Hybrid Web Service Composition: Business Processes Meet Business Rules

Anis Charfi, Mira Mezini
Darmstadt University of Technology
D-64289 Darmstadt, Germany
{charfi,mezini}@informatik.tu-darmstadt.de

ABSTRACT
Over the last few years several process-based web service composition languages have emerged, such as BPEL4WS and BPML. These languages define the composition on the basis of a process that specifies the control and data flow among the services to be composed. In this approach, the whole business logic underlying the composition including business policies and constraints is coded as a monolithic block. As a result, business rules are hard to change without affecting the core composition logic.

In this paper, we propose a hybrid composition approach: The composition logic is broken down into a core part (the process) and several well-modularized business rules that exist and evolve independently. We also discuss two alternative technologies for implementing business rules in encapsulated units, using aspects and a rule-based engine. Our approach allows for a more modular and flexible web service composition.

Categories and Subject Descriptors
H.3.5 [**Information Systems**]: Information Storage and Retrieval– *Online Information Services: Web-based Services*

General Terms
Design, Management, Languages

Keywords
Web Service Composition, Business Rules, Aspect-oriented Programming, Modularity

1. INTRODUCTION
Web services embody the paradigm of *Service-Oriented Computing* [1]: Applications from different providers are offered as services that can be used, composed, and coordinated in a loosely coupled manner. Individual web services are capable of providing some functionality on their own but the greater value is derived by combining several web services to establish more powerful applications. For example, a travel web service can offer full vacation packages by combining several elementary web services such as flight, hotel and car rental.

Several workflow-based composition languages have emerged to express web service compositions, such as the *Business Process Execution Language for Web Services* (BPEL4WS or BPEL for short) [2], WSCI [3], BPML [4]. These languages define a business process that determines the logical dependencies between the composed web services. The process specifies the order of invocations (control flow) and rules for data transfer between them (data flow).

In this paper, we argue that this process-based approach to web service composition exhibits important shortcomings with regard to support for integration of business rules. We will focus on BPEL since this is becoming a standard language for web service composition.

According to *the Business Rules Group* [5], a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control the behavior of the business [5]. Business rules are usually expressed either as *constraints* or in the form *if conditions then action*. The conditions are also called *rule premises*. The business rule approach encompasses a collection of *terms* (definitions), *facts* (connection between terms) and *rules* (computation, constraints and conditional logic) [6]. *Terms* and *Facts* are statements that contain sensible business relevant observations, whereas *rules* are statements used to discover new information or guide decision making.

Business rules are especially useful in decision and policy-intensive business domains such as the finance and insurance sectors. They provide a means to express, manage and update pieces of business domain knowledge independent of the rest of the application. A business rule system is a system in which the rules are separated logically and perhaps physically from the other parts.

Let us now shortly consider the aforementioned shortcomings of BPEL – as a representative of process-oriented web service composition languages – in modeling business rules affecting compositions of web services. The problem is that the whole business logic underlying a web service composition is expressed as a monolithic block, namely the process specification. Each business constraint or business policy that must be enforced

ICSOC'04, November 15–19, 2004, New York, New York, USA.

throughout the process has to be expressed in terms of *activities* and must be integrated with the process specification. Such processes are not modular, complex, and hard to maintain; especially, in policy-intensive web service composition, they contain plenty of nested conditional activities (<switch>, <case>) that model *decision-making* points in the process. The lack of process modularity hampers reusability: Since all the business logic is defined in one unit, it is not possible to reuse parts of it.

Another problem is the lack of flexibility: Process-oriented languages assume that the composition is predefined and does not evolve: The only way to accommodate change is by modifying the process definition; hence, there is no support for dynamic process change. Flexibility and adaptability are very important features, especially in the highly dynamic context of modeling business rules for web services: Organizations involved in a web service composition may often change their business rules, their partners, and their collaboration conditions.

The lack of flexibility is tightly related to the lack of modularity: If we can 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. This motivates the need for a more fine-grained approach; with appropriate support, parts of the composition logic can be created, modified or deleted dynamically at runtime. To achieve these goals, we investigate a hybrid approach, which combines *business processes* as known e.g., from BPEL with *business rules* [5].

The idea is that the core composition specification only defines the basic control and data flow between the services to be composed using process-based approaches. Policy-sensitive aspects of the composition, i.e., business rules that are subject to change are modularized in separated units. This way, when business policies change, we only have to modify the corresponding units that modularize the implementation of the business rules. Moreover, this separation reduces the complexity of the composition and boosts the adaptability.

Presuming some *analysis* phase as the result of which business rules are identified and specified in the form of if/then statements, we focus on the *implementation* phase where rules are implemented in a modularized way with some specific technology. To this end, we consider two alternatives.

