# Applying goal-oriented and model-driven approaches to solve the Payment Problem Scenario

Camlon H. Asuncion[1], Dick A.C. Quartel[2], Stanislav Pokraev[2],
Maria-Eugenia Iacob[3], and Marten J. van Sinderen[1]

[1]Center for Telematics and Information Technology, [3]School of Management and Governance, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{c.h.asuncion[1], m.j.vansinderen[1], m.e.iacob[3]}@utwente.nl,
[2]Novay, P. O. Box 589, 7500 AN Enschede, The Netherlands
{stanislav.pokraev, dick.quartel}@novay.nl[2]

**Abstract.** Motivated by the objective to provide an improved participation of business domain experts in the design of service-oriented integration solutions, we extend our previous work on using the COSMO methodology for service mediation by introducing a goal-oriented approach to requirements engineering. With this approach, business requirements including the motivations behind the mediation solution are better understood, specified, and aligned with their technical implementations. We use the Payment Problem Scenario of the SWS Challenge to illustrate the extension.

**Keywords:** SOA, MDA, service integration, model transformations, goal modeling, business rules, rule transformations.

## 1 Introduction

Designing integration solutions has always been traditionally technology-driven where Information Technology (IT) specialists do most of the job of building solutions based on such technologies as WSDL, BPEL, etc. Business domain experts are merely consulted at the early stage of requirements elicitation. Even if business domain experts do get involved in the collaboration design, they will have to contend with learning such technologies and sophisticated tools which may be too technical for them [1]. Furthermore, infusing the participation of business domain experts in integrating service-based systems is often difficult because technology standards (e.g. WSDL) are inherently defined with so little business semantics that business people do not understand them. For business people to match and compose services at the technology level is a daunting task. It is difficult therefore to compose the integration solution using these standards at the business level [2].

This paper investigates whether a *goal-driven* approach can be used by business domain experts in specifying the requirements of the mediation solution. Goals are high-level objectives of a business, organization, or system. They capture the reasons *why* a system is needed and guide decisions at various levels within the enterprise [3]. They elaborate system requirements, and 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 [4]. As such, we

argue that business domain experts are at the best position to describe the requirements of the service mediation solution through goals.

We investigate the use of *model-driven techniques* to transform these abstract goals into technology-specific implementations. Model-driven development allows us to raise the problem and solution analyses spaces to a level of abstraction that is technology-independent, more suited for business-level analysis [5]. In the process, we show how goals can be refined into *business rules* to constrain the behaviour of the Mediator. The Business Rules Group defines a business rule as "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business [6]". We believe that model-driven techniques should be able to give added flexibility to our solution as it treats goals, business rules, their design specifications, and technical implementations as separate concerns with the resulting artifacts maintainable and reusable.

We demonstrate the combination of goal-driven and model-driven approaches by extending our previous work [7] and applying it in solving the Payment Problem Scenario of the SWS Challenge [8].

The remainder of this paper is structured as follows: Section 2 briefly presents our integration framework. Section 3 shows how we have extended the integration method. Section 4 shows how we used the extended integration method to solve the Payment Problem Scenario of the SWS Challenge. Section 5 presents the conclusions and future work.

## 2 Integration framework

We define *service mediation* as "to act as an intermediary agent in reconciling differences between services of two or more systems." 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 Fig. 1, *Mediator M* offers a service that matches the requested service *S1* of system *A* by composing services *S3* and *S4* offered by system *B*.

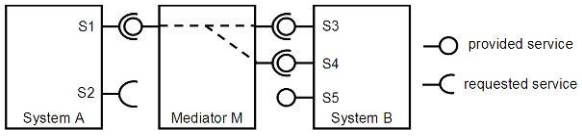


**Fig. 1.** Service mediation as service composition

We assume that collaborating enterprises expose their functionalities as services (i.e. through Web services). These services are exposed publicly, i.e. they can be accessed by other external systems. We also assume that these services cannot be

changed and are therefore fixed. This also implies that these services already exist, and that their services were not designed to match beforehand.

To support the design, implementation and validation of service mediation, we have developed an *integration framework*. It consists of the following elements: (i) a *conceptual framework* for modeling and reasoning about services, called *COnceptual Service MOdelling* (COSMO)[9], (ii) a set of *languages*[10] used to express service models using COSMO, (iii) *techniques* to analyze the interoperability[11] and conformance[12], (iv) *transformations*[13] from platform-independent service models to platform-specific service models, and vice versa, (v) *tools*[14] supporting the editing, analysis and transformation of service models, and (vi) a *method*[7] for developing the service Mediator.

