Towards a flexible service integration through separation of business rules

Camlon H. Asuncion[1], Maria-Eugenia Iacob[2], and Marten J. van Sinderen[1]
Center for Telematics and Information Technology[1], and the School of Management and Governance[2]
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{c.h.asuncion, m.e.iacob, m.j.vansinderen}@utwente.nl

Abstract — **Driven by dynamic market demands, enterprises are continuously exploring collaborations with others to add value to their services and seize new market opportunities. Achieving enterprise collaboration is facilitated by Enterprise Application Integration and Business-to-Business approaches that employ architectural paradigms like Service Oriented Architecture and incorporate technological advancements in networking and computing. However, flexibility remains a major challenge related to enterprise collaboration. How can changes in demands and opportunities be reflected in collaboration solutions with minimum time and effort and with maximum reuse of existing applications? This paper proposes an approach towards a more flexible integration of enterprise applications in the context of service mediation. We achieve this by combining goal-based, model-driven and service-oriented approaches. In particular, we pay special attention to the separation of business rules from the business process of the integration solution. Specifying the requirements as goal models, we separate those parts which are more likely to evolve over time in terms of business rules. These business rules are then made executable by exposing them as Web services and incorporating them into the design of the business process. Thus, should the business rules change, the business process remains unaffected. Finally, this paper also provides an evaluation of the flexibility of our solution in relation to the current work in business process flexibility research.**

Service-oriented mediation; Goal-oriented requirements engineering; Business rules; Model-driven development; Process flexibility

I. INTRODUCTION

Enabled by advances in networking and computing technologies, autonomous enterprises are now joining networks of enterprises (often called *virtual enterprises*) [31]. This collaboration is essential in today's era. To a large extent, an enterprise can ensure its continued competitiveness with its ability to seamlessly interoperate with others to foster innovation, exploit better business opportunities, and ultimately, to collectively achieve added value in offered products and services [10][31].

Achieving such networks of enterprises, however, is not an easy task. Among the current challenges in enterprise computing is providing *interoperability* solutions to support seamless collaboration. Most of today's industries have investments in large legacy systems that were not originally designed to be interoperable, aggravated further by the oversupply or lack of standards, proprietary developments or extensions, and heterogeneous hardware and software platforms. Another challenge is for enterprises to be *flexible* so that they can continuously adapt to dynamically evolving demands and opportunities from within and outside their business environment. This is especially true when business goals need to be changed, new policies are introduced, temporary partnerships need to be created, technological advancements need to be considered, etc. It is important therefore that such networked enterprises are designed for change so that they can quickly respond to emerging business opportunities [31].

This paper responds to these challenges two-fold: Firstly, and in particular, by providing a solution in the context of mismatching process and data specifications in the interoperation of enterprise applications through service mediation. Service mediation is ideal when legacy systems need to interoperate but have existing and often difficult-to-change service implementations. Adapting such systems with respect to the sequence of message invocations and the semantics of message elements is oftentimes difficult, if not impossible. Secondly, we respond to the call for a flexible integration solution by investigating the combination of goal-oriented, rule-based, model-driven and service-oriented approaches. We demonstrate this by building on our previous work [3][4] where we proposed an approach that separates the business process of the integration from the business rules that constrain the process. With this approach, the dynamic aspects of the requirements (i.e. those that are more likely to change rapidly overtime) are specified in terms of business rules, while the more stable aspects remain in the business process. Thus, should there be changes to the business rules, for example, then only the rule specifications change and not the entire business process. Whereas the focus of our previous work was on the overall integration approach, this paper focuses on the further investigation of the flexibility quality of the approach. This quality was previously treated only rudimentarily and thus warrants further exploration, analysis and validation.

The rest of the paper is organized as follows: Section 2 provides a brief introduction of the relevant theoretical concepts. Section 3 describes our generic framework for achieving flexibility including its implementation using specific technologies. Section 4 shows an application of the framework using an illustrative case scenario. Section 5 positions our work with respect to current research on business process flexibility and other related work. Finally, Sections 6 presents our conclusions and future work.

II. PRELIMINARIES

This section briefly discusses some definitions of key concepts used in this paper.

A. Flexibility

A business process is flexible if one can change only those parts that are affected by the change without affecting those that are not; i.e., the entire business process does not have to be completely replaced as a result of the change [27][30]. Additionally, Kasi and Tang [20] propose that a process is flexible if the change can be made in less time, with less cost, and less effort.

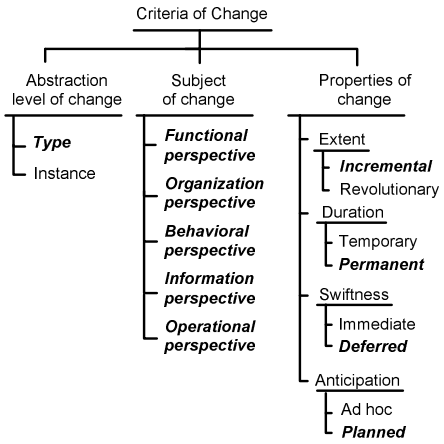

Figure 1. Taxonomy of flexibility criteria [27]

We use the taxonomy of business process flexibility proposed by Regev, et al. [27]. Although the authors do not propose ways to achieve process flexibility using the criteria, we find the taxonomy generic enough to position the flexibility quality of our approach. Regev introduces three orthogonal criteria of change: *abstraction level of change*, *subject of change*, and *properties of change* (see Figure 1).

Changes in the goals, strategy, constraints or stakeholder needs may consequently necessitate changes in the process at two levels of abstraction: process type level or process instance level. The process type level describes changes to the standard way of working created during design time while the process instance level describes slight deviations (e.g., exceptional situations) from the standard way of working during runtime.

As to the subject of change, it can be viewed in terms of perspectives (both at the type and instance levels). Functional perspective describes the goal of the process. Operational perspective describes activities that occur during process execution. Behavioral perspective describes which preconditions cause the activity to be executed. Information perspective describes the information exchanged between activities. Finally, organizational perspective describes the participating actors of the process.

Properties of change include extent, duration, swiftness and anticipation. The extent of change can either be incremental or revolutionary – the former introduces change to an existing process type while the latter requires creating a completely new process. The duration of change can either be temporary or permanent – the former is valid only for a limited period and resets afterwards while the latter is valid until the next change and does not reset. Swiftness of change can be viewed in terms of being immediate or deferred – the former applies the change to all currently running process instances whereas the latter applies the change only to process instances that are yet to be created.