The first alternative is to employ aspect-oriented programming (AOP) for modularizing the implementation of the business rules. The implementation of the business rules tends to cut across several activities of a process definition; AOP provides means to modularize crosscutting concerns and has already been found valuable for modularizing business rules of object-oriented software [7]. We will indicate how concepts from the business rules world relate to AOP concepts and will outline how business rules can be implemented in our aspect-oriented extension of BPEL, called AO4BPEL [8]. As part of our investigation of the aspect-oriented approach embodied in AO4BPEL we will also indicate its limitations.

The second alternative for modularizing the implementation of business rules is to combine the process-based specification of service composition with dedicated approaches to declarative specification of business rules, such as *Java Expert System Shell Jess* [9] and *JRules* [10]. We refrain from a pure rule-based approach in which the entire composition logic is specified in the form of business rules: Such an approach is not appropriate because there would be no global view of the composition.

We consider business rules as part of the implementation of web service compositions i.e., they are comparable with executable business processes in BPEL4WS. Unlike abstract business processes in BPEL, which specify business protocols, business rules in our approach are not intended to be published among partners. They are not visible for external partners who use the resulting composite web service.

The remainder of the paper is organized as follows. In section 2, we briefly introduce process-based web service composition with an example in BPEL4WS. In section 3, we outline our hybrid composition approach and discuss the alternatives for modular implementation of business rules: Using AO4BPEL and an integration of a rule-based system with an orchestration engine. In section 4, we report on related work. Finally, we conclude by highlighting our contribution and outlining areas of future work in section 5.

2. BUSINESS RULES AND PROCESS-BASED WEB SERVICE COMPOSITION

After a brief overview of both process-oriented languages represented by BPEL and of the notion of business rules, this section discusses the problems of process-oriented languages with respect to modeling business rules.

2.1 Process-Oriented Composition Languages

Process-oriented composition languages such as BPEL4WS [2] and BPML [4] glue web services together by means of a process model. The latter specifies the interactions among web services as a workflow; it determines the order of these interactions (control flow) and manages the data exchanged by the participating services (data flow).

The building blocks of a process are called activities or tasks. An activity in BPEL4WS is either *primitive* such as <invoke> or *structured* such as <sequence>. Structured activities manage the overall process flow and the order of the primitive activities. For this purpose, BPEL4WS defines control structures such as loops, conditional branches and parallel execution. *Variables* and *partners* are other important elements of BPEL4WS. Variables are used for data exchange between activities whereas partners represent the external web services that interact with the composite web service.

A typical example for a composite web service is a travel agency scenario where a travel package is created by aggregating hotel and flight web services. Listing 1 shows a sample process specification for such a composition in BPEL. This process invokes the partner web services **airline** and **hotel** sequentially. A full vacation package is generated from the return values of these invocations and returned to the client.

```
<process name = "FullTravelPackage" …/>
 <sequence>
   <receive  partner="client"
             operation="getTravelPackage"
             variable="request"
             createInstance="yes" …/>
      …
   <sequence>
     <invoke  partner="airline"
              operation="getFlight"
              outputVariable="flightout"/>
     <invoke partner="hotel"
             operation="getRoom" …/>
   </sequence>
   <assign>…</assign>
   <reply partner="client"
          operation="getTravelPackage"
          variable="proposition" …/>
 </sequence>
</process>
```

Listing 1. A travel process

2.2 Business Rules

In a real-life travel agency, many business rules and constraints need to be integrated into the simple process of listing 1, such as "*if no flight is found for the dates given in the client request, do not search for accommodation (R1)"*. Other rules relate to pricing policies, such as "*if more than two persons travel together, the third one pays half price (R2)"*.

There are several classification schemas for business rules. According to [6], there are four kinds of business rules:

- A **constraint** rule is a statement that expresses an unconditional circumstance that must be true or false e.g., *a vacation request must have a departure airport and a destination airport.*
- An **action enabler** rule is a statement that checks conditions and upon finding them true initiates some action e.g., *if no flight is found, do not look for accommodation.*
- A **computation** rule is a statement that checks a condition and when the result is true, provides an algorithm to calculate the value of a term e.g., *if more than 2 persons travel together, the third pays only half price.*
- An **inference** rule is a statement that tests conditions and upon finding them true, establishes the truth of a new fact e.g., *if a customer is frequent customer, he gets a discount of 5 %.*

A business rule system puts special emphasis on the *expression*, *management* and *automation* of rules. The STEP principles of the business rules approach [6] are: *Separate, Trace, Externalize* and *Position* rules for change. We focus on business rules that are relevant to web service composition: i.e., either the *condition* or the *action* or both parts span several web services. For instance, the business rule, "*if no flight is found for the dates given in the client request, do not search for accommodation"* involves three web services: The airline and the hotel web service as well as the composition itself.

