Ontology-based Patterns for the Integration of Business Processes and Enterprise Application Architectures[1]

Veronica Gacitua-Decar and Claus Pahl

School of Computing, Dublin City University, Glasnevin, Dublin 9, Ireland.
vgacitua@computing.dcu.ie, cpahl@computing.dcu.ie

Abstract: Increasingly, enterprises are using Service-Oriented Architecture (SOA) as an approach to Enterprise Application Integration (EAI). SOA has the potential to bridge the gap between business and technology and to improve the reuse of existing applications and the interoperability with new ones. In addition to service architecture descriptions, architecture abstractions like patterns and styles capture design knowledge and allow the reuse of successfully applied designs, thus improving the quality of software. Knowledge gained from integration projects can be captured to build a repository of semantically enriched, experience-based solutions. Business patterns identify the interaction and structure between users, business processes, and data. Specific integration and composition patterns at a more technical level address enterprise application integration and capture reliable architecture solutions. We use an ontology-based approach to capture architecture and process patterns. Ontology techniques for pattern definition, extension and composition are developed and their applicability in business process-driven application integration is demonstrated.

Keywords: Software Architecture, Software Development, Data Architecture, Ontology, Software Quality.

1 Introduction

Software applications are built or acquired to provide specialised functionality required to support business processes. If new activities and applications are created and integrated into existing business processes and infrastructures, new architecture and information requirements need to be satisfied. Enterprise Application Integration (EAI) aims to link separate applications into an integrated system driven by business models and the goals they implement.

Business process management (BPM) aims to improve productivity, product quality, and operations of an enterprise. BPM encompasses methods, techniques, and tools to support the analysis, design, implementation and governance of operational business processes. Processes models have a critical role in the redesign of business processes.

However, business analysts and software developers often face difficulties managing challenges such as discovering, modelling, and understanding business processes in the context of their implementation through software applications.

Increasingly, enterprises are using Service-Oriented Architecture (SOA) as an approach to EAI. SOA has the potential to bridge the gap between business and technology and to improve the reuse of existing applications and the interoperability with new ones. Software services are the building blocks of SOA. They can be composed to provide more complex functionality and to automate business processes. However, if applications are created without a structured architectural design, integrating these into a coherent architecture closely aligned with the business processes becomes a significant challenge. Business processes do not map one-to-one to service architecture processes. This gap has turned out to be difficult to approach systematically and to automate.

Abstraction and knowledge representation are principles that can address these challenges. Architecture abstractions like patterns and styles capture design knowledge and allow the reuse of successfully applied designs, thus improving the quality of software. Abstraction is a central driver in software engineering approaches; at the business level the reuse of successfully business designs is equally important. The development of integrated enterprise-wide application architectures is a continuous process. To improve the process and overall quality, the experience of analysts, architects and developers should be captured and reused. Knowledge gained from integration projects should be captured to build a repository of semantically enriched, experience-based pattern solutions.

Reusing proven solutions in a semantically enriched form reduces costs and development time and ensures coherently integrated and architected application systems aligned with business processes. Using patterns enables architects to implement successful application integration solutions.

- Business patterns identify the interaction and structure between users, business processes, and data.
- Architectural patterns at a technical level address enterprise application integration and capture reliable architecture solutions.

A framework is required that can capture architectures and patterns as models and that can integrate representations from the different perspectives of business and software technology. We use an ontology-based approach to capture process and architectural patterns. A set of ontology-based pattern languages and techniques is developed and its applicability in business process-driven enterprise application integration demonstrated. A number of benefits of an ontology-based solution can be identified:

- enhanced extensibility through a taxonomical framework,
- alignment with ontology-based domain models and integration frameworks,
- use of ontologies as a semantical modelling notation,
- exploitation of the logical aspect of ontologies to infer additional knowledge from facts asserted in ontologies,
- use of ontologies as a formal reasoning framework to support architecture activities such as discovery and matching.

We outline the background in terms of SOA and ontology approaches in Section 2. Our proposed architecture framework is presented in Section 3, including a scenario that serves as an application context in which our solution is demonstrated. In Section 4, we

introduce our ontology-based service modelling notation and look at pattern modelling and matching. Section 5 investigates transformations. Finally, we discuss our results and future trends, before ending with some conclusions.

2 Background

SOA is the context of this work. In this section, we provide some background on SOA frameworks and how ontology technology has been utilised to support SOA, in particular in the context of enterprise integration.

2.1 SOA and Ontology Support for SOA Frameworks

SOA frameworks provide methodological guidelines, modelling support, and tools to develop service-oriented architectures. A service architecture design relates two domains that have their own modelling representation and dynamics regarding changes, i.e. the business domain and the software application domain. A SOA framework must provide an integrated modelling solution for both domains.

Successful, proven designs documented in the form of software design patterns have been widely used by software architects to develop software systems with improved quality (Gamma, Helm, Johnson, and Vlissides, 1993; Bass, Clements and Kazman, 2003). Similarly, business patterns provide proven designs for process models and informational models that help business analysts in creating business models (Fettke and Loos, 2006). An architectural framework must provide support to work with pattern descriptions and allow the incorporation of patterns into designs.