## 3 Extending the integration method

This paper extends our previous integration method by introducing *goal models* as Computation Independent Models (CIMs) in specifying service mediation requirements. We argue that doing so gives business domain experts the opportunity to better understand, specify, and validate integration requirements without having to deal with their technical implementations. Figure 2 shows the proposed extended integration method. For brevity, we consider only two systems, but the same steps apply to the case of multiple systems.

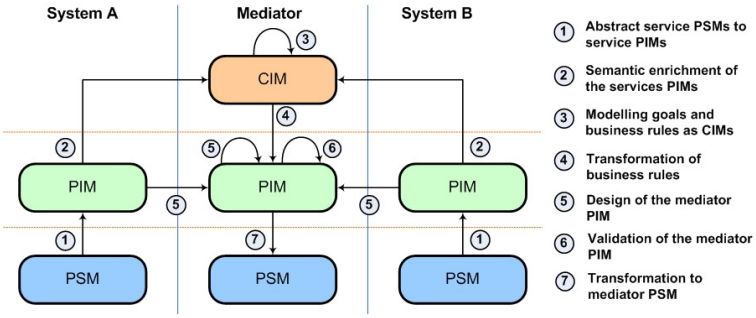


**Fig. 2.** Extending the service mediation method to the CIM layer

The services of Blue and Moon systems are described at implementation (technology) level (using WSDL). Our method starts with abstracting these service descriptions from all implementation-specific information. Such information may unnecessarily complicate the design of an integration solution, and therefore hinder the participation of business domain experts that do not (want to) know how integration requirements are implemented by the means of some technology. In terms of Model Driven Architecture (MDA), this means that we transform the service Platform Specific Models (PSMs) of Blue and Moon to their respective service Platform Independent Models (PIMs)

In the second step, the service PIMs may be semantically enriched by adding information that could not be derived (automatically) from the service PSMs. For example, a WSDL description of a service may be complemented with some text

document that describes part of the service in natural language. Alternatively, interviews or even code inspection may be used to obtain information that is missing in the WSDL descriptions. The purpose of the semantic enrichment step is to make service models precise and complete, which in turn is a necessary condition to reason about and (semi-) automatically generate the Mediator.

Completing the first two steps is essentially bottom up where we first identify existing services to be mediated. As an extension to our integration method, we add two intermediary steps: specifying requirements as goal models at the CIM layer (third step), and transforming business rules derived from the goal models into an executable form (fourth step). These new steps are essentially top down where we use requirements specified through a goal model to dictate the composition of the service Mediator.

In the third step, we introduce goal modeling to engineer the requirements of the integration solution at the CIM layer. As Figure 2 shows, we stress that the goal model depicts only the motivations of the integration (represented by the *single* rounded square under the Mediator column). Although the collaborating enterprises may have their own goals, we are only concerned with the goals of the integration. This stresses the importance of the needed joint collaboration of the different business domain experts from participating enterprises to identify, specify and resolve conflicts of the integration goals at the business level. From the resulting goal model, we *manually map* which existing services identified in Step 1 can best satisfy the refined sub-goals. Furthermore, business rules that constrain some aspect of the integration may also be derived. If no existing service can realize or satisfy these business rules, they may have to be transformed into an executable form, exposed as a service (as described in the fourth step) and integrated into the behaviour model of the Mediator.

The fourth step transforms the business rules derived from the goal model into their equivalent rule specifications at the different layers of the MDA stack. At the CIM layer, we use a controlled language to specify the business rules in near-natural English language facilitating better validation of requirements from business domain experts. These rules are then transformed into an XML-based rule specification for added rule interoperability at the PIM layer. Finally, at the PSM layer, we transform these business rules into an executable form and expose them as a service to constrain the behaviour of the Mediator. This approach essentially separates the business rules of the Mediator from the business logic they constrain. We derive this concept from the work of Iacob, et al. [4].

The fifth step represents the design, validation and implementation of the integration solution of the Mediator PIM. The design step can be split into two parts: (i) the design of the information model and (ii) the design of the behaviour model of the Mediator. The purpose of the information model is to enable the compensation of the data mismatches, by defining a mapping between the information models of the Blue and Moon systems. The purpose of the behaviour model is to enable compensation of the process mismatches by defining a mapping between the services that are requested and the services that are provided by Blue and Moon.

The sixth step is used to analyse whether the proposed integration solution really enables the interoperability between Blue and Moon.

The seventh and final step transforms the mediator PIM to an implementation, the mediator PSM.

## 4 Applying the integration framework

This section presents the application of our framework to the Payment Problem Scenario of the SWS Challenge. For this purpose, the integration method is made concrete by deciding on, amongst others, the type of PSMs that are considered, the languages to be used at PIM level, the business rules specifications to be used, and related to these choices the transformations and analysis techniques that are needed.