2.3 Integrating Rules in BPEL Processes

Business rules can be integrated with the process by adding activities. For example, in order to implement the rule R1 in BPEL, we add a **<switch>** activity which branches to the hotel activity only if a flight has been found previously. For rule R2, we need to find the activity where price calculation takes place and then insert several probably nested **<switch>/<case>** activities. In Listing 1, the price calculation is handled by the **<assign>** activity, which generates the vacation propositions.

The implementation of business rules becomes, however, more difficult if the condition part of the rule requires some logical derivation. This is the case in the following rule: *If a customer is frequent, (s)he qualifies for a discount of 5% on the package price (R3)*. In order to check whether a customer is frequent, we need to look at the other rules and perform additional computation, e.g., by invoking a web service façade of a database with customer information.

Furthermore, price calculation might also include service fees, for the calculation of which several policies can be thought of. There might be a flat-rate fee per request, as it is often the case with travel portals available today. Other policies can be to charge a fee per each product bought, or per database access performed during the booking process, etc. One can also enable clients to choose a policy that best fits their usage profile.

Considering such more sophisticated examples, we observe that the problems of implementing business rules in procedural and object-oriented languages [7][11] also arise in process-oriented languages. These problems are: (1) manually ordering and merging the conditional statements into one application flow, (2) loss of identity of the rules, and (3) dealing with the impact of changing one single rule. In the following, we elaborate on these problems.

Problems (1) and (2) can be traced down to the lack of modularity in the implementation of business rules. Let us illustrate the issue by our example of pricing policies. Even if we are able to use some externalized web service that encapsulates the state needed for price calculation and provides operations for manipulating this state, these operations will need to be triggered in various places of the travel booking processes. That is, the decision about *``where''* and *``when''* during the booking flow to trigger the pricing functionality is not encapsulated in a separate unit.

In fact, by its virtue of constituting a protocol between operations of the travel booking process and the pricing service, the code that is responsible for invoking the latter cannot be modularized in a process-based decomposition: It rather cuts across the modular process-based structure of the travel service composition. That is, implementing business rules effects, in general, sets of points in the execution of the web service composition which transcend process boundaries.

At present, BPEL does not provide concepts for crosscutting modularity [12]. This leads to tangled and scattered process

definitions: One process addresses several concerns and the implementation of a single concern appears in many places in the process definition. One can envisage that with each new business rule the process gets more and more complex. Furthermore, the rules are embedded in the process and they do no longer exist as a separate unit: It is difficult to extract, control and manage business rules because they are mapped to activities of the process model.

As a result, the process evolves into a huge monolithic block and each small change would require a thorough understanding of the whole process code. For example, if the travel agency changes its discount policy, it is necessary to extract all the business logic related to discounts out of the process specification and ensure that the process is modified consistently and is still correct and working after the integration of new business rules, or alteration of existing ones. In the very competitive business context worldwide, business rules evolve very often [6] as a result of new partnerships, changing strategies, mergers, etc.

We conclude that process-based composition languages do not provide support to capture business rules in modular units. In fact, the primary focus of process-based web service composition languages is the specification of activities and their ordering (control flow). The data flow is less important but it is still necessary for having executable processes. In BPEL, for example, only few activities are geared towards data management compared with the activities related to control flow and ordering of activities. 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 [6].

We call for a more straightforward way to express represent and manage business rules as a separate and externalized part of the composition. This way, if the discount policy changes, we just update the corresponding rules or add new ones. This enables us to modify the composition without understanding the whole process specification and makes change easier and faster. We also recognize that a pure rule-based approach is also not appropriate to capture all the aspects of web service composition. In fact, understanding the composition becomes very hard if it is expressed in a multitude of business rules. For these reasons, we advocate a hybrid approach that combines the paradigm of process-oriented composition with the business rules approach.

3. HYBRID WEB SERVICE COMPOSITION

Our discussion assumes a methodology for web service composition which distinguishes two phases, the *analysis* and the *implementation* phase, as illustrated in Figure 1. The analysis phase is out of the scope of this paper. We merely presume that in the analysis phase, the web service composition is specified as a business *process* and business *rules* are expressed in a declarative way. The business process is specified at an abstract level (do not confuse with abstract processes in BPEL) e.g., using a meta-language for process-oriented web service composition [13][14]. The latter relies on meta-model providing commonly used entities within a process definition, such as conditional branching, sequential and parallel activities, etc. The business rules are stated in a way, which is very close to how users think and talk. All business rules are collected in a *rule repository*.

The focus of our work is the *implementation* phase. Here, we propose to keep the separation between processes and business rules at the implementation level; we propose a technology that combines some process execution and rule execution technology, as schematically shown in Figure 1. That is, for the implementation, we have to select a language for the process specification and a concrete technology to implement and manage business rules. For the process we can e.g., choose BPEL and a compliant orchestration engine. We create an executable process by refining the abstract process of the analysis phase.

