pattern “simple merge” is handled, i.e., using control-rules originated from different activities b, c, d and targeting at one activity a and having an OR join condition at the target activity.

- **Implicit Termination**: A given sub-process is terminated when there is nothing left to do, i.e., termination does not require an explicit termination activity. Implicit termination is supposed by the flow construct and as we use flow construct to envelope the control flow, the process is terminated implicitly.

- **Multiple Instances without Synchronization**: Within the context of a single case, multiple instances of an activity may be created, i.e., there is a facility for spawning of new threads of control, all of them independent of each other. This pattern can be handled by using a while activity with a condition and a context of the case that is spawning off new thread (say process A) and using an invoke activity inside the body of while that is invoking an operation of another process (say B). The process B will be receiving message from the corresponding invoke of process A which is have the attribute createInstance assigned to “yes”. This can cause multiple instances.

- **Multiple Instance with Synchronization**: This pattern has three sub categories. All can be handled using our approach because the solution is based on while and pick activity which our approach support. For more details on the solution of these patterns, reader is referred to [2]

- **Deferred Choice**: A point in a process where one among several alternative branches is chosen based on information which is not necessarily available when this point is reached. This can be handled using pick activity.

- **Cancel Activity and Cancel Case**: A cancel activity terminates a running instance of an activity, while canceling a case leads to the removal of an entire workflow instance. Cancel Case is solved with the
terminate activity, which is used to abandon all execution within a business process instance of which the terminate activity is a part. Cancel Activity is dealt with using fault and compensation handlers, specifying the course of action in cases of faults and cancellations.

4.2. Communication Patterns.

The Communication Patterns are related to the way in which system modules interact in the context of Enterprise Application Integration (EAI). They are arguably suitable for the analysis of the communication modeling abilities involving Web services composition, given the strong overlap between EAI and Web services technologies. Two types of communications are distinguished, namely synchronous and asynchronous communication.

Synchronous communication involves Request/Reply, One-Way and Synchronous Polling patterns, whereas asynchronous communication involves Message Passing patterns. All of these patterns are supported in our approach using various arrangements of messaging activities, i.e., receive, invoke and reply. According to the type of communication and the activity selected by the modeler, the system could attaches variable attributes to the activity based on the signatures of operations involved.

5. Contributions and future research

An ECA-based control-rule formalism is introduced to modularize the monolithic BPEL process structure. A control-rule defines the direction of flow from one activity to another along with a transition condition and it captures the control flow details of the business process. BPEL activities are used directly as terms in the rule system instead of conventional if-then-else declarative statements. This helps in defining more refined rules at the execution level that can capture the process behavioral in more realistic fashion. Changes in business policies pertinent to the process flow can be easily incorporated just by changing the rules, providing agility against change. As the flow is automatically generated using rules, the modeler does not have to bother about the interdependencies of activities. Also the rules can be changed independent to each other. According to BPEL specification, an exception, event or a compensation action can span over the entire process, over a group of activities, or over a single activity. Currently exception/event/compensation handling over the process level and single activity levels supported. In the future, flexibility will be provided to the modeler to specify these handlers over multiple activities. This can be achieved by introducing handler over an individual or group of rules. As a single rule can accumulate two activities at a time, it can provide the basis for multiple activity grouping.

References