**Step 1: Abstract from PSMs to PIMs**. This step derives the platform independent information and behaviour models of the services of Blue and Moon, which are specified by WSDL documents as shown in Figure 3. The behaviour models are represented using ISDL while the information models are specified using a combination of UML class diagrams (for visualization) and Java (for execution).

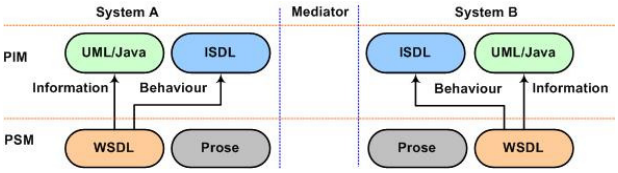


**Fig. 3.** Abstract from PSMs to PIMs

This step is automated using the WSDL import function of the *Grizzle* tool [15] which provides an integrated editor and simulator for ISDL, and uses Java to represent and execute operation parameter constraints. The WSDL import function enables a user to import a WSDL specification by providing the URL of this specification. Furthermore, the user may choose whether the web service should be considered from a *client* or *server* perspective. Accordingly, a behaviour model is generated that represents the user (client) or provider (server) role of the web service, in terms of *operation calls* or *operation executions*, respectively. For example, the `initiatePayment` operation exposed by Blue's will be abstracted as an operation call in ISDL (from the perspective of Blue) as it acts as a client to the Mediator.

In addition, an information model is generated consisting of Java classes that represent the information types that are referred to by the operations in the behaviour model. The transformation of WSDL to ISDL and Java is implemented using JAXB and JAX-WS ([16]). The EclipseUML tool ([17]) is used to visualize and manipulate the information model using UML class diagrams.

**Step 2: Semantic enrichment of PIMs**. The WSDL descriptions of the example scenario define the services that are provided by Blue, Moon and the Mediator, in terms of their operations and the types of the input and output messages of these operations. However, WSDL does not define the interaction protocols of the involved systems, i.e., the possible orderings of the operations. Therefore, to derive the complete PIMs of Moon and Blue, we have to use and interpret the provided textual case descriptions. This is a manual process. Firstly, the behaviour models that were generated in Step 1 are completed by defining relations between operations. These relations can be derived from the scenario description. Secondly, the information model may be enriched by interpreting the scenario description. A WSDL description defines the syntax of the messages that are exchanged, but does not provide

information about their semantics. This semantics can be made explicit by defining new classes, properties and relations among classes.

Semantically enriching the behaviour model of either Blue or Moon is quite easy seeing that each behaviour has at the most two operations. From Blue's Accounting Department System, we know from the case description that `initiatePayment` should be executed first before `processPayment`. This ordering should be reflected in the design of the Mediator.

**Step 3: Model goals and business rules at CIM**. This step aims to capture the *motivation* or *rationale* of the integration where business requirements are specified as goals or business rules without first describing how they are implemented by the underlying systems. We use a goal modeling language for enterprise architectures called *Architectural Modelling of Requirements* (ARMOR) [20] which extends the ArchiMate [20] modeling framework for enterprise architectures. Business domain experts manually model the requirements of the integration expressed as goal models using ARMOR. We adopt the methodology proposed by Mantovaneli Pessoa, et al. [18] for goal modeling in ARMOR which involves three main steps: (i) identification of stakeholders and the primary goal of integration, (ii) refinement of primary goals, and (iii) refinement of sub-goals into requirements.

Figure 4 shows the goal model of the integration in ARMOR packaged under the *value* layer which represents how an enterprise can offer value to the customers through its products and services. Deriving the primary goal of the integration and the relevant stakeholders can be done using goal identification techniques such as searching intentional keywords from the case description as proposed by [19]. In our case, we indicate that the goal of the integration is to "`Pay purchased order`" modeled as a *hard goal* in ARMOR which is a goal whose criteria for satisfaction are clear and precise. We also identify that the main stakeholders of the integration are Blue, Moon, and the Mediator.

Refinement of goals into sub-goals can be done using goal refinement techniques proposed by [19]. Goal refinement is a process of breaking down a higher-level goal into several lower-level goals in such a way that the latter contributes to the achievement or satisfaction of the former. The sub-goals are identified by asking "*how*" questions; that is, the sub-goals of G can be determined by asking "*How can G be satisfied*?"[19]. In our case, we refine the hard goal into a set of *use cases* in ARMOR which describe multiple or alternative sequences of interactions to satisfy goals. Use cases describe who the stakeholders of the integration are, including the goals that these stakeholders need to perform to contribute to the satisfaction of the main goal without first providing details as to how these goals are to be implemented by the underlying systems. For example, from the case description, we know that Blue's goal is to perform the actual processing of payments (depicted in Figure 4 as a use case with the name "`Handle payment`") modeled from the perspective of Blue. The `AND`-realization construct indicates that all use cases must be satisfied to achieve integration hard goal.

