Rule-Based Service Composition and Service-Oriented Business Rule Management

Hans Weigand, Willem-Jan van den Heuvel and Marcel Hiel

INFOLAB, Tilburg University, Warandelaan 2, Tilburg, The Netherlands
H.Weigand@uvt.nl, wjheuvel@uvt.nl, m.hiel@uvt.nl

Abstract. As business processes change constantly, there is a growing need for adaptive composite services. Unfortunately, existing service composition languages and techniques result in rather brittle and rigid processes, whose services live in a straitjacket. In this paper, we propose a rule-driven approach for service composition that is purely declarative, highly adaptive and integrated in a truly service-oriented approach to business rule management.

1. Introduction

Today's business environment dictates organizations to be agile so they are able to accommodate their business processes to rapidly changing market conditions, including updated or new legislations and regulations, swiftly changing consumer demands and novel technological innovations, e.g., new mobile platforms. Service Oriented Architecture captures an emerging paradigm that is quickly gaining broad industry acceptance, and enables the development of a new breed of (cross-) enterprise applications that are comprised of loosely coupled services, which holds the promise that these applications can be modified and/or extended on the fly..

One of the key impediments towards realizing this vision, unfortunately, is that currently services are predominantly composed using block-structured and graph-based languages, notably BPEL, resulting in static and brittle composite services, although some work has been done on trying to make them a bit more adaptable, e.g., [1]. Composite services that are developed in this way are liable of intermingling process logic with business rules, providing the perfect ingredients for unmanageable and rather repellent process/rule spaghetti. This has become even more problematic as companies have begun to apply languages such as BPEL for very agile, real-world applications, and have observed that rules are in fact much more dynamic than business processes. Consequently, updating these rules that are deeply buried in the scattered process definitions has quickly grown into a complex, labor-intensive and cumbersome task.

It has been suggested that business rules can be separated from the BPEL code in a kind of aspect-oriented flavor [2]. Although this alleviates the management problem to some extend, adaptations are still only possible as long as they concern the content of pre-identified business rules that fit into the fixed BPEL frame.

In this paper, we argue that business rules can be used in service composition without the need of such a BPEL frame, thus increasing the adaptability of the orchestration significantly. At the deployment level, we introduce a CA-rule engine that supports rule-based service composition. To keep the business rules manageable themselves, we describe a service-oriented approach.

This paper is organized as follows. In the following section we will introduce a realistic running example that motivates the rule-based approach. In section 3, we will elaborate on this approach and how it fits in a service-oriented architecture. In section 4, we introduce the FARAO approach towards service composition that is based on the use of a CA-rule engine, and analyze to what extent business rule compliance can be realized in this framework. The last section summarizes the main findings of our work, and sketches directions for future research.

2. Motivating Case Study

MultiTech (fictitious name) is a wholesaler SME that buys and sells mobile phones. Its primary business process revolves around (re-)selling mobile phones, which it acquires from various international vendors. In this fictional, yet realistic, case study we exemplify business services, rule services and the actors invoking them, concentrating on the purchase-and-payment cycle of MultiTech.

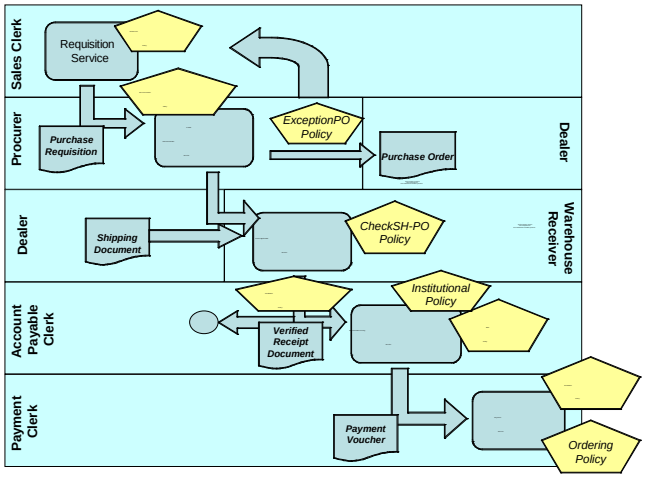

Fig. 1 Swimlane model of the Purchasing and Payment Processes

This cycle is organized as follows. The cycle starts with an authorized sales clerk requisitioning mobile phones. After his permission to requisite a particular product is ascertained, he may issue a purchase requisition to the inventory manager.