Ontologies are means of knowledge representation, defining so-called shared conceptualisations. Ontologies are frameworks for terminological definitions that can be used to organise concepts in a domain. Simple examples of ontologies are taxonomies, i.e. classification schemes used for example to classify animals or plants into hierarchies. Combined with a terminological logic such as description logic (Baader, McGuiness, Nardi, and Schneider, 2003), we obtain a framework for specification, classification, and reasoning in an application domain. The Semantic Web is an initiative for the Web that builds up on ontology technology (Daconta, Obrst and Smith, 2003). XML is the syntactical format. RDF – the Resource Description Framework – is a triple-based formalism (subject, property, object) to describe entities. OWL – the Web Ontology Language – provides additional logic-based reasoning based on top of RDF.

A specific application of ontologies in SOA are service ontologies – which have demonstrated the benefits of ontologies for software composition (Payne. and Lassila, 2004). WSMO (Lara, Stollberg, Polleres, Feier, Bussler, and Fensel, 2005), OWL-S (OWL-S Coalition, 2002) and the Semantic Web Services Framework SWSF (Semantic Web Services Language (SWSL) Committee, 2006) are the predominant service ontologies. Service ontologies are ontologies to describe Web services, aiming to support their semantics-based discovery in Web service registries. The Web Service Process Ontology WSPO (Pahl, 2007) is also a service ontology, but its focus is description and reasoning about service composition and service processes. The FLOWS ontology in SWSF comprises, like WSPO, process modelling.

Our work can be seen in the context of model-driven development (MDD). We can link our integration framework to two of the MDA model layers proposed by the OMG – computation-independent and platform-independent. The need for a specific MDD solution for the services context is a concern. The ubiquity of Web services and the existence of standardised and accepted platform and modelling technology justify this requirement. An OMG initiative to define and standardise an ontology metamodel (ODM) allows the integration of SOA-related ontologies with OMG standards (Object Management Group, 2003). ODM provides mappings to OWL-DL and also a UML2 profile for ontologies. We use ontologies instead of the Unified Modelling Language UML – the OMG suggestion – for modelling. ODM, however, is a standard addressing ontology description, but not reasoning. The reasoning component, which is important in our framework, would need to be addressed in addition to the standard.

2.2 Related Work

Several methodologies for service architecture design and development have been proposed in the past few years from industry and academia (Erl, 2004), (Papazoglou and van den Heuvel, 2006), (Erradi, Anand, and Kulkarni, 2006). Even though these methodologies provide useful guidelines, they do not propose techniques or tools to automate activities during the design process. They also require advances on modelling support for semantic and behavioural aspects of both business and software domains. In (Aalst, Beisiegel, Hee, Konig and Stahl, 2007), behaviour relies on the modelling richness of the process language used and the description of messaging protocols at component model level. Semantic aspects are not considered.

Architecture transformations are the main architectural concerns in the implementation of an architecture design. Traceability among elements of an architecture is an important property for architecture modification and evolution.

- Traceability between software components and relevant business elements of an enterprise has been exploited (Bernus, Nemes and Schmidt, 2003). In (Steen, Strating, Lankhorst, Doest and Iacob, 2005), the authors discuss the relevance and impact of service orientation to enterprise architectures. Traceability between services is used to align different views and to analyse the impact of changes in one view. Even though modelling support for traceability is provided, the architectural description of services in the different views is introduced at a high level.
- In (Dijkman and Dumas, 2004), a multi-view point approach for service-oriented design is introduced. The authors focus on interrelations of viewpoints at service and service composition level to allow consistency between different parties designing an inter-organisational service-based architecture. The views involve interface behaviour, provider behaviour, choreography, and orchestration. They formalise the views from a control flow perspective in terms of Petri nets. Traceability between views enables the static verification of the consistency of composite services.

Expert design knowledge expressed as patterns is often neglected in most architecture design frameworks. Analogously to patterns, styles are architectural abstractions that constrain the types and relations among elements of a design. In (Papazoglou and van den Heuvel, 2006), the advantages of using reference models to guide the definition of normalised business functions are discussed. In (Baresi, Heckel, Thöne and Varro, 2006), an architectural style-based approach to SOA modelling and design is presented. Service architectures models are refinements of business architectures. The refinement

of specific business scenarios into service architectures is based on the refinement of a generic business-level style into a service architecture style.

Some developments have started to exploit the connection between ontologies and layered architectural modelling. In (Djurić, 2004), an MDA-based ontology architecture is defined. This architecture includes aspects of an ontology metamodel and a UML profile for ontologies. A transformation of the UML ontology to OWL is implemented. The work by (Gašević, Devedžić, and Djurić, 2004; Djurić, 2004) and the OMG (Object Management Group, 2003a; Object Management Group, 2003b), however, needs to be carried further to address the ontology-based modelling and reasoning of service architectures. In (Pahl, 2008), we have presented work on semantic model-driven development of Web service architectures that integrates a service ontology framework into an MDA-framework in order to achieve higher degrees of automation. In (Zdun and Dustdar, 2007) and (Foster, Mukhija, Uchitel, and Rosenblum, 2008), MDD approaches to integrate process-driven SOA models are proposed. Beside meta-modelling support, Zdun and Dustdar use software design patterns for the integration of services and the business processes using them. To introduce patterns into the MDD approach, they propose pattern primitives as an intermediate abstraction to formally model informal design patterns. A pattern language for process-oriented integration of services is provided. However, the integration of business-level and service-based architectures needs to be further addressed in the context of Web-based ontology technology.