B. Goal-Oriented Requirements Engineering

Faced by the inadequacy of traditional approaches to requirements engineering, researchers since the 1990s have looked at how objectives that a system should achieve can best be captured at various levels of abstraction. They started to treat these system objectives or goals as first-class citizens. This is in contrast with early practices where requirements focus only on processes and data without capturing the rationale of the software systems, making it difficult to relate requirements to the higher problem domain [25]. This gave way to *Goal-Oriented Requirements Engineering* (GORE) which investigates how goals can be used for eliciting, elaborating, structuring, specifying, analyzing, negotiating, documenting, and modifying software requirements [24].

Goals are "*high-level objectives of the business organization or system. They capture the reasons why a system is needed and guide decisions at various levels within the enterprise*" [2]. Goals are *optative* properties of a system under development. They express intentions, wishes, or desires, and are thus declarative [19], e.g., "*Increase product sales to 5%*". Goals are essential as they can be used to elaborate system requirements, provide decision makers a sufficient level of abstraction in specifying and validating system design choices at the business level, and for communicating such choices among different stakeholders [15]. An important activity in GORE is goal operationalization where a goal is refined or decomposed until its sub-goals have enough detail to define its operation [2][24]. An operation can be seen as a functionality of the system-to-be or be expressed as a business rule through a specification of its pre, trigger and post conditions [15].

C. Business Rules

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 or influence the behavior of the business*" [8]. Business rules are thus statements about guidelines and restrictions that define business operations. They are a set of permissible conditions that guide a business event so that it occurs in a way that conforms to desirable business outcomes [13]. Business rules, like goals, may come from outside the organization, such as laws or customs that guide the action of individuals and the organization, or from within the organization such as business policies that together achieve some higher business goals [34].

However, more often than not, business rules may be scattered everywhere in the organization, and may not be even stated explicitly or made readily available. It is useful therefore that these rules are expressed, managed and updated efficiently and explicitly, independent of the rest of

the processes that use them. When business rules are logically (or even physically) independent from their platform-specific implementation, they are better understood and made more easily accessible [34].

A useful and salient property of business rules is that they can be specified in a near-natural language lending themselves easily to the better understanding, specification, and validation requirements by stakeholders [26]. They can also be specified in an executable rule expression and then deployed into a rule-based engine. Most executable business rules are written in `IF…THEN` statements, such as "*IF the insurance claim is greater than €10000, THEN require approval from the Manager*".

D. Service Mediation

We define service mediation as "*to act as an intermediary agent in reconciling differences between services of two or more systems*" [21]. It involves reconciling two types of differences or mismatches: *process* and *data*. Process mismatches occur when systems use services that define different messages or different ordering of message exchanges. Data mismatches occur when systems use different information models (or vocabularies) to describe the messages that are exchanged by their services.

We approach service mediation as a composition problem: each service that is requested by some system has to be composed from one or more services that are provided by the other systems and, possibly, by the same system. For example, in Figure 2, Mediator *M* offers a service that matches the requested service *S1* of system *A* by composing services *S3* and *S4* offered by system *B*.

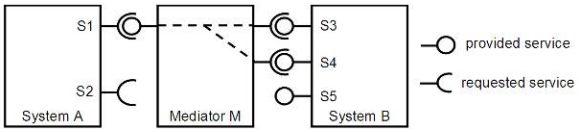

Figure 2. Service mediation as service composition

III. A Framework for Goal-Oriented, Rule-based, Model-driven Service Design

A. Framework

Driven by the need to enable organizations to have their IT systems adapt swiftly and coherently to the ever dynamic nature of business demands, Iacob et al. [15][16] propose a framework for the goal-driven design of service-oriented systems using model-driven techniques.

The framework, shown in Figure 3, is divided vertically into two spaces: *Design space* and *Goals and Business Rule (G&BR) space*. These spaces are divided horizontally into the distinguished levels of MDA: *Computation Independent Model* (CIM) level, *Platform-Independent Model* (PIM) level and *Platform Specific Model* (PSM) level. The Design space expresses models in design languages such as Unified Modeling Language (UML), business process modeling or architectural description languages. The G&BR space models goals and rules in special-purpose specification languages.

The framework argues that there may be a strong symmetry between the design space and the G&BR space; i.e., for any design model, there may be a corresponding rule set specification (indicated by the "+" symbol in Figure 3). Model Driven Architecture (MDA) allows transformation of these models at the different levels (indicated by the arrow). SOA can then be used to *integrate* these business rules in the design and composition of service-based business processes.

Particularly when focusing at the G&BR space, it is therefore possible to derive executable business rules from high-level organizational goals. High-level goals can be refined into sub-goals and operationalized into business rules at the CIM level. At the PIM level, these business rules are translated into an XML-based rule specification for added interoperability. Finally, at the PSM level, they are made executable. Exposing executable business rules as Web services allows them to be incorporated into the design of the business process.

In summary, the framework seeks to incorporate a business view in service development by using the concept of goals in eliciting, structuring, and modeling requirements. Goal models are used to capture the requirements at a level where it is easily understandable and verifiable by business domain experts. High level goals are further refined and operationalized as business rules. These rules are then transformed into a language where they can be executed as Web services and eventually incorporated into the design and composition of services.

Since the business process and the executable rules are integrated using SOA, the framework decouples business rules from the business processes that use them; i.e., the rules are treated as separate design and implementation artifacts. Should there be changes to the business rules, service systems that implement them can respond to the required change rapidly – thus providing better business process flexibility. Furthermore, with the separation of business rules from processes, the business rules are no longer hidden or hard coded in business processes or even in application code. Separating the business rules also allow organizations to manage them explicitly, for example, by employing a rule repository that contains and manages all rules used by their systems.

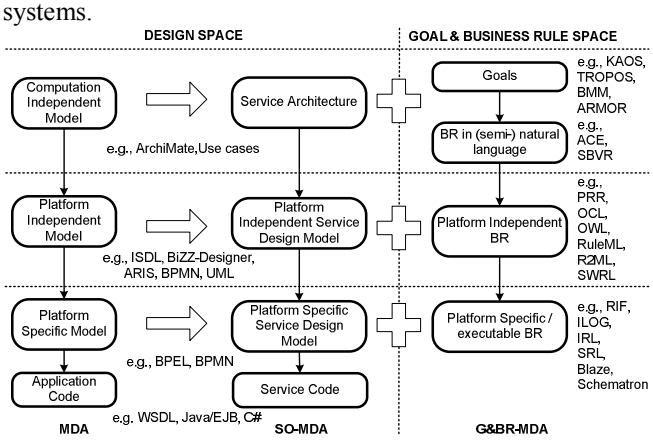

Figure 3. A model-driven view on the integration of service design enhanced with goals/business rules and their specification languages

B. Framework Instance

For the time being, we created a prototype reifying the generic framework discussed earlier through the selection of specific technologies to specify the requirements at the CIM, PIM, and PSM levels, including their transformations. Whereas the framework is generic, the technologies are of our particular choice. Other types of technologies are also possible. Currently, we implemented the generic framework using the technologies shown in Figure 4.