The inventory manager then sends the verified purchase requisition to the purchase agent, whose authorization to deal with this particular kind of order is then checked (`PermissionPolicy`). The purchase agent transforms the purchase requisition in a

purchase order, while ensuring that the used master data complies to the supplier's product code; in case of a problem, an exception is raised (`ExceptionPO-Policy`). Also, the internal stock level is checked; a policy describes the stock replenishment level. Thereafter, he sends the purchase order to the vendor, and issues two additional copies of the purchase order, one to the warehouse clerk and one to the payment clerk.

Service	Business policy	Definition
RequisitionService	PermissionPolicy	The sales clerk should be authorized to requisition particular (quantities of) products, e.g., Sales Clerk "Klaus" is allowed to requisition not more than 1000 mobile phones a time.
CreatePurchaseOrderService	PurchaseOrder-Policy	A sales order can only then be created if, and only if, the purchase requisition is complete, the order would cause stock dropping below the allowed replenishment level, and he is authorized.
CreatePurchaseOrderService	ExceptionPO-policy	A purchase order cannot be created if the ordered product does not exist cf. the master data; in this case, a new purchase requisition is required.
CheckAgainstPOService	CheckSHPO-Policy	An incoming shipment document must be checked against a purchase order document within 24 hours after receipt.
PurchaseProcessService	VerificationPolicy	If the verified document is not valid, then the ordering process is terminated.
PurchaseProcessService	InstitutionalPolicy	A purchase must be financially reported on a real-time basis (cf. institutional policies declared in Sarbanes-Oxley)
PurchaseProcessService	SODPolicy	The actor invoking this service should be another than the actor invoking the requisition process to obey segregation of duties.
PaymentService	EscalationPolicy	In case of dubious credits, the payment should be escalated to a human manager.
PaymentService	OrderingPolicy	A payment can only be invoked, after the ordered products have been received, and checked against the PO.

Table 1. MultiTech Business Policies

The vendor invoices MultiTech by sending an invoice along with its shipment to the warehouse clerk. After receipt of the goods and its accompanying shipment document, the warehouse clerk uses the purchase order and the receiving report to verify the correctness of the delivery. Then he sends the verified receiving report to the accounts payable clerk.

If the purchase is not valid, the process is terminated according to the `VerificationPolicy`, e.g., in case the budget has been exceeded. Also, the accountant is liable of reporting any payments directly to the government cf. Sarbanes-Oxley act, section 409 (`InstitutionalPolicy`). Also, to ensure segregation of duties, and circumvent potential fraud, the accountant cannot be the same person as the sales clerk (`SODPolicy`).

The accountant creates and sends a payment voucher to the payment manager, together with the verified receiving report, purchase order, and verified invoice; only after all this information is available the payment can be processed (`OrderPolicy`). Note that in case of excessively large purchase orders in a specific time period, the payment process is escalated to the management for further consideration (`EscalationPolicy`).

The example makes clear that business policies are first-class citizens in the modern enterprise and directly influence business services.

3. Business Rules and Service Composition

In this section, we first define and classify business rules and policies. In 3.2, we review previous work on rule-based service composition, and in 3.3 a new service-oriented approach to business rule management is described.

3.1 Business Rules and Policies

We follow the fundamental distinction between business rule and policy [3]. Policies arise from internal sources such as business needs, from corporate-level guidance, from external laws and regulations, and from ethical motivations. Based on the OMG Business Motivation Model (BMM) such policies "govern or guide an enterprise," specifying business design aspects that complement information and operation models [4].

Business policies are usually written in natural languages to cater for evaluation by domain experts, viz. business analysts. That evaluation assumes human interpretation, as the ambiguities of natural languages must be resolved and application of policies to specific business contexts generally requires analysis of impacts, consequences, and trade-offs. Thus, policies provide guidance but insufficient detail for implementation. Considerable research has been conducted into the conceptualization of business policies using languages such as ORM ([5], [6]), ILOG and OCL.

The application of policies in specific contexts leads to business rules, meaning highly structured, discrete, atomic statements "carefully expressed in terms of a vocabulary" [4] to enforce constraints (integrity rules), to deduce new information (derivation rules) or to trigger actions on satisfied conditions (reaction rules) [7]. If a business rule “defines and constrains some aspect of the business” [4], we can distinguish between norms or constraints (*constraining*) and definitions (*defining*). The former category can, without loss of generality, be expressed as prohibitions, indicated in deontic logic with the F modality, whereas the latter typically take the form of derivation rules.