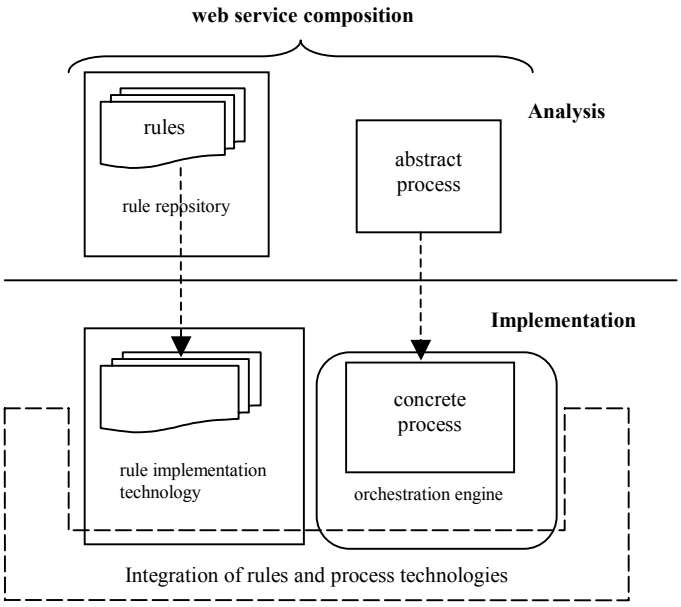

Figure 1. Hybrid composition approach

In this paper, we discuss two approaches for implementing business rules. One way is by using aspect-oriented programming [15] and encapsulating rules into aspect modules. To this end, we will investigate how business rules can be implemented in an aspect-oriented dialect of BPEL we have developed, called AO4BPEL [8]. Alternatively, we will briefly consider using a rule engine as the rule implementation technology [10][16].

Independent of the implementation technology used for the business rules, the key point is that in the approach schematically shown in Figure 1, business rules are a *separate*, *externalized* logical part of the composition. This has several advantages.

Once business rules are expressed explicitly as first-class entities, they can be *reused* across several compositions i.e., the same rule can be applied to many web service compositions. As a result, enterprise-wide business rules, as opposed to process-wide business rules, can be implemented more easily. A further advantage of separating business rules is that they can change independent of each other and of the rest of the composition.

This increases the flexibility of the composition especially if we take into account the fact that business rules tend to change more often than the rest of the composition [17]. R.G. Ross states “The

most significant changes do not come from re-engineering workflow, but from rethinking rules" [18]. This implies that by supporting change at the business rule level, we cover many adaptability requirements of the composition. With appropriate tool support, business rules can even be modified at runtime, enabling a configurable and dynamic web service composition. Thus, dynamic business rules turn out to be important instruments of adaptability.

3.1 Business Rules with AO4BPEL

This sub-section discusses an implementation of business rules based on aspects. We first give a short introduction to AO4BPEL, an aspect-oriented dialect of BPEL that we have developed. Next, we explain why aspect-oriented programming is a suitable vehicle for modeling business rules. Finally, we show by means of examples how business rules can be implemented in a modularized way in AO4BPEL.

3.1.1 Introduction to Aspects and AO4BPEL

AO4BPEL is an aspect-oriented extension to BPEL4WS that allows for more modular and dynamically adaptable web service composition [8]. Aspect-Oriented Programming (AOP) [15] is a paradigm that addresses the issue of modularizing crosscutting concerns - concerns whose implementation cuts across a given modular structure of the software resulting in code tangling and scattering [12]. Canonical examples of such concerns are authorization and authentication, business rules, profiling, object protocols, etc. [19].

AOP introduces units of modularity called *aspects* to overcome the inherent problem of code *scattering* and *tangling* due to crosscutting concerns in complex systems. Aspects associate sets of *join points* - well-defined points in the process execution - with additional behaviour defined in an *advice*. In AO4BPEL, each activity is a potential join point.

A collection of related join points is identified by a *pointcut* – a query over joint points. That is, a pointcut specifies the crosscutting structure of a concern and advice associate behavioural effect to this structure. The pointcut language of AO4BPEL is XPath [20]. That is, XPath expressions are used to select the activities where the advice code should be executed. Pointcuts can span several processes.

An *advice* in AO4BPEL is a BPEL activity that specifies some crosscutting behavior that should execute at certain join points. Like AspectJ [21], we support *before, after* and *around* advices. That is, the behavior defined in an advice can be executed before, after or instead a join point activity. The around advice allows replacing an activity by another.