We can further refine a parent use case by asking "*how*" questions, and relating its child use case using the `<<include>>` relation which shows that the parent use case will be satisfied only when the sequence of interactions between the child use cases are accomplished. For example, the parent use case "`Handle payment`" can be refined further by asking how the handling of payment should take place. From the case

description, we can see that Blue accomplishes this by requiring authorization for purchase order amounts greater than €2000. We depict this as a child use case named "Require payment authorization".

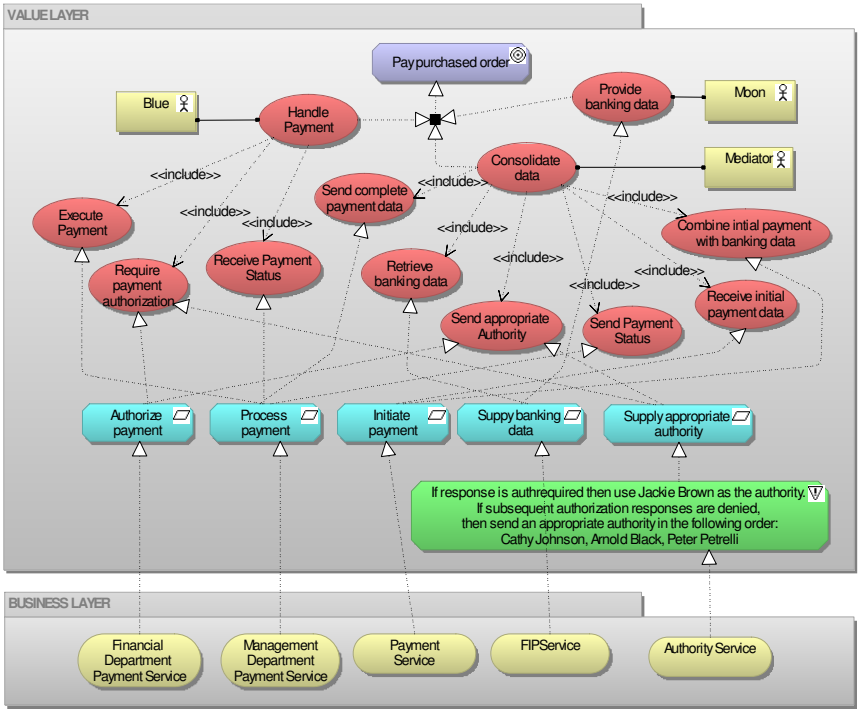


**Fig. 4.** Payment Problem goal model in ARMOR (entire ArchiMate model not shown)

In principle, goal decomposition stops if the sub-goal, which in our case is a child use case, can be assigned to a system(-to-be) that can satisfy its achievement. We describe this sub-goal using the *requirement* construct in ARMOR. Thus our next objective is to refine the child use cases into requirements in ARMOR. For example, the "Require payment authorization" is refined by the "Authorize payment" requirement. This means that the underlying system should support the "Authorize payment" requirement in order to satisfy the child use cases and ultimately the goal of the integration.

At this point, we are ready to manually match which existing services identified in Step 1 can be used to satisfy a given requirement in ARMOR. Figure 4 shows the mapping of the requirements in ARMOR to the existing services modeled in the *business layer* of ArchiMate using *realization* construct. For example, The "Authorize payment" requirement is satisfied and concretely realized by the Financial Department Payment Service using its authorize operation.

From the case description, the requirement "Supply appropriate Authority" cannot be realized by any service derived from Step 1, and thus we assume that this is

to be handled by the Mediator. However, the case requires that an appropriate Authority should be sent to Blue when an amount is greater than €2000. The first and last names of the Authority, with a certain designated amount, should be sent with subsequent invocation to the `processPayment` operation of the Moon's Financial Department Service. We can represent this as a business rule as it essentially constrains the business process. Figure 4 shows how we refined the requirement into a business rule whose description is stated in plain English and modeled using the *business rule* construct in ARMOR. Since no service current exists to realize this requirement, we need to make this business rule executable.