A natural application of ontologies in architecture design would focus on structural aspects of the architecture. In (Pahl, Giesecke and Hasselbring, 2007), an ontology-based approach for modelling architecture styles is presented. The authors introduce operators for style modification and combination between styles. Relations between quality requirements and style modelling are investigated. This work, however, is not adequately applicable to service process modelling and patterns as process-style abstractions. Grønmo et.al. (Grønmo, Jaeger, and Hoff, 2005) introduce – based on ideas from (Djurić, 2004) – an ontology-based service and service process modelling approach. Starting with a UML profile based on activity diagrams, services are modelled. These models are then translated into OWL-S.

3 An Architecture Framework for SOA-based EAI

A layered architecture for service-centric enterprise application integration is the backbone of our proposed framework. This framework shall now be introduced. We also introduce a case study that illustrates the main challenges of business process and architecture integration.

The problems of application integration in terms of modelling aspects, but also architecture and transformation issues need to be addressed:

- Modelling aspects. Modelling different views such as information hierarchy, behaviour, and semantics needs to be represented in an integrated framework.
- Architecture aspects. Architecture description and architecture abstractions in terms of patterns and styles are central. Reusable architecture design knowledge can be captured in the form of patterns.

- Transformation issues. Essentially, vertical transformations from business to software architectures are of relevance. Business aspects need to be mapped to services. A synchronisation between the respective model aspects is required.

3.1 Model Alignment and Traceability

Enterprise application integration aims to link separate applications into an integrated system driven by business models and the goals they implement. A central problem of applications integration is maintaining alignment between business and technical dimensions involved. Consistence between the dimensions can be improved through explicitly connecting the modelling elements of the EAI problem with elements of the architecture solution through traces. Change management and traceability are important aspects of IT system and business alignment. Two viewpoints need to be considered:

- Business View: Two main models describe the business dimension of the EAI problem: process models and domain models. While business process models capture the dynamics of the business, domain models capture structural relations between business concepts.
- Applications View: An application architecture represents a system composed of several software applications. Application architectures at enterprise scale normally grow in a decentralised way and involve different technologies, paradigms of development and modelling notations.

In order to provide traceability support to the views and elements involved in the integration problem, we develop an architectural approach that supports explicit traces between elements from different layers in a layered architecture. An explicit traceability model maintains the dependencies between elements of the integration problem and its service-centric solution. Traces provide flow dependency information among services that define the technical services composition. A pattern-driven transformation technique between the layers is the solution (Gacitua-Decar and Pahl, 2008b). Ontology support provides the necessary semantic integrity.

3.2 Layered Architecture

Our integration solution is structured based on a layered architecture called LABAS (Layered Architecture for Business, Applications and Services) (Gacitua-Decar and Pahl, 2008a). LABAS contains different layers capturing the business view, the applications architecture view and the view of the service-based integration solution. From a modelling point of view, an ontology-based meta-model will later define the common constructs among the different layers and provide the modelling support for the transformation from business to software levels. Fig. 1 shows the LABAS layers.

- *Business Modelling Layer*. The Business Modelling Layer (BML) is a container for the elements of the process model and the domain model. BML represents the business context of the integration problem. We use BPMN to model business processes and a basic ontology to represent domain information model entities.
- *Application Architecture Layer*. The Application Architecture Layer (AAL) is a container for the application components supporting the business processes in BML. AAL is organised as a process-wide applications architecture. Applications might be owned by different process roles in the BML. In order to describe an AAL model in architectural terms, we adopt the component and connector view (Garlan and Schmerl 2006), which describes a software system as a set of components. Each

component has a set of ports with interfaces, which enables the interaction with other components through connectors.

- *Business-Application Intermediate Layer*. The Business-Application Intermediate Layer (BAIL) provides a consolidated view of the integration problem. The aim behind jointly modelling the business and applications views is to derive an integration solution from BAIL models. The consolidated model integrates the models through a traceability model. The integrated models are the business process model, the domain model and the application architecture model. Traces between process steps and domain model elements, and between domain model elements and application components are the core elements that consolidate the business and application views of the integration problem.
- *Service Architecture Layer*. The Service Architecture Layer (SAL) provides a service-based integration view. It is a container for software services. These services, organised in a service architecture (Alonso, Casati, Kuno, and Machiraju, 2004) implement the integration solution. We use a process-centric component and connector view (Allen and Garlan, 1997) to represent AAL elements.