The activity of integrating aspects into base functionality is called *weaving*. A weaver is a tool that integrates a base program's execution with aspects. In the case of AO4BPEL, the base program is the BPEL process. AO4BPEL supports dynamic weaving, i.e., aspects can be deployed or un-deployed at process interpretation time. We have implemented AO4BPEL as an aspect-aware orchestration engine for BPEL. This engine is both the aspect weaver and the process interpreter, i.e., with this engine, we can implement the composition without requiring an additional component to integrate the process engine and the business rules technology (cf. Figure 1).

3.1.2 Aspects and Business Rules

Aspects also feature the *if/then* flavor of business rules. They answer two questions: *when* and *what*: Join points specify *when* crosscutting functionality is required in the execution of the base program; advice captures that crosscutting functionality (*what*). Business rules also answer the same questions. The condition part of the rule answers the question *when* certain conditions are fulfilled, whereas the action part answers the question *what* action must be performed. We also put forward the analogy between the *base program* in AOP and, *terms* and *facts* in the business rule approach: conditions are statements over facts and terms like the pointcuts are statements over the static or dynamic structure of the base program.

To justify our claim that AO4BPEL is appropriate to implement business rules, in the following, we go through the different kinds of business rules and explain how each of them can be expressed by means of aspects in AO4BPEL.

The ***action enabler*** rules can be implemented in a straightforward way. Action enablers test conditions formulated on *facts* and *terms* and upon finding those true initiate some activity. In our case, facts and terms are partners, variables, and activities. We have to identify all those activities in the process where the *facts* or *terms* used in the condition change in such a way that the condition may become true. The action part of the rule can be expressed as an activity.

For example, in order to implement the rule *R1 - if no flight is found, do not look for accommodation* - we first find out which activities and variables relate to the condition part. In the BPEL process whose definition was shown in Listing.1, these are the **<invoke>** activity, which calls the airline web service and the variable **flightresponse** that holds the return value of the invocation. The *action* part of R1 affects the **<invoke>** activity of the accommodation procurement. So, the action is equivalent to skipping the invocation of the hotel web service.

The *condition* of the rule R1 is equivalent to the condition: *"if after invoking the airline web service, the **flightresponse** is null"*. Since the value of the variable **flightresponse** remains unchanged until the invocation of the hotel web service, the whole rule can be rewritten as *if before invoking the hotel web service the variable flightresponse is null, skip this invocation*. In AO4BPEL, this rule can be implemented by the aspect shown in Listing 2.

In this aspect, the action part of the business rule is expressed as an *around* advice activity, which is executed instead of the activity captured by the pointcut. The join point where the advice is weaved is the **<invoke>** activity that calls the hotel web service. The advice code is expressed as a **<switch>** activity. If **flightresponse** is null the around advice branches to the activity **<empty>**; otherwise, it branches to the original **<invoke>** activity. The **<proceed>** keyword denotes the original activity captured by the pointcut.

```
<aspect name= "if no flight found, do not
search hotel">
<variables>…</variables>
<pointcutandadvice type= "around">
<pointcut name="hotel procurement">
//process[@name="FullTravelPackage"]
//invoke[@portType="hotelPT"and
        @operation="getRoom"]
</pointcut>
<advice>
 <switch>
  <case condition="bpws:getVariableProperty
        (flightresponse,isnull)=1">
    <empty/>
  </case>
  <otherwise>
   <proceed>
  </otherwise>
 </switch>
  ...
</advice>
</pointcutandadvice>
</aspect>
```

Listing 2. Business rule as an aspect

Constraints are business rules that declare restrictions on some data e.g., *R4: a vacation request must contain a departure and destination airports*. In order to implement a constraint in AO4BPEL, we identify the activities where data is manipulated by the process or exchanged between the process and its partner web services. To enforce the constraint R4 we define a *before* advice, which executes before the **<receive>** activity shown in Listing1. The advice tests the variable **request** and if departure and destination airports are not specified a fault can be thrown using the **<throw>** activity. Other kinds of constraints can be implemented by identifying the data handling activities in the process where the value of some data changes. For more examples of how to use aspects to enforce constraints see [19].

We look now at ***computation rules***. These are statements that check a condition and provide an algorithm upon finding it true e.g., *R2: if more than two persons travel together, the third pays only half price.* We first consider the condition part of the rule and identify which activities and variables in the process are related to it. The number of persons is contained in the client request to the process of Listing 1. So, the condition is equivalent to *if the part numberOfPassengers of the variable request is greater than 2*. The variable **request** is used by the **<receive>** activity.

The action part of the rule R2 can be implemented as an *after* advice, which executes after the pricing activity. The pricing activity takes place after accommodation procurement. It calculates the price of the vacation package by summing up the price of the flight and hotel and the profit of the travel agency. The price information is part of the response messages of the partner web services. The rule R2 can be implemented as an aspect that declares a pointcut capturing the pricing activity and an *after advice* that modifies the price if more than 2 persons travel together. The transformation of the price data can be handled by means of an **<assign>** activity.

