

# A Rule-based Middleware for Business Process Execution

Adrian Paschke<sup>1</sup> and Alexander Kozlenkov<sup>2</sup>

<sup>1</sup>RuleML Inc., Canada

Adrian.PaschkeATgmx.de

<sup>2</sup>Betfair Ltd., London

alex.kozlenkovATbetfair.com

**Abstract.** While past research in service oriented computing has focused on the fairly static functional description and the operators of services the dynamic and flexible composition and collaboration of services in business processes in style of semi-autonomous multi-agent systems has not been as thoroughly explored. Executable business process description languages such as BPEL provide only limited expressiveness to describe (business) decision logic and conditional reaction logic. In this paper we propose a rule-based representation approach to describe service-oriented business processes and implement a distributed execution middleware for such rule-based process specifications using rule engines as execution environments which are deployed as distributed agents / web-based inference services. This declarative rule-based approach which allows for semi-autonomous decisions and reactions within service-based business process execution has the potential to profoundly change the way IT services are used and collaborate on the Web.

## 1 Introduction

Flexibility in dynamically composing new business processes and integrating heterogeneous information systems (HIS) enabling ad-hoc cooperations is one of the main aims of the recent service oriented computing (SOC) paradigm. While past research in SOC has focused on the fairly static functional description and the operators of services - including publication, discovery, selection, binding and composition of basic services - the flexible and possibly self-autonomous collaboration, management and coordination of complex IT services in business processes is still an open task.

The currently most well-known language for formally describing business processes and business interaction protocols in a machine-readable way is the Business Process Execution Language (BPEL) [7]. BPEL is an orchestration language for web services providing facilities to enable sending and receiving messages. It describes processes as activities with control flow (e.g. for executing commands in sequence or in parallel). However, its expressiveness for representing conditional decision logic, e.g. for conditionally branching in parallel concurrent channels (split) or merging channels (join), is rather limited and constraint to procedural if-then-else flow control constructs. That is, BPEL only expresses qualifying conditions in the sense of material truth implications. For instance, a rule set "if given  $p$  and if  $p$  then  $q$  and if  $q$  then  $r$ " is expressible as a

nested truth implication rule ( $p \supseteq q \supseteq r$ ), but not as an automatically chained rule set as in logic programming based approaches. Accordingly, the BPEL approach is not modular and larger rule-based decision logic, such as business rules, can not be easily expressed in a compact declarative way as in logic programming. Moreover, the treatment of variables in BPEL is on the level of pure value assignments and ground pattern matching and a declarative (typed) unification as in logic programming approaches is missing.

In this paper we propose a declarative rule and event-based middleware for business process orchestration which combines technologies from declarative rule-based programming, in particular logic programming, with enterprise service technologies for complex event processing (CEP). This rule-based approach has the potential to profoundly change the way IT services are used and collaborate in business processes. Akin to multi-agent systems (MAS) the rule-based logic layer which wraps the existing web services and data sources allows for semi-autonomous decisions and reactions which are necessary e.g., for IT service management (ITSM) processes such as such service level management (SLM), change management, availability management (see ITIL for an overview [8]), business activity monitoring (BAM), and business process management (BPM) such as service execution in workflow-like business processes. Ultimately, our rule-based design artifact might put the vision of highly flexible and adaptive service supply chains into large scale practice. In this paper we contribute with a declarative rule-based service-oriented methodology and a scalable middleware to operationalize such a distributed rule-based approach where event-based communication and rule-based behavioral and decision logic plays a central role in connecting the various IT resources and Web-based services to business service networks. This addresses an urgent need businesses do have nowadays: to declarative describe and dynamically change their decision and business process logic which underpins their applications and service offerings in order to adapt to a flexible business environment.

The rest of the paper is organized as follows: In section 2 we give an overview of the relevant concepts. In section 3 we discuss the use of rules for describing conditional business process execution flows and introduce our rule-based approach combining standard derivation rules to represent decision logic and messaging reaction rules to describe message-driven process workflows. In section 4 we introduce the main components of our rule-based inference middleware. Section 5 concludes this paper.

## 2 Background

In this section we give an insight into the applied technologies and define relevant concepts and technologies.