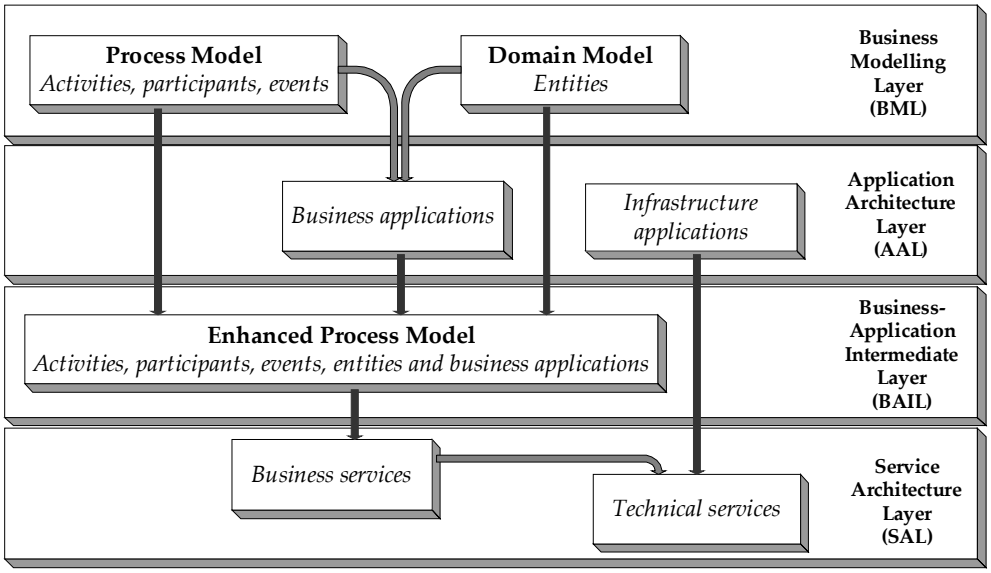

Fig. 1. Layered Integration Architecture Framework. (© 2008, Veronica Gacitua-Decar. Used with permission.)

We propose an incremental transformation from business models into service architecture descriptions to design the solution to the process and applications integration problem. The transformation is supported by architectural abstractions (patterns) and techniques allowing patterns manipulation. The pattern-based techniques provide support to business analysts and software architects to incrementally design the service architectures. The main supported activities encompass:

- *Business model augmentation* involves the addition of patterns into business models. Pattern instantiation techniques support this activity.
- *Business and technical service identification* involves the analysis of business models and their relation with the supporting applications in order to identify the services of the architecture solution – using pattern identification and matching.
- *Incremental business model to service architecture transformation* generates, step-by-step, an architecture solution to the business and applications integration problem.

This activity is supported by pattern-based transformation templates that represent predefined transformation steps.

- *Service architecture augmentation* incorporates SOA patterns to provide solutions to technical issues such as communication, security, and distribution of services.

3.3 Application Scenario

We illustrate the modelling, integration and transformation needs to solve a business process and enterprise application integration problem through an application scenario. It serves us as requirements elicitation tool for the ontological framework and acts as a framework to demonstrate where and how ontology-based techniques are beneficial.

The case study looks at the proposed framework within a scenario of integrated financial network services from an application perspective. Fig. 2 describes three actors and their processes involved in loan request and approval activities.

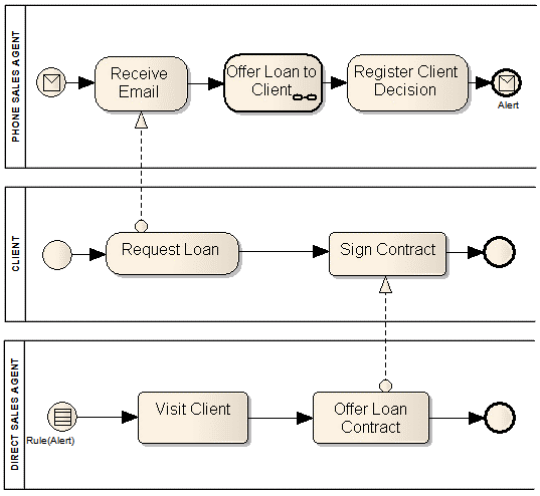

Fig. 2. Loan Management Process. (© 2008, Veronica Gacitua-Decar. Used with permission.)

The business scenario is a simplified process for loan management in a bank. The process starts when a client requests a loan by email. A phone bank's agent from a call centre calls the client to present an offer. The agent registers the relevant information regarding the client acceptance. If the client accepts the offer and the amount of money involved in the loan is sufficient to meet the sales goals of the bank, then a business alert is triggered. The alert triggers the visit of a direct bank's agent located near the client. This agent visits the client with the objective of signing a contract.

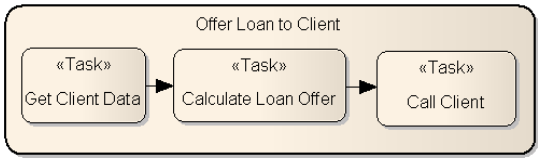

Fig. 3. Loan-to-Client Pattern. (© 2008, Veronica Gacitua-Decar. Used with permission.)

The business scenario describes a situation where individual processes from different parties have to be integrated – a central problem we aim to address. The consequence at technical level is an enterprise application integration problem. In order to demonstrate how to solve the business process and enterprise application integration problem, we detail the *offer loan to client* activity from the business process shown in Fig. 2. Then, we sketch the steps followed for the rest of the activities. Finally, we discuss how all the described steps accomplish the goal of solving the integration problem.