The fourth kind of business rules [6] are ***inference rules*** e.g., *R3: if a customer is frequent, (s)he qualifies for a discount of 5 %.* This kind of rule is more difficult to implement, because it requires logic-based reasoning. We need to resolve the condition part using other business rules. Therefore, we have to look for the business rules that specify when a customer is frequent. Let us assume that we find two rules that answer this question. The first one is *R5: if a customer has bought more than 5 travel packages, he is a frequent customer* and the second one is *R6: if a customer has bought products for a sum exceeding 4000 euros, he is a frequent customer*. The rule R3 can then be written as two rules without inference, which we will have to implement as explained before. The situation becomes more complex if either R5 or R6 was an inference rule that again must be resolved. The case of inference rules shows that a *business rule engine* would drastically ease rule management, since it automatically handles the logical rule dependencies.

So far, we saw that all kinds of business rules can be implemented by means of aspects in AO4BPEL. In this implementation, business rules are separated from the rest of the composition (process) not only logically but also physically. Each aspect can be defined in a separate file. In the analysis phase, business rules are expressed declaratively as if/then statements. In the implementation phase, each business rule is mapped to an aspect in AO4BPEL. Since AO4BPEL supports dynamic weaving, business rules can be activated or deactivated dynamically at process interpretation time. In this way, we adapt the composition to changing business policies and avoid disruptive change.

Before concluding this part, we would like to draw the attention to a fine difference between business rules and aspects with respect to the way conditions or pointcuts are specified. Business rules declaratively define conditions on data by saying *what* the state should be like in order for the action to fire, whereas aspects in AO4BPEL say *how* we can come to this state. The condition part of a business rule consists of patterns that match facts and not method calls or field access operations. That is, with AO4BPEL pointcuts rules are specified at a lower level of abstraction. We will return to this issue at the end of Sec. 3.2.

3.2 Issues in Using a Rule Based Engine

As already discussed, business rules can be implemented using aspects in AO4BPEL but the programmer still has to implement the rules himself and understand the interactions of the rules with the process. In addition, the programmer must also manage all the rules. This includes rule dependencies, checking rule consistency, combining rules, solving conflicts when two rules have the same conditions or overlapping conditions, etc.

As indicated in [7], the manual expression of all rule combinations is a cumbersome endeavour. We already mentioned that *inference rules* are especially hard to implement as aspects because they require logical reasoning and combination of rules.

For example, if we have two rules *R: if A then B* and *R': if B then C*, we would like to have an intelligent tool that automatically derives the rule *R'': if A then C*. This kind of logical reasoning is called *inference*.

Rule management can be performed by a *business rule engine*, thus letting the composer focus on process implementation. A *rule engine* is a component that applies business rules to application data (the process activities and variables) in a highly optimized way. The rules are stated in the declarative if/then form and are executed by the engine i.e., the rule implementation is generated by the rule engine and the programmer does not need to care about it. The rule engine controls the selection and activation of rules automatically. The *working memory* is a component of a rule engine that contains the data on which the rules operate. In the case of web service composition, the working memory would contain the process data and its runtime information.

Some rule technologies [9][10] can even generate code out of some high-level declarative specification of business rules. This provides for more *simplicity*. Business rules are specified in a *declarative way*, in a plain language, and independent of any technology. They are also *self-documenting*. This offers improved visibility to programmers and non-technical business people and enables them to change the composition without worrying about all process details.

We are aware of the fact that the simplicity from the user perspective is accompanied by an increasing complexity of the underlying orchestration engine. The execution of the processes becomes more complicated because it must also be integrated with the implementation of the business rules. This is the function of the integration layer shown in Figure 1.

There are two formalisms behind rule-based systems: *production rules* [22] and *first-order predicate logic*. Several systems use these formalisms to integrate a rule-based language with object-oriented languages like Java. The most notable ones are *Jess* [9], *JRules* [10] and *Java SweetRules* [23].

However, these systems are not appropriate for process-based web service composition. What is needed is a tight integration between rule-based languages and web service composition languages in order to be able to use *activities*, *variables* and *partners* as *terms* and *facts* in the rule language. To the best of our knowledge, there is no hybrid system integrating rule-based language and process-based service composition yet. Such a hybrid system is not among the contributions of this paper, either. In what follows, we simply consider some issues that arise and need to be carefully investigated when undertaking such an endeavor. We are considering these issues in our ongoing work.

Similarly to the many hybrid systems available today integrating rule-based and object-oriented paradigms [7], there are two possible ways to integrate process-oriented and rule-based paradigms. One way is to adapt one of the languages to be more compatible with the other by extending e.g., the rule-based language with process-oriented features, or the other way around. The second way is to enhance one of the languages with an interface to the other language, so that the features of the latter can be used in programs written in the former.