Following [3], we posit that business rules are about business requirements, rather than about execution. They model “what” is required, rather than “how” it should be implemented. Hence we distinguish business rule languages from (executable) production rule languages such as ECA-Rules [8] and “IF…THEN” (CA)-rules [9]. SBVR is an OMG proposal for the representation of business rules in Structured English [10].

In order to operationalize constraints, often information has to be added. According to [11], a constraint (called norm frame in their terminology) should consist of 5 elements: a norm condition, a violation condition, a detection mechanism, a sanction and repairs. The *violation condition* is a formula denoting the state when the norm is violated. Although in simple cases, there is a 1-1 relationship between norm and violation condition – for example, if the norm is F(α) for an observable action α then the violation condition is DONE(α) – it is not possible to derive one from the other in all cases. For example, when a certain action is not defined in the operational context, or when the norms cannot be interpreted in isolation. The *detection mechanism* provides the procedure necessary to determine whether the violation holds at a certain moment. For example, the OBL(α BEFORE d), expressing that action α must be performed before deadline d, can be checked efficiently by a trigger that fires when the deadline d has been reached (based on a clock signal), and that checks DONE(α). Note that the detection mechanism here is more specific than the violation condition. The *sanction* is an action that is to be performed when a violation has been detected, whereas a *repair* is an action that tries to undo or compensate a violation. Following this approach, it is clear that the translation from norm to executable rule is not a simple transformation.

In SOA, a series of (partially overlapping and conflicting) specifications and standards have been proposed that can be used to render business policies and rules. WS-Policy entails a family of semantic-agnostic languages to express assertions about constraints and capabilities of service end-points. These constraints and capabilities can be either very generic, or domain-specific, e.g., defining security-, transaction or reliability policy constraints (cf., WS-Security, WS-Transactions and WS-Reliaiblity). KAoS [12], Cassandra [13] and Rei [14] denote executable policy specification languages from the semantic web community, which are based on RDF and OWL. RuleML [15] and the Semantic Web Rule Language [16] constitute two general-purpose executable rule languages.

3.2 Rule-driven Service Composition –state of the art

Service composition sits at the heart of the Service Oriented Architecture, allowing service requesters to assemble several services that meet their requirements, into composite services. Unfortunately, languages like BPEL, suffer from severe problems, especially with regard to their flexibility and adaptability. Instead, rules have been investigated as an alternative declarative approach, boasting the following key advantages:

- **Intuitive formal semantics**: Rule-based languages exploit a limited set of primitives with the formality of an underlying logical and/or mathematical framework, and the quality of being meaningful to the domain expert.
- **Direct support for business policies**: business rules *enact* business policies in that policies can be transformed into business rules in a straightforward and transparent manner. These business rules are externalized and managed separately from the processes in which they are applied
- **Flexibility**: rule-based compositions are believed to be more flexible than BPEL-like compositions, given their ability to pursue alternative execution paths

in case a particular execution path fails, without having to redefine the composite service and redeploy it on a service engine.
- **Adaptability**: given the declarative nature of rule-enabled service compositions, they can be modified and/or extended to accommodate context-specific situations, e.g., in terms of external services or the deployment platform.
- **Reusability**: Since rules isolated from the business process context, they can be more easily reused in other service application contexts.

Recently, considerable efforts have been invested in rule-engines to support service compositions. In particular, we herein wish to mention the following key contributions. Firstly, in [7], a service-oriented rule engine was introduced that allows enterprises to access business rules by invoking distributed service-enabled ruleML engines that sit at the service supplier's service end-point. [17] introduces an alternative service execution environment in which rules can be defined, and subsequently injected in WSDL specifications, after which they can be deployed on a service executor. [2] introduces AO4BPEL, an aspect-oriented extension to BPEL that is able to weave business rules into BPEL frames. Alternatively, in [18] an approach is suggested to incorporate business rules in BPEL specifications, while enforcing them in rule engines that work in concert with BPEL engines, and coordinate themselves through an ESB. This approach basically works as follows; an interceptor is used to catch incoming and outgoing BPEL service invocations (activities), after which a business rule broker service is initiated, through which applicable business rules can be accessed. Depending on the interceptor mode (before/after), the BPEL activity is either fired or the control flow is continued.