**Step 4: Transformation of business rules**. This step aims to transform and deploy the business rule derived from the goal model in Step 3 into an executable form and expose it as a service. Briefly, this requires the following: Firstly, we use a controlled language called *Attempto Controlled English* (ACE) [22] to specify the business rules in near-natural English language at the CIM layer. Secondly, we transform ACE into an XML-based rule specification using *Rule Markup Language* (RuleML) [23] for added rule interoperability at the PIM layer. Thirdly, we transform RuleML into an executable form using Java *Expert System Shell* (Jess) [24] at the PSM layer. Finally, 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 added into the behaviour 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. Figure 5 shows the resulting model transformation approach (c.f. [4]).

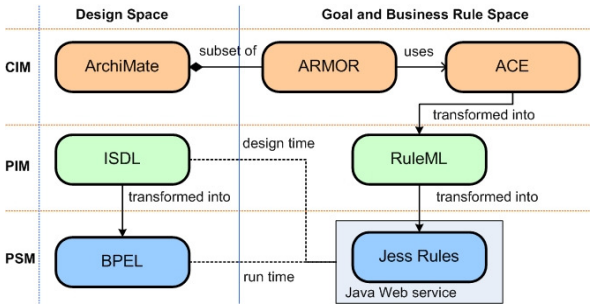


**Fig. 5.** Model transformation approach

As ARMOR does not currently support automated translation of its business rule construct to a controlled language, the business rule modeled in Figure 4 will have to be translated *manually* into valid ACE sentences. The *Attempto Parsing Engine Web Client*[27] can be used to correctly construct an ACE sentence. In our case, we break the business rule modeled in Figure 4 into four concrete `IF…THEN` ACE sentences. The first of these rules is shown in Listing 1.

```
If the response is authrequired
then the next authority is Jackie-Brown.
```

**Listing 1**: Business rule in ACE

We use the work of Bahr [25] to automatically transform ACE to RuleML. ACE uses an inter-lingua called *Discourse Representation Structures* (DRS) [28] for interoperability with other standards (e.g. OWL-DL). DRS is a syntactical variant of first-order logic that eliminates ambiguities in natural language. The DRS representation of an ACE sentence is literally translated into RuleML. Listing 2 shows the equivalent transformation of the business rule in ACE to RuleML (via DRS).

```
<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 2:** RuleML fragment of Authority1 rule

For the time being, we created a prototype to transform RuleML to Jess using XSLT based on the work of Tabet [26]. The transformation is however specific to the Payment Problem Scenario. Basically, we parse relevant values (e.g. property, predicate, and object) of the `<Implies>` element as the *pattern* or *right-hand side* of the "`=>`" operator. Relevant elements of the `<Exists>` element are parsed and treated as the *action* or *the left-hand side* of the Jess rule. The XSLT transformer inserts other relevant Jess rule constructs such as `defrule`, `assert`, the "`=>`" operator, and the parentheses in their appropriate locations. The transformed RuleML to Jess rule is shown in Listing 3.