Under the Design space, we use the *ArchiMate* framework [23] for modeling enterprise architectures, and its extension called *Architectural Modeling of Requirements* (ARMOR) [22] for modeling requirements of the architecture using goal-oriented requirements engineering at the CIM level. We use the *Interaction System Design Language* (ISDL) [17] to model the behavior of the Mediator at the PIM level. At the PSM level, we use the *Business Process Execution Language* (BPEL) [9] to provide an executable version of the Mediator.

Under the G&BR space, we use a controlled language called *Attempto Controlled English* (ACE) [5][12] to specify the business rules in near-natural English at the CIM level. We transform ACE into an XML-based rule specification using *Rule Markup Language* (RuleML) [29]. An XML-based rule specification at the PIM layer, such as RuleML, allows for better rule interoperability should there be a need to migrate to another rule specification. We select RuleML as it is XML-based and is mature enough so that a number of transformation solutions have been proposed (albeit preliminary). We then transform RuleML into an executable form using *Java Expert System Shell* (Jess) [18] at the PSM level. The Jess rules are then exposed as a Web service by wrapping them in Java code and deploying them in the Jess rule engine. During design time, the Jess rule is incorporated into the behavior model of the Mediator in ISDL. At runtime, the BPEL version of the ISDL Mediator model invokes the rule deployed in the Jess rule engine.

The transformation of a low-level sub-goal into a business rule (i.e. from ARMOR to ACE) is currently done manually. ARMOR is, however, supported by a commercially available editor [7] while the proponents of ACE provide a web client [5] to correctly construct ACE sentences. At the moment, we are using the work of Bahr [6] to transform ACE sentences to RuleML automatically. For the time being, we developed our own prototype for transforming RuleML to Jess using *Extensible Stylesheet Language Transformation* (XSLT) based on the earlier work of Tabet [33].

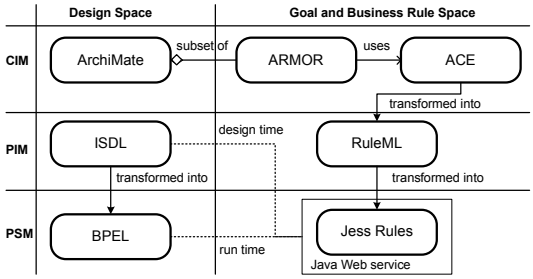

Figure 4. Technology instance of the framework

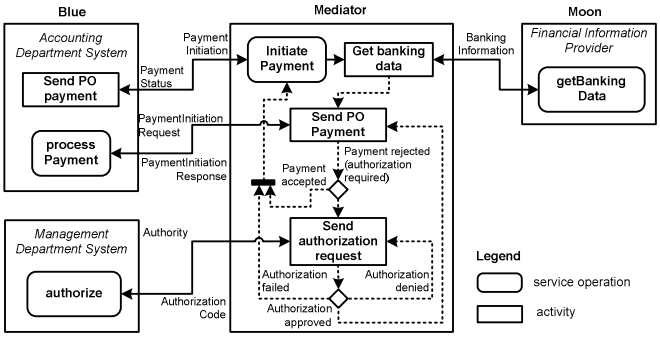

Figure 5. SWS Challenge Payment Problem Scenario (adopted from [32])

IV. SWS Challenge Payment Problem Scenario

We introduce the *Semantic Web Service (SWS) Challenge Payment Problem Scenario* [32] as an illustrative case to demonstrate the flexibility of our integration solution. The scenario represents an example of a *purchase order payment initiation procedure*. We first introduce the case description and then the solution.

A. Case Description

The scenario describes the interaction of two fictional companies, *Blue* and *Moon*, to process purchase order (PO) payments as shown in Figure 5. In this scenario, the Mediator acts as an integration platform between the *Accounting Department System* (ADS) and *Management Department System* (MDS) of Blue and the *Financial Information Provider* (FIP) of Moon.

Using the `initiatePayment` operation, Blue initiates request for payment of PO by sending a message formatted according to *UNIFI ISO 20022 Payments Standard Initiation - Customer Credit Transfer Initiation v02* to the Mediator. The Mediator thereafter retrieves Blue's banking information from Moon's FIP using the `getBankData` operation. Moon uses its own format to return banking information. The results of these operations are combined to form the complete payment initiation thereafter invoking the `processPayment` operation of Blue's ADS.

Blue requires authorization when the PO amount is greater than €2000; otherwise, payment is processed immediately. Thus, either a response of `AUTHREQUIRED` or `PROCESSED` is returned from the `processPayment` operation, respectively. This PO limit is handled by Blue's ADS and may change any time. When authorization is required, the Mediator will need to make subsequent calls to Blue's MDS using the `authorize` operation inserting an appropriate *Authority* value for each call. An Authority is a Blue employer who can authorize a certain range PO amount as shown in Table 1. Blue also requires that the least senior Authority should be called first (i.e., the Authority who can decide on payments up to the maximum designated amount). The Mediator calls the `authorize` operation iteratively until the response is `ACCEPTED` and the necessary `authorization code` is returned. A `DENIED` response is returned when a PO amount is not within the range of the current Authority. The `authorize` operation then has to be called again, this time

TABLE I. AUTHORITIES AND THEIR DESIGNATED AMOUNTS

Order of invocation	Authority	Designated Amount in € (maximum)
1st	Jackie Brown	2 000
2nd	Cathy Johnson	3 000
3rd	Arnold Black	10 000
4th	Peter Petrelli	50 000

referring to a higher Authority. For example, if the PO amount is € 2500, Jackie Brown will have to be called first. However, since she is not authorized with that amount, a DENIED response is returned. The next Authority, Cathy Johnson, will be called next. This time, since the amount is below the designated amount, an ACCEPTED response and an authorization code are returned.

A response of FAILED occurs when the response is still DENIED and all Authorities have been iterated over. In all of the message exchanges between Blue and the Mediator, the same message format is used, up until the payment status message is sent. At this time, the Mediator uses the *UNIFI ISO 20022 Payments Standard Initiation - Payment Status Report V02* format to create the payment status message. This message is sent back as the response to the initiatePayment operation. The payment status code can either be PI_ACCEPTED when Blue has accepted payment or PI_REFUSED_AUTH_FAILED when request for payment and authorization has failed.