- The *offer loan to client* activity is performed by a bank agent, who calls clients to explain the conditions of a loan. The agent requires information of the client's funds, contact information and the available offer (loan). The amount of money and conditions of the loan depends on the goals that the bank agents have to meet.
- The *offer loan to client* activity consists of three more fine-grained process steps. The process steps, the domain model elements involved, and the applications supporting the activity are shown in Fig. 4. Note that the model from Fig. 4 is at the Business-Application Intermediate Layer (BAIL) and adds traces between elements from the Business Modelling (BML) and Applications Architecture Layer (AAL).

Loan-to-Client is a common pattern that is instantiated in the loan management process. We assume that the structure and process steps of the *offer loan to client* activity are described in a business pattern catalogue as the *Loan-to-Client* pattern (see Fig. 3). In the scenario here, matching of a business pattern is done straightforward by inspection of the process model and a business pattern catalogue.

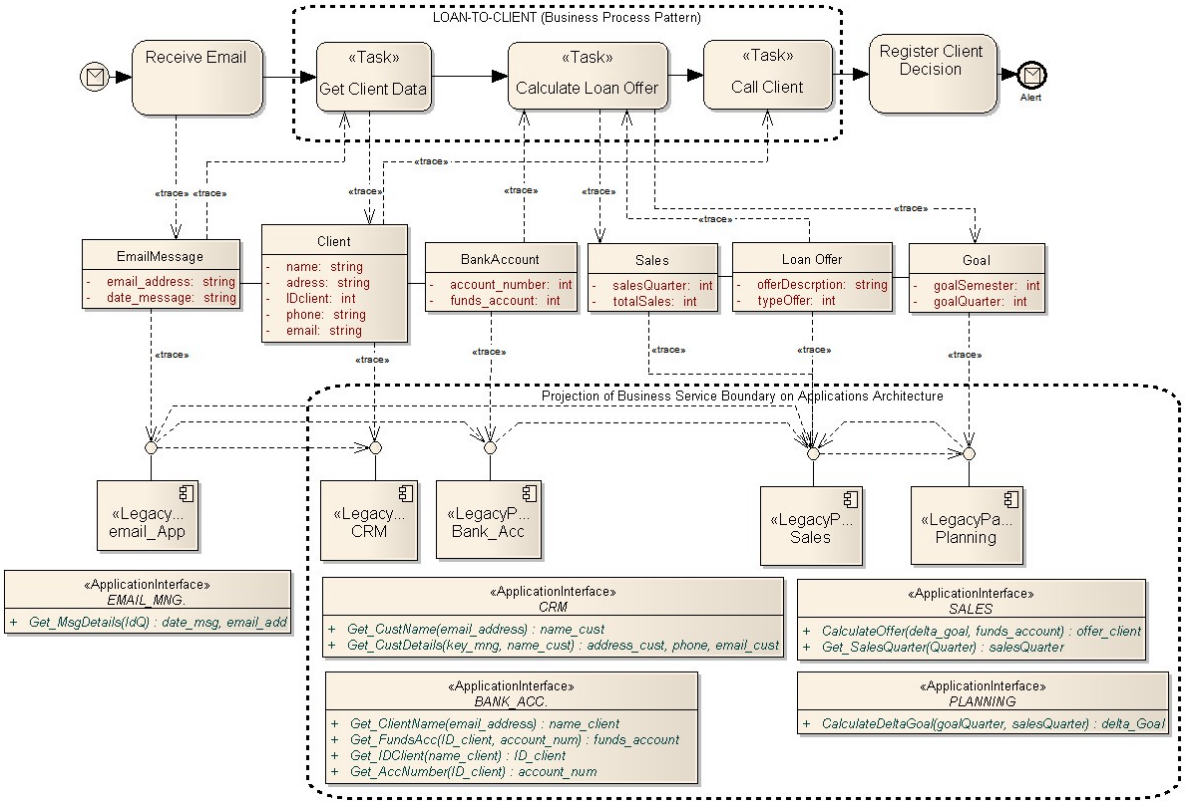

Fig. 4. Enhanced Business Process at Business-Applications Intermediate Layer BAIL. (© 2008, Veronica Gacitua-Decar. Used with permission.)

The central architectural problem of integration is addressed through the BAIL. As indicated in Fig. 1, it merges aspects from the BML and the AAL. On the technical level, the applications involved with the *offer loan to client* activity are a mail server application (EMAIL_MNG), a customer relation management application (CRM), an application managing the client's bank account (BANK_ACC), and two applications managing the information of sales and plans of the bank (SALES and PLANNING, respectively). The available operations from the applications that support to the process are shown in the application interfaces at the bottom of Fig. 4. The link between the AAL application interfaces (bottom), BML information elements (middle) and the BML process (top) is created through traces modelled by the software architect.