In addition to research prototypes that were developed for the purpose of validation, several service-oriented rule engines are nowadays available, viz. the Oracle Fusion Middleware Rule Engine. This rule engine allows specifying business rules as ILOG facts that can be inserted into BPEL specifications. This is achieved by allowing users to map BPEL variables to facts in a rule repository.

3.3 A service-oriented approach to business rule management

Business rules are an example of crosscutting concerns, especially those encountered in composite services with a coordinating function, and run the risk of getting scattered over the system. In a service-oriented approach it is possible to encapsulate a certain business policy into a service. The advantage is that this service can be called from anywhere, and rule redundancy can be avoided (cf. [19]). However, given the presumed autonomy of services in SOA, it is not immediately clear how compliance to such rule services is ensured. This situation is similar to the situation in Multi-Agent Systems (MAS), where autonomy is a fundamental property of agents as well. In MAS, the problem is addressed by an institutional approach. As one of the earliest papers on this topic, [20] described a market place architecture for agents that draws on exception handling third parties that act like "institutions" as we know them from human societies (e.g. notary, registry). To realize such an institutional approach, [20] suggests three concepts. First, each agent in the system is assigned a "sentinel" that mediates the interactions of the agent with other agents. These sentinels monitor message traffic, detect violations to commitments, and apply resolution handlers. The

sentinel incorporates domain-independent exception handling expertise. Secondly, the system includes institutional or ancillary services such as a reputation service that can be called upon by the sentinels. Thirdly, agents cannot just enter the system; the only way to join is to register at the Registrar service, who only allows entrance after having assigned the agent a sentinel. Is it possible to use this solution approach in SOA?

As we just noted, the second element of the solution can be easily applied in SOA. We can introduce institutional services as services, and as far as they represent not only *mechanisms* (which is the focus of [20]) but also *policies*, it is possible to implement them using a rule-based approach. The first element requires more attention, as sentinels are clearly not part of SOA. However, there are recent developments within SOA that provide each service with a *service manager* (e.g. [21], [22]). This service manager can be realized as a service and has the possibility to adapt the service via a management interface (MOWS). In ASOA [22], the service manager follows a monitor-plan-act cycle as envisioned in autonomic computing, which is close to the specified behavior of the sentinel.

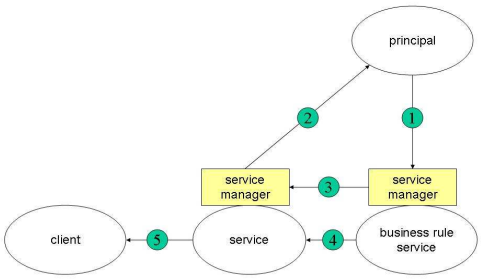

Fig. 2. Service-oriented business rule management, an institutional approach

In order to apply an institutional solution approach and to solve the compliance problem, we make the fundamental assumption that services have a dual orientation. One orientation is the client-orientation that lies at the heart of SOA. However, implicitly or explicitly, there is also another orientation towards someone who wants this service to be delivered. This party can be called the *principal*, and the relationship is one of *delegation*. Delegation means that a party wants to achieve something – typically providing a service to some customer – but rather than doing it himself, he asks another party ("agent") to do it on his behalf. Conceptually, the relationship can be characterized as a service offered by the "agent" to the principal. The delegation provides us with a mechanism to introduce services. A service X is introduced by a service provider – which we identify with a service manager – by replying to a request from the principal to deliver service X.

Now it becomes clear how a service can be bound to a business policy. When the principal requests the service manager to deliver service X, the request contains a reference to all the policies that should be respected as well. By adopting the request, the service manager commits himself to respect these policies. Within these policies,

a distinction can be made between generic rules, such as for detecting norm violations and reporting, and context-specific rules; the latter can be offered as separate services, and a generic rule only says that the service manager should call these services for this or that occasion. Fig. 2 describes the process of service introduction. A certain business rule service is assumed to exist representing some policy, for example, the PermissionPolicy of MultiTech. The principal uses this service (1). The service manager of the "agent" provides a service delivery service to the principal (2) upon his request. In performing this service, it uses and invokes the business rule service (3). Typically, in the case of a composite service, this implies that the service execution itself involves the business rule service (4), by orchestration. This all being in place, a client can call the service (5).