B. Solving the Scenario

We have provided a general discussion of our approach in [3][4]. This paper focuses on the separation of business rules from the business process. As part of the integration framework for service integration [21], our approach comprises an integration methodology shown in Figure 6.

The integration methodology has *seven* steps:

Step 1: The first step requires abstracting from the platform independent information and behavior models of the services specified in the WSDLs of both Blue and Moon. The *behavior models* are represented using ISDL while the *information models* are specified using a combination of UML class diagrams (for visualization) and Java (for execution). This step is *automated* using the WSDL import function of the Grizzle tool [17] which provides an integrated editor and simulator for ISDL, and uses Java to represent and execute operation parameter constraints.

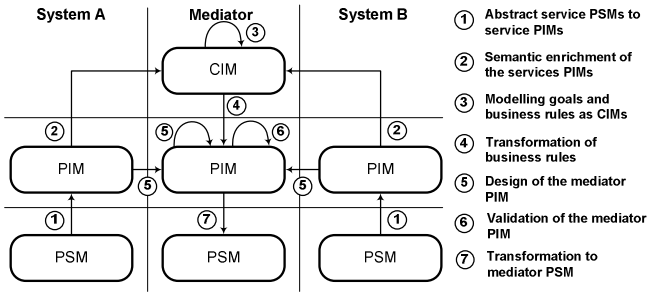

Figure 6. Methodology for service integration

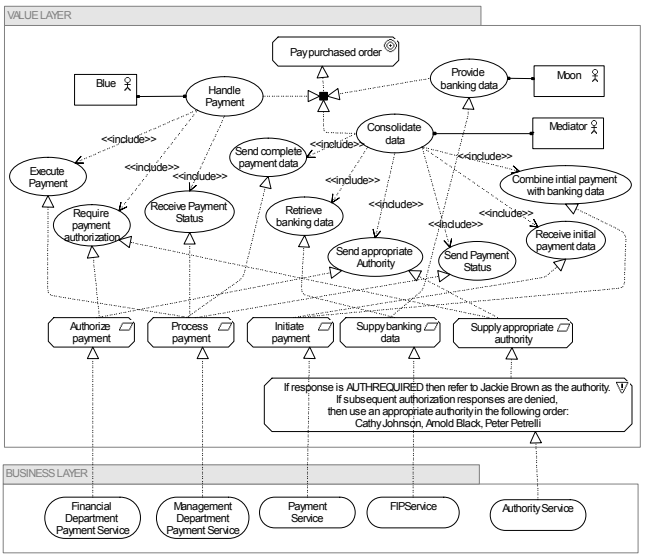

Figure 7. Goal model of the integration in ARMOR + ArchiMate (partial)

Step 2: As WSDLs do not define interaction protocols (i.e. the ordering of the message exchange between Blue and Moon) and also lacks semantic denotation of the message elements, this step thus aims to semantically enrich the behavior and information models. Currently, this is a *manual* process that involves interpreting the case description to define the relations between operations, and the semantic equivalence between the message elements by defining new classes, properties and relations among classes. Steps 1 and 2 allow us to first determine the available services that need to be used for integrating Blue and Moon (bottom-up).

Step 3: The next step aims to capture the motivation or rationale of the integration where business requirements are first specified as goals using ARMOR. These goals are then refined further into sub-goals (through use cases) until they can be assigned to a service that can fulfill them. Modeling goals in ARMOR can be done using BizzDesign tool [7]. For example, the main goal of the integration, which is to "*Pay purchased order*", is refined into use cases and finally as one of the sub-goals *Initiate Payment*. This sub-goal, in turn, is assigned to the *Payment Service* which when invoked satisfies the sub-goal and eventually the main goal. The integration's goal model in ARMOR is shown in Figure 7.

However, there are no services from either Blue or Moon that can fulfill the goal *supply an appropriate authority*; the Mediator will thus have to implement this goal. Our approach has been to refine this goal into a business rule in ARMOR and, for now, state simply the business rule in plain English as shown in Listing 1. This business rule will have to be made executable to satisfy the goal.

Step 4: This step requires a series of transformations from plain English to ACE (at the CIM level), to RuleML (at the PIM level), and finally to Jess (at the PSM level). This is done so that the business rules can be made executable and be incorporated into the design of the Mediator PIM. As ARMOR does not currently support the automatic

```
If the response is AUTHREQUIRED then refer to
Jackie Brown as the authority. If subsequent
authorization responses are DENIED, then use an
appropriate authority in the following order: Cathy
Johnson, Arnold Black, and Peter Petrelli.
```

Listing 1. Business rule in ARMOR

transformation of a business rule into a controlled language such as ACE, the ARMOR to ACE transformation is done *manually*. To do this, we separate the business rule in Listing 1 into 4 valid IF…THEN ACE sentences (see Listing 2).

The business rule in ACE will need to be transformed into RuleML. An example of the equivalent RuleML specification of *Authority1* is shown in Listing 3. For now, we use the work of Bahr [6] to *automatically* transform ACE to RuleML. Steps 3 and 4 allow us to model the integration according to the requirements specified by the case description, and finally matching the requirements with the available services later on (top-down). To execute the business rules, the RuleML specifications need to be transformed into Jess and deployed as a Web service. Listing 4 gives an example of the equivalent Jess code of *Authority1*.

Currently, additional coding needs to be done at the PSM level to add platform-specific code and to deploy Jess as a Web service. For example, a Jess-specific code such as the `store()` function which returns an appropriate Authority to the Java environment must be added as shown in Listing 4. The 4 Jess rules are then collected into one batch file. We create a WSDL specification that exposes the batch file as a Web service with the operation `getNextAuthority` taking the *response code* (i.e. `AUTHREQUIRED` or `DENIED`) as the input parameter and the *first and last names* of the *Authority* as the output parameters. We use Java to deploy the Jess rules as an Apache Axis2 Web service in Apache Tomcat. One advantage of Jess is that it is tightly integrated with Java so that we can simply treat Jess rules as simple Java objects, and vice versa.