BAIL elements need to be transformed to SAL representations. The *Loan-to-Client* pattern has an associated transformation template, i.e. a collection of rules that guide the process transformation. The transformation template has a description of the business process pattern, the transformation and the associated service description. The business service description from the transformation template is pre-established, and the application-centric service description is generated for each specific application through a set of transformation rules. The following outlines the applied transformation rules:

- process patterns (as boundaries) to service ports: as default an implementation for the common pattern in the form of a service or service process is assumed – e.g. the abstract Loan-to-Client pattern is mapped to a respective technical service,
- applications to (legacy) systems: existing applications represented at the AAL layer are mapped to legacy application service providers (legacy participant) – e.g. BANK-ACC, CRM, SALES or PLANNING,
- business actors to business service providers (service participant): business actors provide business services that might be implemented through composition of services provided by legacy participants,
- process steps to service operations: operation-based mapping including in/output and pre/postconditions – e.g. the steps to get client data, calculate loan offer and call client, which are mapped to (legacy) component operations exposed as services.

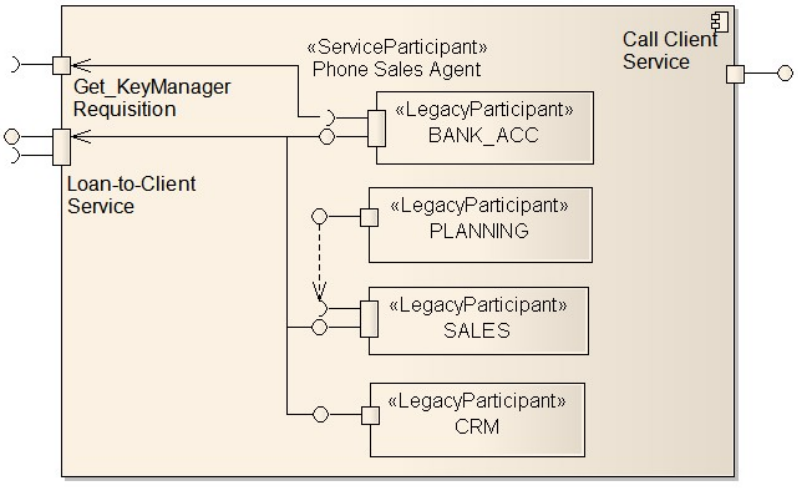

Fig. 5. SAL Applications architecture. (© 2008, Veronica Gacitua-Decar. Used with permission.)

The transformation from the BAIL layer (Fig. 4) to the SAL layer maps business processes to service architecture elements. Fig. 5 shows the application transformation

result. It integrates legacy components as services into the SAL architecture. The Loan-to-Client pattern is implemented as a separate service that encapsulates the corresponding workflow internally. Fig. 6 shows the process pattern and the associated composed service description. We have detailed one of the pattern activity steps to indicate that patterns are often used at a high level of abstraction. If needed, planning approaches can be used to determine the low-level flow (McIlraith and Son, 2002).

4 Patterns Modelling and Matching using a Service Process Ontology

We present our ontology-based architectural framework in this section. This framework supports, in addition to behavioural specification and modelling of processes through a process pattern ontology, also pattern matching and layered transformation through refinement techniques and meta-modelling and structural connectivity through an architectural pattern ontology – see Fig. 7.

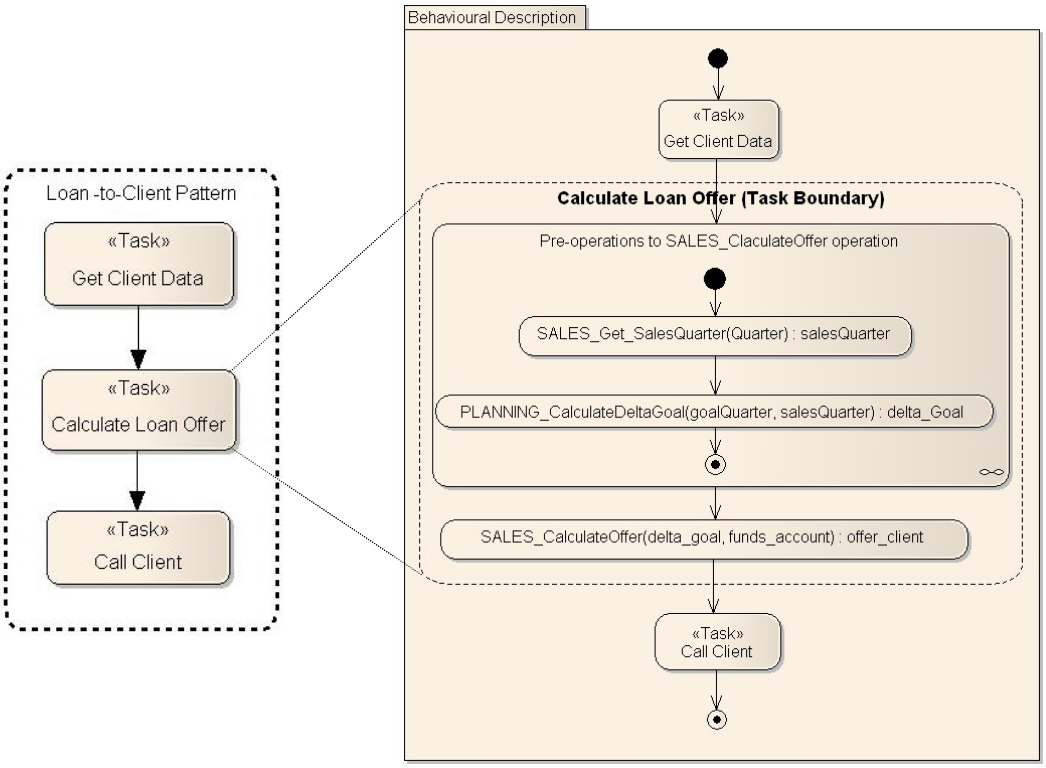

Fig. 6. SAL-level process for the Loan-to-Client Pattern. (© 2008, Veronica Gacitua-Decar. Used with permission.)