As argued in [7], both approaches suffer from the lack of seamless integration. In either approach, there is paradigm mismatch which the programmer is confronted with. The programmer modeling the core composition logic using a process-based language is confronted with constructs of a rule-based language, or vice-versa. In this regard, an implementation of our hybrid composition approach by a hybrid system integrating a rule-based language with a process-based web service composition language is inferior to the aspect-oriented implementation. Using an aspect-oriented dialect of BPEL for the definition of the business rules as discussed in the previous subsection has the advantages of consistency for process authors. In addition, the verification of the properties of the resulting composition would be easier if the same paradigm is used, i.e., in our case activities for both process and advice specification.

There are two alternatives for addressing the integration of a rule based system with a process interpreter. The first alternative would be to adapt to the web service composition context the approach presented in [7] for integrating rule-based and object-oriented languages. The basic idea is to have the core composition be modeled as a BPEL process and business rules be modeled in a rule-based language. Aspect-oriented concepts would then be used to specify how to map process model specifications and rule specifications behind the scenes.

More promising, though, seems to investigate using a more expressive pointcut language with aspects, such as the one used in the aspect-oriented language ALPHA [24]. Alpha uses Prolog [25] as the pointcut language, i.e., pointcuts are Prolog queries operating on rich models of program execution as diverse as the AST, the execution trace, and the object graph model. In the future, we will investigate how well business rules can be modelled in ALPHA and in case of promising results we will consider integrating a similar pointcut language in AO4BPEL.

4. RELATED WORK

A lot of research work is being conducted in the area of web service composition. Most of this work focuses on process-based composition. The originality of this paper lies in the use of business rules as an integral part of the composition logic.

In [13], *Yang et al.,* present a rule-based approach to the composition lifecycle. They introduce a phased approach to the development process of service compositions. Their approach spans abstract definition, scheduling, construction, execution and evolution. They specify the composition using a process in BPEL or a similar language. Unlike our idea, they consider only phase-related business rules e.g., resource selection constraints or runtime constraints. Their objective is to make the lifecycle of the composition more flexible. In our work, business rules are part of the composition itself and not just used as instruments to support the composition lifecycle.

Several rule-based systems have been integrated with object-oriented languages. In [7], the author uses aspects to implement rules in object-oriented languages and then show several shortcomings of such an approach. Instead, she proposes using aspects for the purpose of integration. With aspects, she separates and encapsulates the connection of business rules to the core application in order to achieve high flexibility and reusability.

Aspect-oriented languages encapsulate the connector code. In our work, we highlighted the similarities of business rules and aspects and considered AOP as an implementation technology for business rules and not as a connection technology between rules and processes. Some of the shortcomings presented in [7] are also present in the field of process-based composition web service composition but they do not fully apply because of the distributed and loosely coupled nature of web services.

In [26], the authors compare *Event Condition Action* rules (ECA) in the area of active data base with aspects in *AspectJ* and they have identified several commonalities. ECA rules can be seen as a subset of business rules because business rules do not necessarily require a database. Moreover, ECA rules are not adequate to express *constraints* and *inference*. Business rules are present in different areas e.g., in object-oriented applications, in relational databases and in workflow management systems [27]. They also come in various flavors either as *production rules*, *integrity rules in SQL*, *ECA-rules*, *logic rule* like in Prolog, etc.

Some *Business Process Management* software already offers rule support e.g., the *Corticon Decision Management Platform* [28] allows users to specify business rules that apply to a workflow. Since process-based composition is also a kind of workflow, we think the same approach can be also applied in the context of web service flows.

5. CONCLUSION AND FUTURE WORK

In this paper, we applied *the divide and conquer* principle to web service composition by explicitly separating business rules from the process specification. The combination of the business rules approach with the process-oriented composition solves a twofold problem. First, we provided a solution to the problem of dynamic adaptation of the composition. In fact, current standards for process-based web service composition are not capable to deal with the flexibility requirements of composite web services. Second, business rules are important assets of a business organization that embody valuable domain knowledge. So, it is no longer acceptable to bury them in the rest of the composition.

Our hybrid approach allows for a more understandable web service composition by reducing complexity and avoiding monolithic composition. Each part of the composition logic is expressed in the more suitable way either as a business rule or as a process activity. We call for an approach which consists of two phases: In the analysis phase business rules are discovered and expressed declaratively and the process is specified in an abstract way. In the implementation phase, the process is defined in a specific language and the business rules are implemented with some technology that needs to be integrated with the process orchestration engine.