```
(defrule Authority1
(response authrequired)
 => (assert (authority Jackie Brown)))
```

**Listing 3:** Business rule in Jess

Next, we created a Web service to wrap the Jess rule in Java, gave it the operation name `getNextAuthority`, and deployed it along with the Jess rule engine in Apache Tomcat. This operation takes a response code of either `AUTHREQUIRED` or `DENIED` as input, and provides the first and last names of the appropriate Authority as output.

This service is modeled as the *Authority* service in Figure 4 which realizes the business rule modeled in ARMOR. At this point, the deployed Jess rule Web service must now be used as one of the services (along with those identified in Step 1) to compose the Mediator. To do this, this Web service will also have to be abstracted in ISDL and Java as was done with the existing services in Step 1.

**Step 5: Design of the Mediator PIM**. Finally, with all required services identified and made available, we are ready to design the behaviour and information models of the Mediator. The information model of the Mediator is constructed from the union of the information models of Blue and Moon. The construction of the behaviour model of the Mediator 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 Mediator provides one service that must match the service requested by Blue. The service provided by the Mediator can initially be defined as the 'complement' of the service requested by Blue. The complement of a service is obtained by changing each operation call into an operation execution, and vice versa, while keeping the same parameters. For example, `initiatePayment` of Blue is represented as an operation call (requested service) in Step 1. Taking the complement of this operation on the Mediator's side, the `initiatePayment` will be represented as an operation execution (provided service). This results into the skeleton of the Mediator.

The design of the Mediator behaviour can now be approached as the search for a composition of the requested services that conforms to the provided service. The structure of this composition is defined by the (causal) relations among the operations. Most of these relations can be found by matching the input that is required by each operation to the output that is produced by other operations. For example, a causal relationship can be established between Blue's `initiatePayment` and Moon's `getBankingData` because the latter requires a `requestId` which is provided by the former. Matching inputs and outputs is however insufficient to find all relations. Furthermore, specific processing logic may have to be designed manually.

Figure 6 shows the behaviour model of the Mediator PIM and the mapping functions between operations. We designed the `processPayment` to be called twice: first when the purchase order amount with less than `€2000`, and second when greater than `€2000`. Although modeled separately in ISDL as `processPayment1stCall` and `processPayment2ndCall`, the same `processPayment` operation provided by Blue's Financial Department Payment Service is called.

Looking at the case description, the authorize and `getNextAuthority` operations must be invoked iteratively until the response from Management Department Payment Service's is `ACCEPTED`, it is best to move these operation calls into a separate *behaviour type* so that they can be reused. This also implies that the original operation calls will now be represented as *delegated operation calls* in the Mediator (depicted as a grayed operation call).

The definition of the data transformations among operation parameters can be approached as a refinement of the relations among operations. These relations define for each operation which other operations it depends on, and therefore which output parameters can be referred to or used in the generation of its input parameters. The data transformations then define how the value of each input parameter is generated from the values of the output parameters. This involves the definition of translations

between the vocabularies used by Blue and Moon. However, these translations only need to address those parts of the vocabularies that are related via the relations defined in Step 2.

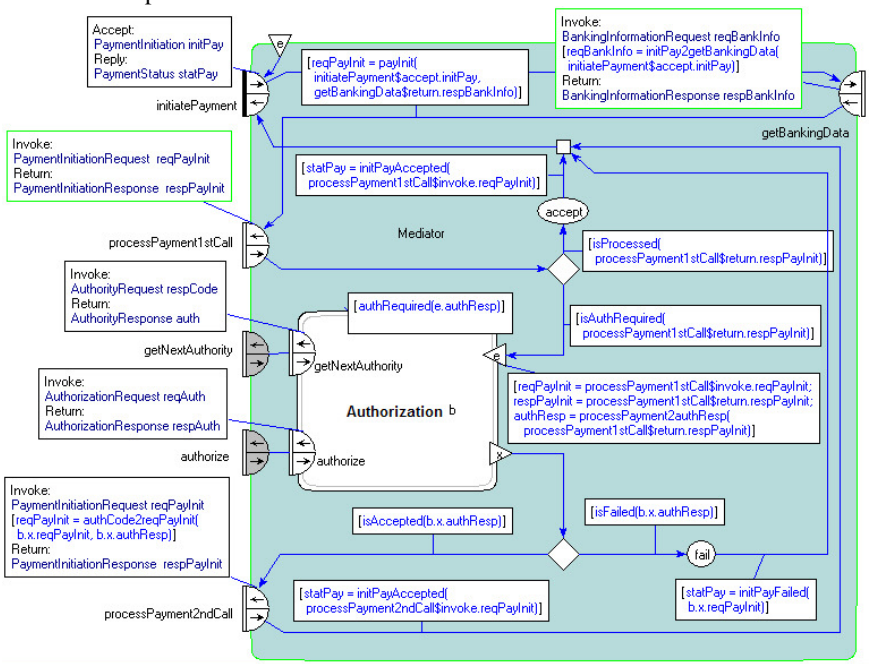


**Fig. 6.** Behaviour model of the Mediator PIM in ISDL

In relation to data transformation, we have developed an Eclipse plug-in editor called *Tizzle* for mapping information specification between Blue and Moon using our domain specific language. The tool provides syntax highlighting, expanding/collapsing of code fragments, and an important content-assist functionality which we believe can reduce the time-consuming problem of fixing syntactically incorrect mapping definitions. Figure 7 shows a screenshot of Tizzle.

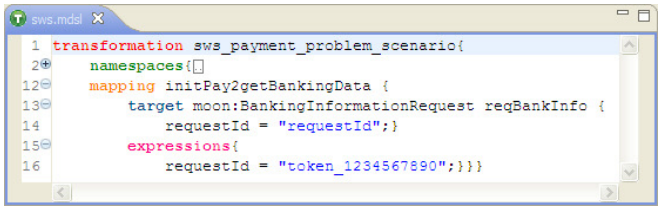