Step 5: As we have all the services needed, we now design the *information* and *behavior* models of the Mediator at the PIM level. The information model is constructed from the union of the information models of Blue and Moon. The behavior model requires the definition of (i) the services provided and requested by the Mediator, (ii) the composition of these services by relating the operations of the services, and (iii) the data transformations among the parameters of the operations. The design process is done *manually* with the help of Grizzle. The semantic mapping between data elements is specified in a *domain- specific language* (DSL)

```
Authority1:
If the n:response is a:authrequired then the
a:next n:authority is p:Jackie-Brown.
Authority2:
If the n:response is a:denied and the
a:previous n:authority is p:Jackie-Brown then
the a:next n:authority is p:Cathy-Johnson.
Authority3:
If the n:response is a:denied and the
a:previous n:authority is p:Cathy-Johnson then
the a:next n:authority is p:Arnold-Black.
Authority4:
If the n:response is a:denied and the
a:previous n:authority is p:Arnold-Black then
the a:next n:authority is p:Peter-Petrelli.
```

Listing 2. Business rule in ACE

```
<Forall>
  <Var>A</Var>
  <Var>B</Var>
  <Var>C</Var>
  <Implies>
    <And>
     <Atom>
       <Rel>
        property
       </Rel>
       <Var>A</Var>
       <Ind>
        Authrequired
       </Ind>
       <Ind>pos</Ind>
     </Atom>
     <Atom>
       <Rel>predicate</Rel>
       <Var>B</Var>
       <Ind>be</Ind>
       <Var>C</Var>
       <Var>A</Var>
     </Atom>
     <Atom>
       <Rel>object</Rel>
       <Var>C</Var>
       <Ind>response</Ind>
       <Ind>countable</Ind>
       <Ind>na</Ind>
       <Ind>eq</Ind>
       <Data>1</Data>
     </Atom>
    </And>
    <Exists>
      <Var>D</Var>
      <Var>E</Var>
      <And>
        <Atom>
          <Rel>predicate</Rel>
          <Var>D</Var>
          <Ind>be</Ind>
          <Var>E</Var>
          <Var>named
            ('Jackie-Brown')
          </Var>
        </Atom>
        <Atom>
          <Rel>property</Rel>
          <Var>G</Var>
          <Ind>next</Ind>
          <Ind>pos</Ind>
        </Atom>
        <Atom>
          <Rel>object</Rel>
          <Var>E</Var>
          <Ind>authority</Ind>
          <Ind>countable</Ind>
          <Ind>na</Ind>
          <Ind>eq</Ind>
          <Data>1</Data>
        </Atom>
      </And>
    </Exists>
  </Implies>
</Forall>
```

Listing 3. Authority1 in RuleML

[21]. The complete Mediator PIM in ISDL is shown in Figure 8. Notice that, at this point, the execution for handling Authorities (i.e. using the `getNextAuthorty` operation) is no longer contained in the ISDL model of the Mediator PIM as this is now maintained separately as a Web service. The business process thus stays isolated from the business rules that constrain it.

Step 6: This step allows designers to validate the sequence of invocations, the messages being passed, and the overall business logic by *simulating* the Mediator PIM model in ISDL using a tool called *Sizzle* which forms part of the Grizzle editor.

Step 7: This last step requires the *automatic* transformation of the ISDL Mediator model into BPEL for deployment and production use.

V. TOWARDS PROCESS FLEXIBILITY

This section discusses the flexibility of our solution in relation to existing business process flexibility research.

A. Achieving Flexibility

Our solution proposes a separation of business rules from the business process of the Mediator. We do this by specifying the integration requirements that are most likely to change over time in terms of business rules. These business rules are first specified in near-natural language and later transformed into an executable specification. The executable business rule is then exposed as a Web service and added to the more stable parts of the business process. Since the dynamic parts of the payment process are separated from the more stable ones, a change in either of them does not affect the other adversely. For example, should existing business rules change (e.g., changing the order of the invocation of the Authorities, adding a new, or deleting an existing Authority), the ISDL specification of the Mediator need not be affected as the change is confined to the

```
defrule Authority1
?code <- (response authrequired)
 =>(assert (authority Jackie Brown))
   (store AUTH-FIRST-NAME Jackie)
   (store AUTH-LAST-NAME Brown)
   (retract ?code))
```

Listing 4. Authority1 in Jess

specifications under the G&BR space. However, we note an important assumption here: the message parameters of the `getAuthority` operation should remain the same. Conversely, should there be changes to the activities related to how the payment process is done (e.g. changing the order of invocations between Blue or Moon, removing some or adding new the services), the change is confined only to the specifications under the Design Space.

Since business rules are now treated as separate artifacts, neither are they part of the behavior model of the Mediator nor are they hidden and scattered in some application code. For example, we could have implemented the requirement to assign an appropriate Authority within the behavior model of the Mediator in ISDL itself as shown in Figure 9. However, by looking at the ISDL model alone, it is not evident that such functionality exists. This is because the business rule is implemented in the DSL which is used to transform data elements between message exchanges. One of the many functions that need to be created is the `createAuthList()` that initializes a set of Authorities as in Listing 5. Changes to the rules will thus have to be done in the DSL which may not be that intuitive.

Business rules can be specified in a form that is intuitively understandable when expressed in a near-natural language (*c.f.* Section II.C). Thus, treating business rules explicitly also allows them to be understood, managed, and validated better by business domain experts who do not (want to) have technical knowledge about implementation details. Finally, since rules are exposed as Web services,

```
mapping createAuthList{
    target util:List authList{}
    expressions{
        list       = createList();
        firstName1 = "Jackie"; lastName1 = "Brown";
        firstName2 = "Cathy";  lastName2 = "Johnson";
        firstName3 = "Arnold"; lastName3 = "Black";
        firstName4 = "Peter";  lastName4 = "Petrelli";
        authority1 = createAuth(firstName1,lastName1);
        authority2 = createAuth(firstName2,lastName2);
        authority3 = createAuth(firstName3,lastName3);
        authority4 = createAuth(firstName4,lastName4)
        authList   = addToList(list, authority1);
        authList   = addToList(list, authority2);
        authList   = addToList(list, authority3);
        authList   = addToList(list, authority4);}}
```

Listing 5. Coding Authorities in DSL

other business processes should be able to reuse them.

B. Positioning the Solution

Positioning our approach with the taxonomy proposed by Regev, et al. [27], we can say that at the abstract level of change, our solution supports process type change. The subjects of change include all perspectives: functional, organizational, behavioral, information and operational. As to the properties of change, we support incremental, permanent, deferred and planned change. Although the following discussion uses the specific technologies described in the framework instance (*c.f.* Section III.B) in relation to the case scenario, the same assessment can be made even when different technologies are used.

For a graphical depiction of the flexibility criteria that our solution supports based on the taxonomy of Regev, et al., please refer back to the italicized text in Figure 1.