This scenario offers a solution to the compliance problem, but it does not assume a central Registrar authority. Each principal can impose its own policies. However, what the principal imposes is not an autonomous decision, as it depends on the policies imposed on him by powers above him.

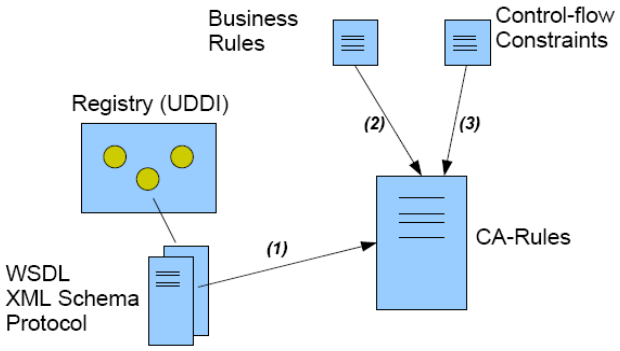

Fig. 3 The FARAO approach towards service composition

4. Framework for Designing, Reusing and Evolving Business Rules in Service Compositions

FARAO stands for a FrAmewoRk for Adaptive Orchestration. The ultimate goal of FARAO is to support the development of adaptable service orchestrations and to prepare for adaptivity by providing a manageability interface to a service manager, such as described in xSOC [23] and ASOA (Adaptive Service Oriented Architecture) [22]. Fig. 3 conceptualizes the relations between the ingredients of our service orchestration. Given a set of services to be orchestrated, the designer starts with retrieving the interface and data descriptions, typically from the registry. From these descriptions, Condition-Action (CA) rules are derived that manage the data flow. We have chosen for CA-rules rather than ECA-rules as the latter introduce more dependencies between the rules. In step (2), these rules are extended with business rules that typically steer the decisions in the orchestration. In step (3), the designer has the opportunity to add additional control-flow constraints, if required. In ASOA, all

three steps will be delegated to the service manager who executes them autonomously or semi-autonomously.

4.1 Data dependencies

The FARAO lifecycle model starts with a data-driven approach where the process structure is derived essentially from the data dependencies between the services involved in the orchestration. For example, if an orchestration involves both an Inventory service that returns, among others, the actual price of the product and a message to the customer with a quote, there is a data dependency between the two services that (implicitly) enforce that the former precedes the latter. If there is no data dependency between two services, there is no need to schedule one before the other, and by refraining from an arbitrary ordering we increase flexibility.

As hinted at in the above, we generate a CA-rule for each message that the orchestrator has to send. From the WSDL of the service in question, we derive the structure of the document it expects. Range restrictions on the message elements are copied into the conditions of the CA-rule. If there is not a range restriction, a NOT NULL condition is generated. In the action part, we put a *send* action that takes the service and its input document as parameters.

Rules refer to data items. In order to increase adaptability, we require that the orchestration is based on a shared ontology. WSDL-S [24] provides a mechanism to add semantics to web services. This allows, among others, that the message elements of the service are mapped to a given ontology. By requiring the WSDL descriptions of the services to be semantically annotated, we can let the rules refer to data items in terms of the shared ontology. In this way, changes in the service interface do not influence directly the orchestration, as long as the services adhere to the shared ontology.

4.2 Inference rules

The CA rules generated from the data dependencies provide an executable orchestration, but it only works well to the extent that the data items in the documents are seamlessly integrated. This is not always the case: sometimes an inference step is needed. For example, if one data item is "credit rating" and another is "creditworthy", then we need a rule to correlate the two that essentially prescribes when a person is creditworthy (for example, if credit rating > 10).

The general format of these inference rules is:

IF <condition> THEN $a_1 = v_1$ .. $a_j = v_j$

Technically, these inference rules are not CA-rules. We coerce them into CA rules by giving the consequent part an assignment interpretation: if the conditions are satisfied, then assign values v_1 .. v_j to the variables a_1 .. a_j

Example: After the accounts payable clerk has got the information from the CheckAgainstPO service it must decide whether or not to further process the order. The business rules for this decision can be formulated as follows:

```
Rule 1: IF verified-shipping-doc != "ok" THEN shippingstatus = reject
Rule 2: IF verified-shipping-doc == "ok" THEN shippingstatus = accept
```

These rules are to be used in combination with the rule (for the action) that processes the verified shipping document. This rule contains the condition that `shippingstatus = accept`. The rules 1 and 2 can be fed directly into the CA-engine, but they may also be part of a business rule service included in the orchestration. In the latter case, they are much easier to maintain of course. In a real-life implementation, a combination of the two approaches can consist in a caching solution, where the rules from the business rule service are moved to the CA-engine of the service temporarily. This approach saves on the communication overhead attached to service invocations. The cache has to be refreshed when the business rules are modified at the source.