We distinguish two types of patterns, representing different architectural perspectives:

- Process patterns are workflow patterns, i.e. behaviour is the main focus. These are behaviour-oriented patterns that represent common process building blocks.
- Architectural patterns are structurally oriented patterns that focus on structural service connectivity. They define architectural styles and that have a direct link to quality attributes.

These patterns occur at different levels of abstraction (business process and services architectures), possibly based on different notations. An integrative mechanism with

rigorous formal semantics is provided through ontologies. For both, we define suitable ontology languages and techniques. We introduce the ontological tools based on the structure imposed by our architectural approach:

- behaviour through process ontologies in the context of description and modelling,
- refinement in the context of pattern matching,
- connectivity through architectural patterns in the context of transformations.

An ontology-based process-centric service architecture language is the central tool that supports modelling needs arising in our integration framework. The challenge of incorporating ontology technology to enrich architecture abstractions and architecture integration is to find a language that satisfies two criteria: firstly, to integrate business and technology-level descriptions and, secondly, to support pattern formulation, identification and matching for service processes. Ontology languages can provide the required language. Various models are used in the development process, such as a computation-independent business domain models that captures the characteristics of the application domain and a platform-independent, but technology-oriented architecture models that describe the service-based software system in abstract terms.

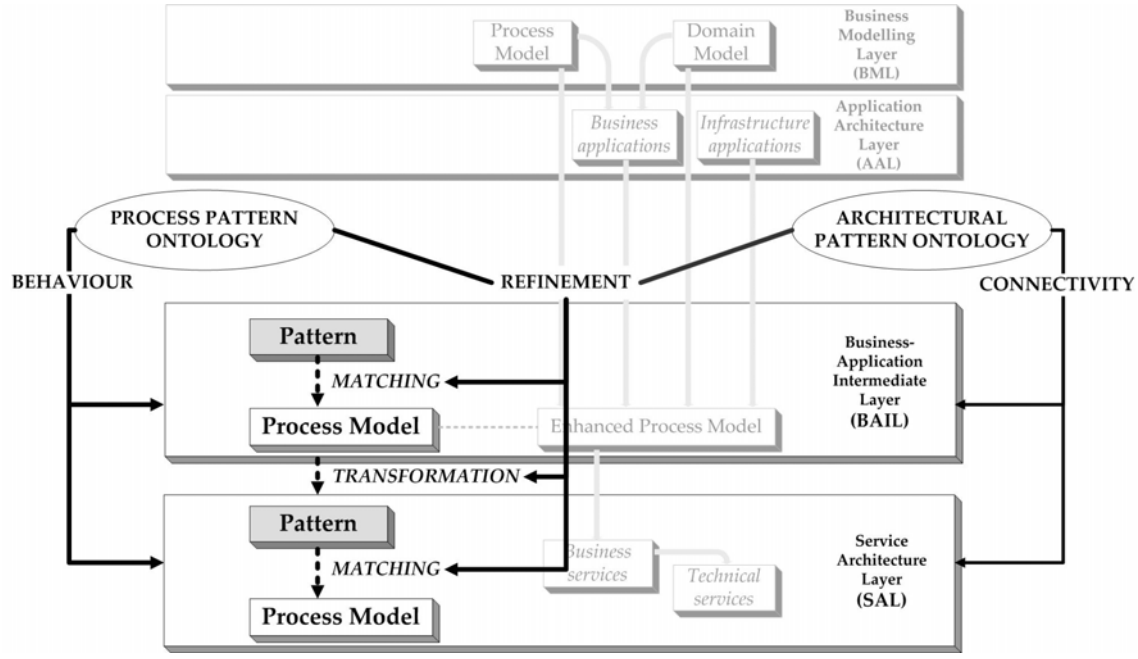

Fig. 7. Ontology-based Semantic Architecture Framework. (© 2008, Veronica Gacitua-Decar. Used with permission.)

4.1 A Service Process Ontology

The ontological modelling of services is our central concern. We need to look at how these models are used in the business process-driven software development. We can use Semantic Web-based ontologies to formalise and axiomatise business and service processes, i.e. to make statements about processes and to reason about them. Description logic, which is used to define OWL, is based on concept and role descriptions. Concepts represent classes of objects; roles represent relationships between concepts. Concept descriptions are based on logical combinators (negation, conjunction) and hybrid combinators (universal and existential quantification).

We first present a common ontology-based process modelling language, before explaining its application to BPMN, which is the main language for the BML and BAIL. Ontologies are a description of concepts and their relationships. Our solution is based on the service process ontology WSPO. WSPO is based on an extension of description logics. In particular, a rich role expression sublanguage is added to model service processes (Pahl, 2007). Service composition in Web- and other service-oriented environments is interaction. Services are considered as independent concurrent processes that can interact (communicate) with each other. Central in the composition are the interaction connectivity of services and the abstract effect of individual services.