We outlined the similarities between aspects and business rules and explained how business rules can be implemented using AO4BPEL and our aspect-aware orchestration engine. This engine plays the role of an integration technology for business rules and BPEL processes. We also mentioned that a rule engine is an attractive alternative to implement business rules because it takes on rule management. In the future, we will focus on the following issues. First, we will consider in what extent we can apply aspect composition techniques to resolve conflicting business rules. Second, we will investigate the possibility to generate aspects automatically from business rules. Third, a methodology is needed to distinguish the parts of the composition that should be specified as business rules from those that should be specified as process activities. Last but not least, we plan to investigate the alternative of using a rule-based engine as the base technology to which business rules are mapped. As indicated, this alternative would provide direct support for rule management and composition. The issue that remains to be solved in this case is the integration of a rule engine with the process engine. In the vein of the work presented in [7], an approach to consider is to use aspect-oriented technology for implementing this integration.

6. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful suggestions. We also thank all members of the *Software Technology Group* for their comments on earlier drafts of this paper, and Dimka Karastoyanova for the fruitful discussion on web service composition.

7. REFERENCES

[1] M. P. Papazoglou, *Service-Oriented Computing: Concepts, Characteristics and Directions*, 4th International Conference on Web Information Systems Engineering (WISE'03) , Rome, Italy, 2003.

[2] F. Curbera et. al. *Business Process Execution Language for Web Services*, version 1.1, May 2003.

[3] A. Arkin et al., *Web Service Choreography Interface 1.0*, W3C, 2002.

[4] A. Arkin et al., *Business Process Modeling Language-BPML 1.0*, 2002.

[5] The Business Rules Group, *Defining Business Rules, What are they really?* www.businessrulesgroup.org, July 2000.

[6] B. von Halle; *Business Rules Applied: Building Better Systems using the Business Rules Approach*, Wiley, 2001.

[7] M. D'hondt: *Hybrid Aspects for integrating Rule-based Knowledge and Object-Oriented Functionality*, Phd Thesis, Vrije Universiteit Brussel, May 2004.

[8] A. Charfi, M. Mezini. *Aspect Oriented Web Service Composition*, in Proceedings of the European Conference on Web Services ECOWS 2004, LNCS 3250.

[9] E.J. Friedmann-Hill, *JESS: The Java Expert System Shell*, http://herzberg.ca.sandia.gov/jess/

[10] ILOG JRules, http://www.ilog.com/products/jrules/

[11] P. Jackson, *Introduction to Expert Systems*. Addison-Wesley, 1986.

[12] H. Masuhara, G. Kiczales. *Modeling Crosscutting in Aspect-Oriented Mechanisms*. In Proceedings of ECOOP2003, LNCS 2743, pp.2-28, Darmstadt, Germany, 2003.

[13] J. Yang, M. Papazoglou, B. Örriens, W. van Heuvel., *A Rule Based Approach to the Service Composition Life-Cylce*, 1st International Conference on Service Oriented Computing ICSOC, Trento, Italy, 2003.

[14] D. Karastoyanova, A. Buchmann, *A Methodology for Development and Execution of Web Service-based Business*

Processess, 1st Australian Workshop on Engineering Service-Oriented Systems, Melbourne , 2004.

[15] Aspect–Oriented Software Development, http://www.aosd.net

[16] Blaze Advisor Rules Management Technology, http://www.blazesoft.com

[17] A. Arsanjani. Rule Object 2001: *A pattern language for adaptable and scalable business rule construction*, 8th PLoP conference, Illinois, USA, 2001.

[18] R. G. Ross, *Principles of the Business Rules Approach*, Addison-Wesley, 2003.

[19] R. Laddad. *AspectJ in Action*. Manning Publications, 2003.

[20] XML Path Language 1.0, http://www.w3.org/TR/xpath

[21] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold., *Overview of AspectJ*, Proceedings of ECOOP 2001, Budapest, Hungary

[22] L. Brownston, R. Farrell, E. Kant, N. Martin., *Programming Expert System in OPS5: An introduction to Rule-based Programming*. Addison-Wesley, 1985.

[23] B. Grosof, Y. Kabbaj,T. Poon, M. Ghande et al., *Semantic Web Enabling Technology (SWEET)*

[24] K. Ostermann, M. Mezini, *Design and Implementation of Pointcuts Over Rich Program Models.* Technical Report, Department of Computer Science. Darmstadt University of Technology, June 2004.

[25] P. Flach, *Simply Logical: Intelligent Reasoning by Example*. John Wiley, 1994.

[26] M. Cilia, M. Haupt, M. Mezini, A. Buchmann., *The Convergence of AOP and Active Database: Towards Reactive Middleware,*. 2nd International Conference on Generative Programming and Component Engineering (GPCE), 2003.

[27] B. Grosof, H. Boley, *Introduction to RuleML,* Invited talk at Joint Committee on Agent Markup Languages, 2002.

[28] Corticon Decision Management Platform, http://www.corticon.com/html/so_platform.htm