### 2.1 Service Oriented Computing

The computing paradigm that utilizes services for developing applications in open distributed environments such as the Web is known as Service Oriented Computing (SOC). The vision is to build large scale service supply chains (also known as business services networks) which enable enterprises to define and execute web services based transactions and business processes across multiple business entities and domain boundaries using standardized (web) protocols. Web services, which emerged in the last time as the prevailing technology for implementing IT services on the web, have received a

great commitment from industry and research. A service-oriented architecture (SOA) expresses a perspective of software architecture that defines the use of loosely coupled software services to support the requirements of the business processes and software users. In a SOA environment, resources on a network are made available as independent services that can be accessed without knowledge of their underlying platform implementation. A SOA is not tied to a specific technology. It may be implemented using a wide range of technologies, including REST, DCOM, CORBA or Web Services. Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems using a SOA. Recently semantic web services (SWS) which provide an approach for representing the functionality of Web Services with the help of Semantic Web ontologies and BPEL (business process execution language) for web service orchestration attracts much attention from industry.

## 2.2 Complex Event Processing

Complex Event Processing (CEP) is a paradigm, which allows real-time reactions to detected complex events and state changes. A complex event type (i.e., the detection condition of a complex event) is built from occurred atomic or other complex event instances according e.g. to the operators of an event algebra. Event processing describes the process of selecting and composing complex events from raw events (event derivation), situation detection (detecting transitions in the universe that requires reaction either "reactive" or "proactive") and triggering actions as a consequence of the detected situation (complex event + conditional context). Events can be processed in real time without persistence (short-term) or processed in retrospect as a computation of persistent earlier events (long-term), but also in aggregated from, i.e., new (raw) events are directly added to the aggregation which is persistent. It can be done either actively (pull-model), i.e., based on actively monitoring the environment and, upon the detection of an event, trigger a reaction, or passively (push-model), i.e., the event occurrences are detected by an external component and send as event messages to the event system for further processing.

## 2.3 Declarative Rule Programming and Business Rules Management

Rule based systems have been investigated comprehensively in the realms of declarative programming and expert systems over the last two decades. And, in recent years we have seen the rise of a new type of software called business rule management systems (BRMS). These are systems to externalize business rules and to provide a facility for business rule management. Using (inference) rules has several advantages: reasoning with rules is based on a semantics of formal logic, usually a variation of first order predicate logic, and it is relatively easy for the end user to write rules. The basic idea is that users employ rules to express *what* they want, the responsibility to interpret this and to decide on *how* to do it is delegated to an interpreter (e.g., an inference engine or a just in time rule compiler). Early manifestations of business rule engines which have their roots in the realm of artificial intelligence and inference systems were complex, expensive to run and maintain and not very business-user friendly. Improved technology providing enhanced usability, scalability and performance, as well as less costly maintenance and better understanding of the underlying inference systems makes the current generation of business rule engines (BRE) and rules technology more usable.

There are different types of rules reaching from derivation rules, transformation rules to reaction rules and integrity rules. For the representation and execution of rules there are various ways, e.g., if-then constructs in procedural programming languages such as Java or C/C++ (with control flow), decision tables/trees, truth-functional constructs based on material implication, implications with constraints (e.g., OCL), triggers and effectors (e.g., SQL trigger), ECA rules as in active databases, production rules, or logical knowledge representation (KR) approaches based on subsets of first order logic and non-monotonic logic such as logic programming (LP) techniques. In this paper we exploit logic based rule languages; that is, derivation rules represented as logic programs, which has several advantages:

1. reasoning with derivation rules is based on a semantics of formal logic - a variation of first order predicate logic - enabling automated rule chaining by resolution and variable unification, which alleviates the burden of having to implement extensive control flows as in imperative programming and allows for easy extensibility without affecting the underlying mechanisms and architectures,
2. it allows for a compact and comprehensible human and machine-oriented representation and high levels of automation and flexibility to adapt to rapidly changing requirements, and
3. it is relatively easy for the end user to write rules and hence rapidly engineer and maintain rule-based decision and reaction logic

### 3 Rules for Business Processes Execution

In this section we argue for a rule-based approach to describe service-oriented business processes. In particular, we draw on backward-reasoning logic programming (LP) concepts to formalize decision logic in terms of derivation rules and combine them with forward-directed messaging reaction rules in order to exploit the benefits of both worlds.

For instance, a rule set from a Service Level Agreement (contractual service agreements describing non-functional properties such as quality of service) might define three different service schedules:

```

if current time between 8am and 6pm then prime schedule.
if current time between 6pm and 8am then standard schedule.
if current time between 0am and 4am then optional maintenance schedule.

if prime schedule then the service level "average availability"
has a low value of 98%, a median of 99% and a high value of 100%
and a response time which must be below 4 seconds.
...
if standard schedule then the responsible role for service outages
is the second admin.

```

As shown e.g. in the RBSLA project [9] such rule sets can be adequately represented as logic programs and easily extended with further rules, e.g. rules describing escalation levels if service levels are missed. Integrated into messaging reaction rules this rule-based decision logic can serve as expressive conditional logic in the reactive control flow of the business process execution and for the execution of activities in process instances.