4.3 Control flow constraints

The most prominent control flow constraint is the precedence constraint, where a certain service can only be executed after some other service has happened or some state has been reached. In Linear Temporal Logic, such a precedence constraint is usually described as: ¬β `UNTIL` α, where α and β are arbitrary propositions. In the case of orchestration, we restrict ourselves to constraints in which β is a service call. Then the meaning of the constraint is that this service cannot be called as long as α is not true.

For example, ¬`send(PaymentVoucher) UNTIL (PurchaseProcessing = "ok")` which enforces payment voucher is not issued to the service `PaymentService` before the `PurchaseProcessing` service has been concluded (MultiTech `OrderingPolicy`). A fundamental restriction of SOA is that services are autonomous, so the orchestrator cannot verify himself whether a certain service is finished; he is dependent on return messages of that service. If no return message is returned, there is no way to enforce precedence. Hence we restrict the α part of the precedence constraint to propositions on data (and not on the completion of some service as such). Within the present context, the β part is restricted to the event of sending a document.

In FARAO step (3), the control flow constraints are inserted into the CA-rules. In step (1), a CA-rule has been generated for each outgoing document. Let this rule be of the form "`IF D THEN send(M)`", and let a control flow constraint be ¬`send(N) UNTIL C`". If M=N, then we derive the rule "`IF D AND C THEN send(M)`".

Atomic prohibitions such as the `PermissionPolicy` in MultiTech can be injected into the CA-rule condition in a similar way (not worked out for lack of space).

4.4 Business Rules Implementation in FARAO

As far as business rules are concerned, we made a distinction between definitions and constraints. In the above, we have indicated how definitions can be incorporated in (the CA-engine of) FARAO as inference rules. Precedence constraints can be injected into the CA-rules. However, the interpretation of a norm frame requires more than precedence constraints. Not all norms are enforceable. In that case, the norm frame includes detection and remedy parts, among others. In FARAO, these can be directly implemented as CA-rules, although preferably, the service orchestration itself only

contains detection rules and the remedy is left to the service manager or an institutional service.

A type of rules not mentioned so far are permissions. If we follow the "everything is permitted unless forbidden" regime, permissions are not strictly needed. However, often permissions function as "second-order" constraints, determining which prohibitions can be added and which can not. In other words, they prohibit certain prohibitions, in which case they can be treated as constraints.

5. Conclusions

At present, organizations typically rely on block-structure, light-workflow specifications such as BPEL, to realize their business processes as composite Web-services. Unfortunately however, this style of composition assumes that at run-time, a detailed and complete process layout is "carved in stone", making its adaptation cumbersome, complex and time-consuming, requiring re-compilation of the process engine, and causing disruptions in, potentially mission-critical, business processes.

In this paper, a declarative and rule-driven framework to dynamic service composition, labeled "FARAO", is introduced, while its ramifications are further explored and illustrated with a realistic case study. The "heart-and-soul" of FARAO constitutes business rules that prescribe the way in which services can actually be aggregated dynamically into processes. The business rules are fed into the engine in a service-oriented way, that is, by a principal requesting a service delivery in accordance with given policies and by the service manager accepting this request. The business rules are maintained and updated outside the operational services. Given the platform independence offered by SOC, this can be anywhere inside or outside the company.

Our current research efforts concentrate on the implementation of the FARAO framework to experiment with rule-based service composition. A topic for future research is the mapping of our business rule representation to standard business rule languages, and to define a transformation from this language to the operational FARAO environment using a model-driven engineering approach.