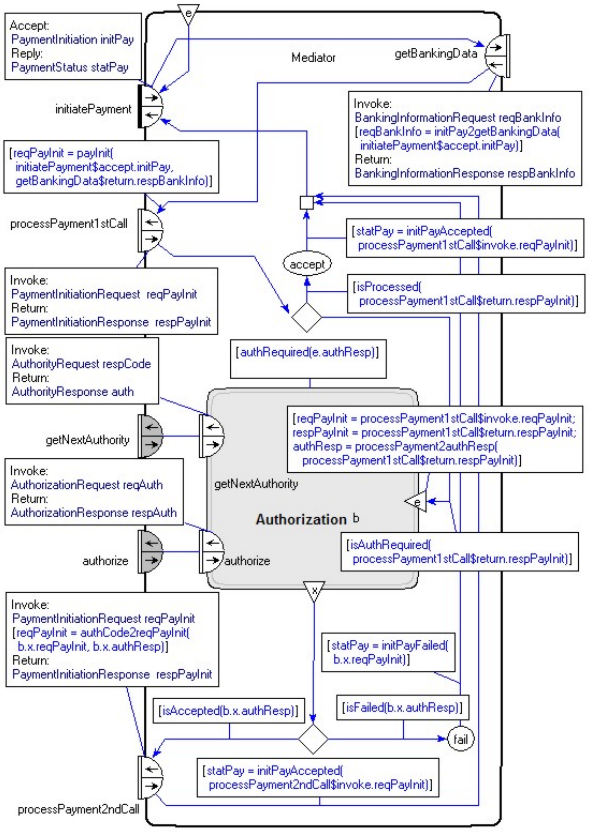

Figure 8. Mediator PIM with separated business rules in ISDL

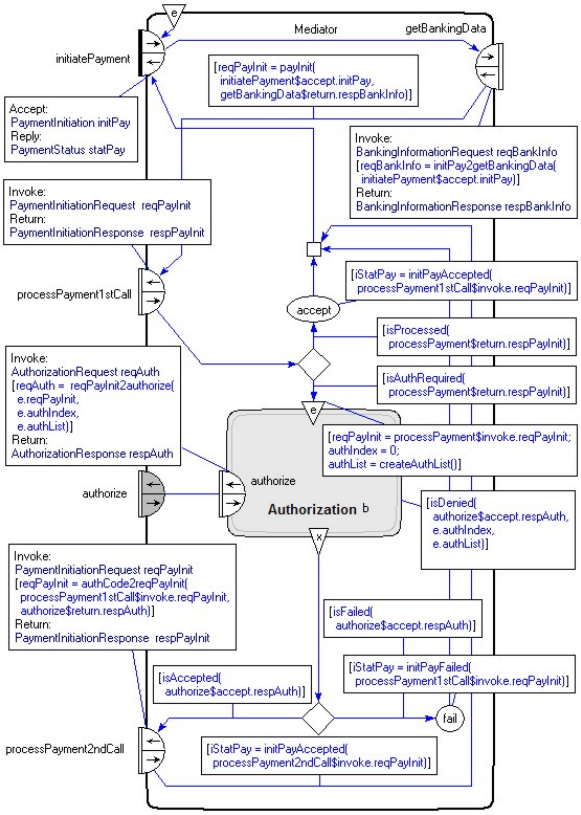

Figure 9. Mediator PIM with embedded Authority logic in ISDL

1) Abstraction Level of Change

Our solution handles changes at the process type level. Should there be changes to the requirements of the integration, the goal model at the CIM level has to change first; however, the entire goal model need not change assuming the main goal of the integration (in our case, "*Pay purchase order*") does not change. Following the principles of model transformations, changes to the goal model at the CIM level should also be propagated to the PIM and PSM levels within the Design space, G&BR space, or both. If business rules are created or changed, then the change should be reflected only in the affected specification at design time. For example, if the change is isolated to a change in the list of Authorities such as when a new Authority needs to be added, the change must be introduced first in the goal model and then translated into the different rule specifications (i.e. from ACE, to RuleML, and to Jess). However, changes to either specification in the Design or G&BR space require redeployment in order for the change to take effect.

2) Subject of Change

As we take a goal-oriented approach to service mediation, our solution can handle changes to the functional perspective of the subject of change. These goals are explicitly modeled using ARMOR. Furthermore, a targeted change can also be done in terms of the operational perspective by making changes to the process activities of the Mediator. These activities are modeled in ISDL using the operation calls and operation executions constructs.

Our solution supports the behavioral perspective of the subject of change. The Mediator itself specifies the order of invocation between operations through its behavioral model, which in turn describes the imperative or declarative constraints. In our solution, the ISDL behavior model shows the imperative constraints by specifying the order of operation invocations (e.g. the operation call `getBankingData` will never be enabled unless the `initiatePayment` is enabled). The declarative constraints are specified in terms of business rules (e.g., `Peter Petrelli` will never authorize an amount of € `10000`).

The Mediator provides semantic matching between incompatible message elements of collaborating systems which is specified in an information model. In our solution, this is expressed using ISDL's *information type* construct and the domain specific language. Our solution thus supports the information perspective of the subject of change.

Finally, the goal model provides a high-level view using the stakeholder construct to model the organizational perspective of the subject of change. Here, relevant stakeholders are modeled including their individuals goals. In our solution, the integration's goal model depicts three actors: Blue, Moon and the Mediator modeled in ARMOR.

3) Properties of Change

As to the extent of change, our solution supports incremental change. Designers of the integration solution do not need to redesign the entire business process to effect a single change. They can build on the existing specifications. For example, if the order of invoking the Authorities needs to be changed, part of the goal model that specifies the business rule will have to be changed. This then requires rewriting the ACE sentences, generating the necessary RuleML and Jess specifications and then redeploying the changes. This does not require redesigning the entire goal model, existing rule-based specification can be preserved, and the ISDL and BPEL models need not be affected. However, the rules will have to be redeployed thereafter.

As our solution does not support runtime changes, only permanent change can be done. This means that should a change be introduced, it should first be reflected in the process type, made permanent, and then redeployed in order to effect the change. The permanent change can either occur at the Design space or at the G&BR space.

As to the swiftness of change, our solution supports the deferred type. The change can only take effect when a new process is created as our solution cannot handle process instance change. This further implies that the execution of all currently running instances needs to complete first before new changes can take effect. Finally, our approach can so far handle planned changes. Ad hoc changes to handle exceptions in the process are possible; however, they have to be modeled explicitly at design time at the process type level. For the most part, should changes need to be made to the process, planned changes are required.

C. Imperative vs. Declarative Approach: How to Decide?

Over the years, designing processes that truly meet the highly dynamic demands of businesses has been a difficult research challenge; this is especially true when the issue of flexibility is in question. Up to this time, there is a continuous debate as to which manner of specifying business processes to achieve flexibility is best: an *imperative* or *declarative* approach. We discuss existing work in this area, and position our work accordingly.