**Fig. 7.** Information mapping in Tizzle

**Step 6: Validation of the mediator PIM**. In this step, the design of the Mediator is validated by means of (i) assessment of the interoperability between the services of Blue, the Mediator and Moon, and (ii) simulation of the interacting behaviour of these

services. The interoperability assessment method has been presented in [11]. In short, the method checks whether each individual interaction can establish a result and whether the service composition as a whole can establish a result. The simulation of behaviours is supported by the *Sizzle* tool [15]. Simulation allows a designer to analyse the possible orderings of operations occurrences, as well as the information results that are established in these operations. In addition, the simulator enables us to perform real web service invocations and incorporate the results that are returned by web services during the simulation. This means that that the simulator provides, in principle, an implementation for the Mediator. However, this simulator does not support important properties of an execution environment, such as performance, monitoring, etc. Therefore, in the next step we transform the Mediator design to a BPEL process.

**Step 7: Derivation of the mediator PSM**. In this step, an implementation is derived for the Mediator design. For this purpose, a transformation has been developed that transforms an orchestration model from ISDL to a BPEL specification that can be executed on a standard BPEL engine. This transformation consists of two main tasks: (i) the recognition of common behaviour patterns, such as the workflow patterns, and their translation to a composition of one or more of the following basic patterns: *sequence, concurrence, selection* and *iteration*, and (ii) the realization of these basic patterns using the BPEL constructs `bpel:sequence`, `bpel:while`, `bpel:flow` and `bpel:if`, respectively. In addition, the Mediator model has to be annotated with information that is required as input to the transformation. This information concerns choices in the mapping of abstract ISDL behaviour constructs onto concrete BPEL constructs or extra design information that is needed at platform specific level. For more information on the transformation we refer to [13].

## 5 Conclusions and future work

This paper extends our previous service integration methodology and illustrating it using the Payment Problem Scenario of the SWS Challenge. Our objective is to integrate goal oriented requirements engineering in the design of service mediation solutions using model driven techniques. We argue that such a solution allows business domain experts to gain a better understanding of the requirements and how such requirements are realized by the architecture and related artifacts. We showed how goals, refined as business rules, can be represented at the CIM, PIM and PSM layers and be incorporated to constrain the design of the Mediator. The salient feature to this approach is that business rules are made explicit and manageable as they are separated from the business logic they constrain.

A significant limitation of this research is the transformation between the DRS representation of ACE in RuleML into Jess. As our solution is largely a prototype and specific only to the example scenario. Possible research work can be directed towards this end. The *SweetRules* project which in an open source platform for semantic web business rules transformations may provide a possible direction [30].

The goal modeling approach we proposed in Step 3 is still at its early stages and may require further refinement once ARMOR has been applied to more case studies both in the industry and the academe. The methodology provided by the KAOS goal modeling language may provide an initial starting point [19].

Our research has not explored the validation between the generated goal models at the CIM layer and their implementations in the PIM and PSM layers; that is, we still need formal ways to verify if whether or not the overall effect of the service provided by the Mediator, when executed, does indeed satisfy the overall achievement of the integration's hard goal. On the business rule side, the semantic equivalence between the different specifications of the rules at the different layers of the MDA stack also needs formal verification.

Finally, since business rules have now been treated as separate design and implementation artifacts, rule management can be challenging especially when the number of business rules grows. We shall study the possibility of managing rules in a *rule registry* which provides, among others, the ability to list all available rules, their relations, author, mutation, and location [29].

## References


1. Pokraev, S., Quartel, D.A.C., Steen, M.W.A., Wombacher, A., & Reichert, M. (2007). Business Level Service-Oriented Enterprise Application Integration. In Proceedings of the 3rd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2007),Funchal, Portugal, March 28-30, 2007, pp. 507-518. Berlin: Springer Verlag.