References

[1] Reichert, M., Rinderle, S.: "On design principles for realizing adaptive service flows with bpel". In: Weske, M., Nttgens, M., (eds), EMISA. Volume 95 of LNI., GI, pp.133–146, 2007.
[2] Charfi, A. and Mezini, M., "AO4BPEL: An Aspect-oriented Extension to BPEL", World Wide Web, V.10, nr. 3, pp.: 309-344, 2007.
[3] N. Nayak et al., "Core business architecture for a service-oriented enterprise", IBM Systems Journal, Vol 46, No. 4, pp. 723-742, 2007
[4] The Business Motivation Model, Business Rules Group and the Object Management Group (OMG), http://www.businessrulesgroup.org/bmm.shtml
[5] Hargreaves, A. "Expressing Business Rules with Object Role Modeling", Proceedings of the 17th NACCQ, 2004.
[6] Halpin, T. "Business Rules and Object Role Modeling", Database Programming and Design, Oct 1996.

[7] Nagl, C., Rosenberg, F., Dustdar, S., ViDRE - A Distributed Service-Oriented Business Rule Engine based on RuleML. In: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06), 16-20. October 2006, Hong Kong, China.
[8] Chen, L et al., "ECA Rule-based workflow modeling and implementation for service composition", IEEE Transactions on Information & Systems, Vol. 89, Nr. 2, pp. 624-630, Feb. 2006.
[9] Geminiuc, K. "A Services-Oriented Approach to Business Rules Development", In: SOA Best Practices: SOA Cookbook, Oracle, 2007. Available at: http://www.oracle.com/technology/pub/articles/bpel_cookbook/geminiuc.html .
[10] Semantics of Business Vocabulary and Business Rules (SBVR), The Object Management Group, Sept 2006, http://www.omg.org/cgi-bin/apps/doc?dtc/06-08-05.pdf
[11] Vasquez-Salceda, J., H. Aldewereld, F. Dignum, Implementing norms in multiagent systems. In: G. Lindemann, J. Denzinger, I. Timm, R. Unland (eds), Multi-Agent System Technologies. LNAI 3187, Springer Verlag, pp. 313-327, 2004.
[12] Bradshaw, J.M., S. Dutfield, B. Carpenter, R. Jeffers, and T. Robinson. "KAoS: A Generic Agent Architecture for Aerospace Applications", in Proc. of the CIKM'95 Intelligent Information Agents Workshop. Baltimore, MD, 1995.
[13] Becker, M.Y.; Sewell, P Cassandra: distributed access control policies with tunable expressiveness**.** Proc. Fifth IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY 2004), pp. 159-168, 2004.
[14] Kagal, L., Finin, T., Joshi, A., 'A Policy Language for a Pervasive Computing Environment', Proceedings of IEEE 4th Int. Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy, 2003.
[15] Schroeder, M. and G. Wagner (Eds.): Proc. of the Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web., Italy, June 2002. CEUR-WS Publication Vol-60
[16] Horrock, I. et al., "SWRL: Semantic Web Rule Language", http://www.daml.org/rules/proposal/ , Dec. 2004.
[17] Kamada, A. and M. Mendes, "Business Rules in a Service Development and Execution Environment", Proc. of the Int. Symposium on Communications and Information Technologies, pp. 1366-1371, IEEE, 2007.
[18] Rosenberg, F. and S. Dustdar, "Business rules integration in BPEL: A service-oriented approach", Proceedings of the 7th International Conference on E-Commerce Technology, IEEE, 2005.
[19] Rosenberg, F. and S. Dustdar, "Towards a distributed service-oriented business rule system", Proceedings of the Third European Conference on Web Services (ECOWS'05), IEEE, 2005.
[20] Dellarocas, C., Klein, M., and Rodriguez-Aguilar, J. A. An exception-handling architecture for open electronic marketplaces of contract net software agents. In Proc. of the 2nd ACM Conf. on Electronic Commerce, EC '00, pp.225-232. ACM, 2000.
[21] Papazoglou, M.P. and W.J. van den Heuvel, "Service oriented architectures: approaches, technologies and research issues", VLDB Journal, Vol.16(3):389-415, 2007
[22] Hiel, M., H. Weigand and W.J. van den Heuvel, "An Adaptive Service-Oriented Architecture", In: Mertens, K, R. Ruggaber, K. Popplewell, X. Xu (eds), Enterprise Interoperability III, pp.197-208, Springer, 2008.
[23] Papazoglou, M.P., Extending the Service-Oriented Architecture. In: *Business Integration Journal*, February 2005, pp. 18-21.
[24] W3C, "Web Services Semantics", Version 1, W3C member submission, 2005.