Schonenberg, et al. [30] propose a distinction between an imperative versus a declarative approach. An *imperative approach* focuses on the explicit and precise definition of relevant activities that a process has to perform including their exact order of execution. The constraints which dictate the activities' execution order is explicitly modeled using connectors and gateways that relate activities and include the condition of their execution. Thus, the approach largely describes *how* a process is to be done in a rigid manner. An example of a language that supports the imperative approach is BPEL.

A *declarative approach*, on the other hand, focuses on *what* the process should do rather than *how*. It starts from the assumption that everything should be allowed, unless explicitly forbidden. Thus, constraints (also called rules) inhibit some possible execution options while allowing those that do not violate the constraints. A rule-based language such as Jess is one example of a declarative language. In summary, to increase flexibility in an imperative approach, the number of execution paths must be *increased* by explicit modeling. Increasing flexibility in the declarative approach requires *reducing* the number of constraints [30]. Van der Aalst [1] argues that the trade-off between these approaches is difficult, in the sense that, while there are some aspects of a business process that require correct and desirable process execution, users involved in the execution do not want their

actions to be constrained by the process.

Heinl, et al. [14] identify several problems related to an imperative approach to business process design. One is that it is almost impossible to identify all constraints and execution paths a priori. If these paths are indeed identified, it is difficult to decide whether or not to put them in the business process. It is often the case that only the most relevant and often used paths are modeled explicitly in the business process. Furthermore, it is not ideal to prescribe all paths in detail as the execution of these paths may entirely depend on context or the execution state. Thus, a purely imperative approach is ideal only when processes are well-structured and do not require much flexibility. On the other hand, some authors advocate a more declarative approach to business process design to allow greater flexibility. With declarative approaches, the exact order of execution need not be determined a priori. Thus, the process is more flexible as the execution options are stated implicitly. This is ideal if such systems offer a wide selection of execution alternatives. However, a declarative approach makes no sense if the process is strictly procedural in nature. Furthermore, languages that support it tend to require more cognitive effort from the designer to understand the process especially if the number of constraints increases. This is in contrast with imperative languages where graphical modeling environments abound and thus are more readable [1].

Taking all these arguments into account, we propose a "*hybrid approach*" as another viable solution to business process flexibility. By hybrid, we mean the combination of imperative and declarative approaches – drawing strengths from each approach. As we have shown in our solution, a hybrid approach consists of specifying those aspects of the business processes that are most likely to change over time as business rules while keeping those that are more likely to remain stable as part of the business processes. This is done so that should there be frequent changes to the dynamic parts of the process, the rest does not necessarily have to be affected.

With our experience solving the SWS Challenge Payment Problem Scenario, the interaction between the services of Blue and Moon such as the initiation of payment, processing of payment, getting an appropriate banking data and authorization are modeled imperatively using ISDL as these are stable and require activities that the Mediator must choreograph. However, modeling the business logic behind the sending of an appropriate Authority from the Mediator to Blue's ADS imperatively would make the ISDL model significantly more complicated as the Mediator needs to remember the last Authority that was sent to Blue's ADS. This was therefore modeled declaratively.

Our solution was also verified by the organizers of the SWS Challenge Workshop. One of the verification challenges was that, at any instant, an Authority can be made to refuse any amount; e.g. Blue's ADS can always return a `DENIED` response for a request with `Jackie Brown` as the Authority. Our solution handled this requirement without any change to the specifications. This was possible since so long as the response from Blue's ADS remains `DENIED`, the Mediator just proceeds on to assign the next senior Authority until a response of `ACCEPTED` was returned (thus, it is still within the constraints of the declarative specification).

Furthermore, with the separation of the stable and dynamic parts of the process, this allows us to do *process migration* in the future. Of course, we will never able to predict if requirements will change or stay the same during design time. Should a requirement turn out to change often, we may migrate it to a business rule. Conversely, if some business rules do not fire frequently, they can be migrated to become part of the stable business process. Reflecting on our experience in solving the SWS Payment Problem Scenario, our earlier versions of the Mediator PIM (*cf.* Figure 10) required us to constantly update the rules related to assigning an appropriate Authority for authorization. This proved to be a tedious task since the SWS Challenge test bed was frequently giving varying results (e.g. some Authorities would not authorize any PO amount). As this requirement was unstable, we migrated this part of the business process to a business rule to accommodate future variations on the requirement.

D. Related Work

The approach of Rittgen [28] separates the overall business process into stable and flexible parts. The high-level interactions within and between organizations are first modeled in an Interaction Model. This model is then refined into Transaction Models where the stable and flexible process parts are specified. The stable parts are expressed as Collaboration Models while the flexible parts are expressed as business rules. In our approach, we specify the stable parts using ISDL but we take a model-driven approach to specifying the business rules at different abstraction levels.

Van Eindhoven, et al. [11] also discuss flexibility in terms of separating the business rules from the business process. They use variation points to analyze process variability. The variable and non-variable parts of the process are first identified. Variation points of the process are then isolated. An appropriate combination of workflow patterns is then used to model each variant in a variation point. Finally, these workflow patterns are then implemented as business rules. Our approach does not use variability analysis to determine which parts of the process can be expressed as business rules; however, we do use the GORE approach to decide which requirements must be treated dynamically, and hence be specified as business rules.

Another similar approach is proposed by Charfi and Mezini [36] where the composition logic is divided into a core process part and a business rule part. The latter is modularized allowing it to exist and evolve independently by implementing it as aspects using the BPEL extension AO4BPEL. They can then be dynamically (un)deployed during process interpretation time. Whereas their solution specifies the business rules at the PSM level (i.e., AO4BPEL), our solution specifies the business rule at the CIM, PSM, and PSM levels facilitating validation by non-technical experts. With our solution, incorporating the business rule, exposed as a Web service, can only be done at design time; whereas, their solution allows dynamic deployment of business rules.

VI. Conclusion and Future Work

This paper attempts to provide an approach to increase the flexibility of integration solutions in the context of service mediation by separating the more dynamic aspects of the requirements as business rules while keeping the more stable parts in the business process. To achieve this, we combine goal-driven approaches with model-driven and service-oriented techniques. Flexibility is achieved since the dynamic parts of the business process are separated from the more stable ones – a change between the either of them does not affect the other adversely. Using the process flexibility taxonomy of Regev, et al., we support process type change. The subjects of change include all perspectives: functional, organizational, behavioral, information and operational. As to the properties of change, we support incremental, permanent, deferred and planned change.