2. An, L., & Jeng, J-J. (2007). Business-Driven SOA Solution Development. In: IEEE International Conference on e-Business Engineering (ICEBE 2007), 439-444.
3. Anton, A.I. (1996). Goal-based requirements analysis, In: Second IEEE International Conference on Requirements Engineering ( ICRE `96), Colorado Springs, Colorado, 136-144.
4. Iacob M.-E., Rothengatter D., van Hillegersberg J. (2009). A Health-care Application of Goal-driven Software Design. Applied Medical Informatics, 24(1-2), 12-33.
5. Miller, J. & Mukerji, J. (Eds.). (2003). MDA Guide Version 1.0.1, Object Management Group doc.omg/2003-06-01, 12 June 2003, Retrieved June 12, 2009 from http://www.omg.org/docs/omg/03-06-01.pdf.
6. Business Rules Group (2000). 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.
7. Quartel, D.A.C. and Pokraev, S.V. and Mantovaneli Pessoa, R. and van Sinderen, M.J. (2008) Model-driven development of a mediation service. In: Twelfth International IEEE Enterprise Computing Conference, EDOC 2008, 15-19 Sep 2008, Munich, Germany. pp. 117-126. IEEE Computer Society Press. ISSN 1541-7719 ISBN 978-0-7695-3373-5
8. SWS Challenge Payment Problem Scenario. http://sws-challenge.org/wiki/index.php /Scenario:_Payment_Problem
9. Quartel, D. A. C., Steen, M. W. A., Pokraev, S. and van Sinderen, M. J. COSMO: a conceptual framework for service modelling and refinement. Information Systems Frontiers, 9 (2-3). pp. 225-244. ISSN 1387-3326, 2007
10. Ferreira Pires, L. Architecture Notes: a Framework for Distributed Systems Development. PhD thesis. CTIT Ph. D.-thesis series No. 94-01. ISBN 90-9007461-9, 1994
11. Pokraev, S.V. and Quartel, D.A.C. and Steen, M.W.A. and Reichert, M.U. (2006) Requirements and Method for Assessment of Service Interoperability. In: Proceedings of the Fourth International Conference on Service Oriented Computing

(ICSOC'06), 4-7 Dec 2006, Chicago. pp. 1-14. Lecture Notes in Computer Science 4294. Springer Verlag. ISSN 0302-9743 ISBN 978-3-540-68147-2
12. Quartel, D. A. C., Steen, M. W. A., Pokraev, S. and van Sinderen, M. J. COSMO: a conceptual framework for service modelling and refinement. Information Systems Frontiers, 9 (2-3). pp. 225-244. ISSN 1387-3326, 2007
13. Dirgahayu, T. and Quartel, D.A.C. and van Sinderen, M.J. (2007) Development of transformations from business process models to implementations by reuse. In: Proceedings of the 3rd International Workshop on Model-Driven Enterprise Information Systems, MDEIS 2007, 12 June 2007, Funchal, Portugal. pp. 41-50. INSTICC Press. ISBN 978-989-8111-00-5
14. Quartel, D. A. C. Simulation and execution of service models using ISDL. In: Proceedings of the 1st International Workshop on Architectures, Concepts and Technologies for Service-Oriented Computing, ACT4SOC 2007, July 22, 2007, Barcelona, Spain. INSTICC Press, pp. 19-27. ISBN 978-989-8111-08-1
15. ISDL. http://ctit.isdl.utwente.nl
16. JAX-WS and JAXB. http://java.sun.com/webservices/technologies/index.jsp
17. ElipseUML. http://www.eclipsedownload.com/
18. Mantovaneli Pessoa, R., van Sinderen, M.J., & Quartel, D.A.C. (2009). Towards Requirements Elicitation In Service-Oriented Business Networks Using Value And Goal Modelling. In: Proceedings of the 4th International Conference on Software and Data Technologies (ICSOFT 2009), 26-29 July 2009, Sofia, Bulgaria. 392-399. INSTICC. ISBN 978-989-674-009-2
19. Lamsweerde, A. van. (2009). Requirements Engineering – From System Goals to UML Models to Software Specification. John & Wiley and Sons, England.
20. Quartel, D.A.C., Engelsman, W., Jonkers, H., & van Sinderen, M. (2009). A Goal-Oriented Requirements Modelling Language for Enterprise Architecture. In: Proceedings of 13th IEEE International EDOC Conference (EDOC 2009), Auckland, New Zealand, 31 August - 4 September 2009. ISBN: 978-0-7695-3785-6
21. Lankhorst, M., et al. (2005). Enterprise Architecture at Work. Springer-Verlag Berlin Heidelberg. ISBN-10 3-540-24371-2.
22. Fuchs, N., Schwertel, U., & Schwitter, R. (1999). Attempto Controlled English (ACE) Language Manual Version 3.0. Institut für Informatik der Universität Zürich.
23. Rule Markup Initiative. http://ruleml.org/.
24. Friedman-Hill, E. (2003). Jess in Action: Rule-Based Systems in Java, Manning Publications Co.
25. Bahr, P. (2008). The ACE2RRML Web Service – A Web Service for Translating Controlled Natural Language into Reaction RuleML. Retrieved August 4, 2008 from http://www.pa-ba.info/?q=pub/ace2rrml.
26. Tabet, S. (2002). RuleML and Jess. Retrieved August 24, 2009 from http://home.comcast.net/~stabet/page3.html.
27. Attempto Parsing Engine Web Client. http://attempto.ifi.uzh.ch/ape/
28. Fuchs, N.E., Hofler, S., Kaljurand, K., Schneider, G., & Schwertel, U. (2005). Extended Discourse Representation Structures in Attempto Controlled English. Technical Report i-2005.08, Department of Informatics, University of Zurich, Zurich, Switzerland, 2005.
29. Morgan, T. (2002). Business Rules and Information Systems: Aligning IT with Business Goals. Addison Wesley.
30. SweetRules. http://sweetrules.semwebcentral.org/