Our future work: Firstly, we want to improve the transformations between the rule specifications as they are currently limited. Although many well-established rule specifications are proposed, research work related to their transformations is still largely immature. Secondly, we are investigating the more expressive *Semantics of Business Vocabulary and Business Rules* (SBVR) as an alternative to ACE. Thirdly, we still need formal ways to validate whether the execution of the Mediator PSM does indeed achieve the overall goal of the integration. Fourthly, our solution uses a disparate set of tools. The state of the art does not provide a mature integrated development environment that leverages goals/rules, MDA, business process, and SOA in one place. We are thus interested in developing such an environment. Finally, since we treat business rules as separate artifacts, rule management is important particularly when the number of rules grows. We thus want to investigate the use of a *rule registry* which provides the ability to list all available rules, their relations, author, mutations, etc.

References

[1] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative Workflows: Balancing Between Flexibility and Support", Computer Science - Research and Development, vol. 23, pp. 99–113, 2009.

[2] A.I. Anton, "Goal-based requirements analysis", Proc. 2nd Int. Conf. on Requirements Engineering, pp. 136-144, 1996.

[3] C.H. Asuncion, D.A.C.Quartel, S. Pokraev, M.-E.Iacob, and M.J. van Sinderen, "Applying Goal-Oriented and Model-Driven Approaches to Solve The Payment Problem Scenario". Proc. 8th Semantic Web service (SWS) Challenge Workshop, The Netherlands, 2009. http://eprints.eemcs.utwente.nl/17831/

[4] C.H. Asuncion, "Goal-Driven Service Mediation Solution," M.S. thesis, University of Twente, 2009. http://purl.org/utwente/e59477.

[5] Attempto Controlled English. http://attempto.ifi.uzh.ch/ape.

[6] Bahr, P. (2008). "The ACE2RRML Web service ". Retrieved August 4, 2009 from http://www.paba .info/?q=pub/ace 2rrml.

[7] BiZZDesign.http://www.bizzdesign.nl/joomla/products/architect.html

[8] Business Rules Group, "Defining business rules – what are they really? Business Rules Group". Retrieved July 7, 2009, from http:// www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed. pdf.

[9] Business Process Execution Language (BPEL). http://docs.oasis-open.org./wsbpel/2.0/wsbpel-v2.0.pdf

[10] Y. Charalabidis, G. Gionis, K.M. Hermann, C. Martinez; "Enterprise Interoperability Research Roadmap, Version 5.0", 2008.

[11] T. van Eijndhoven, M.-E. Iacob, and M. L. Ponisio, "Achieving Business Process Flexibility with Business Rules", Proc. 12th Int. EDOC Conf., pp. 95-104, 2008. IEEE Press.

[12] N. Fuchs, U. Schwertel, and R. Schwitter, "Attempto Controlled English (ACE) Language Manual Version 3.0", Institut für Informatik der Universität Zürich, 1999.

[13] B. von Halle, "Business Rules Applied: Business Better Systems Using the Business Rules Approach", 2002, John Wiley & Sons.

[14] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke, "A Comprehensive Approach to Flexibility in Workflow Management Systems". Proc. Int. Joint Conf. on Work Activities Coordination and Collaboration, pp. 79-88, 1999. ACM Press.

[15] M.-E. Iacob, D. Rothengatter, and J. van Hillegersberg, "A Health-care Application of Goal-driven Software Design". Proc. Applied Medical Informatics, vol 24, pp. 12-33, 2009.

[16] M.-E. Iacob and H. Jonkers, "A Model-Driven Perspective on the Rule-Based Specification of Services", Proc. 12th Int. EDOC Conf., pp.75-84. 2008, IEEE Press.

[17] Interaction Systems Design Langauge. http://ctit.isdl.utwente.nl

[18] E. Friedman-Hill, "Jess in Action: Rule-Based Systems in Java", Manning Publications Co, 2003.

[19] M. Jackson, "Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices, 1995. ACM Press.

[20] V. Kasi and X. Tang, "Design Attributes and Performance Outcomes: A Framework for Comparing Business Processes". Proc. 8th Annual Conf. of the Southern Assoc. of Info. Sys., pp. 226–232, 2005.

[21] D.A.C Quartel, S. Pokraev, R. Mantovaneli Pessoa, and M. van Sinderen, "Model-Driven Development of a Mediation Service". Proc. 12th Int. EDOC Conf., pp. 117–126, 2008. IEEE Press.

[22] D.A.C. Quartel, W. Engelsman, H. Jonkers, and M. van Sinderen, "A Goal-Oriented Requirements Modelling Language for Enterprise Architecture", Proc. 13th Int. EDOC Conf., pp. 1-11, 2009. IEEE.

[23] M. Lankhorst, et al. Enterprise Architecture at Work, 2008, Springer.

[24] A. van Lamsweerde, "Requirements Engineering – From System Goals to UML Models to Software Specification", Wiley, 2009

[25] A. Lapouchnian, "Goal-oriented Requirements Engineering: An Overview of the Current Research", 2005. University of Toronto.

[26] O. Nicolae, and G. Wagner, "Verbalizing R2ML Rules into SBVR". Proc. 10th Int. Symp. on Symbolic and Numeric Algorithms for Scientific Computing, pp. 265-272, 2008, IEEE.

[27] G. Regev, and A. Wegmann, "Taxonomy of Flexibility in Business Processes". Proc. 7th Workshop on Business Process Modeling, Development and Support, 2006.

[28] P. Rittgen, "Supporting Planned and Ad-Hoc Changes of Business Process Modelling". Proc. 7th Workshop on Business Process Modeling, Development and Support, pp. 105-111, 2006.

[29] Rule Markup Initiative, http://ruleml.org/.

[30] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, and W.M.P. van der Aalst. Towards a taxonomy of process flexibility. Proc. CAiSE Forum, pp. 81–84, 2008.

[31] M.J. van Sinderen, "Challenges and Solutions in Enterprise Computing", Enterprise Info. Sys., vol. 4, pp. 341-346, 2008.

[32] SWS Challenge Payment Problem Scenario, http://sws-challenge.org/wiki/index.php/Scenario:_Payment_Problem.

[33] S. Tabet, "RuleML and Jess" Retrieved August 24, 2009 from http://home.comcast.net/~stabet/page3.html.

[34] T. Taveter, and G.Wagner, "Agent-Oriented Enterprise Modeling Based on Business Rules", Proc. 20th Int. Conf. on Conceptual Modeling, pp. 527-540, 2001.

[35] G. Wagner, "How to Design a General Rule Markup Language? Proc. 1st Workshop on XML Technologies for the Semantic Web, 2002.

[36] A. Charfi and M. Mezini, "Hybrid web service composition: business processes meet business rules". Proc. ICSOC, pp. 30–38, 2004.ACM.