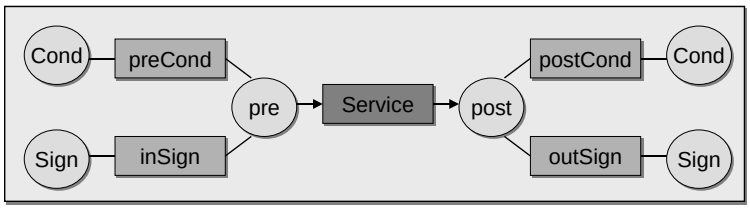

Fig. 8. A Service Process Ontology. (© 2008, Claus Pahl. Used with permission.)

An intuitive approach to represent process interaction behaviour in an ontological form is to consider services as central concepts. We, however, propose a approach that is particularly suitable for the abstract description of service *effects* and service process *connectivity* (Plasil and Visnovsky, 2002). Our objective is to represent services based on inherent notions of state and state transition. States of the systems are the central concepts; transitions (service) are represented as roles, see Fig. 8. Service executions lead from old (pre)states to new (post)states, i.e. the service is represented as a role or relationship in an ontological sense (a rectangle in the diagram). For instance, we could specify that a customer may check his/her account balance, or, that a transfer of money must result in a reduction of the source account balance. Usually, relationships in ontologies are used to express static properties, but they can also be seen as accessibility relationships between states of a system. We introduce a basic set of role expressions (composed ontological relationships) as service *connectors* for sequential composition *R;S*, iteration *!R*, choice *R+S*, and concurrency *R×S* into a basic ontology language to describe processes. These are common operators in process algebras. Using this language, we can express ordering constraints for services. For instance,

Login; !(BalanceEnq + Transfer)

is a role expression describing a connector process of an online banking user starting with a login, then repeatedly executing balance enquiry or money transfer.

The transitional *connector* roles are complemented by more descriptional *effect* roles. The effect is a contract-based specification of invariants, pre- and postconditions describing the obligations of users and providers. For instance, *preCond* associates a precondition to a prestate; *inSign* associates the type signatures of possible service parameters. Some properties, such as the service name, remain invariant. A logical *effect* specification focussing on safety is

positive(balance) -> Transfer . reduced(balance)

saying that if the account balance is positive, then money can be transferred, resulting in a reduced balance. Here, *Transfer* is the service; *positive(balance)* and *reduced(balance)* are pre- and postcondition, respectively. These conditions are concept expressions. *Transfer* causes a system to transfer from a prestate *pre* to a poststate *post*.

We use a connection between description logic and dynamic logic – a modal logic for the description of programs and processes based on operators to express necessity and possibility (Kozen and Tiuryn, 1990) – to address safety (necessity of behaviour) and liveness (possibility of behaviour) aspects of service process behaviour. The idea behind this is that roles can be interpreted as accessibility relations between states, which are central concepts of process-oriented software systems. We have investigated this from a theoretical perspective in (Pahl, 2007). Although we have introduced an extended role expression sublanguage, the combination of operators still remains decidable.

A more complex example shall illustrate the description of service processes in terms of offered individual services and the connectivity.

process BankAccount
    service
        Login (no:int,user:string) : bool
        CheckAcc (dest:int) : bool
        Logout (no:int) : void
        Balance (no:int) : real
        Lodgement (no:int,sum:real) : void
        Transfer (no:int,dest:int,sum:real) : void
    connector
        Login; !(Balance+Lodgement+(Transfer×CheckAcc)); Logout

We assume that the bank account process uses services provided elsewhere, such as login, logout and the check-account function:

process AccountRegistry
    service
        export CheckAcc (no:int) : bool
    connector
        !CheckAcc

process LoginServer
    service
        export Login (no:int,user:string) : bool
        export Logout (no:int) : void
    connector
        !(Login+Logout)

This describes a central online banking process, defined in the *BankAccount* process, that uses (or imports) other services to fulfil its tasks, i.e. *BankAccount* is a client of *AccountRegistry* and *LoginServer*.

Domain models complement the process models at the BML layer. Domain models often form the starting point for software development. Central concepts of a domain

have to be identified and described in their properties (as relationships to other concepts). For the banking sector, we identify concepts such as account or account user (which are static objects) and account login, lodgement, and transfer (which are dynamic activities or processes). The capture of these objects and processes is particularly important. Processes for instance are described in terms of the objects they process. The resulting model is a semantic net consisting of (two types of) concepts and roles relating these concepts. A taxonomy, a basic ontology providing a structured vocabulary, shall form the domain model for this investigation.

4.2 A Semantic BPMN Enhancement

BPMN is a standard notation to model business processes that is especially suitable for business analysts. In order to describe processes, the notation provides constructs to represent business events, process steps, elements that control the order of the process steps and elements that groups the previous constructs into organisational roles. It allows the representation of processes owned by one organisation, but also of shared processes between two or more organisations. BPMN allows hierarchical modelling of processes. It provides support to model information flow between process steps.

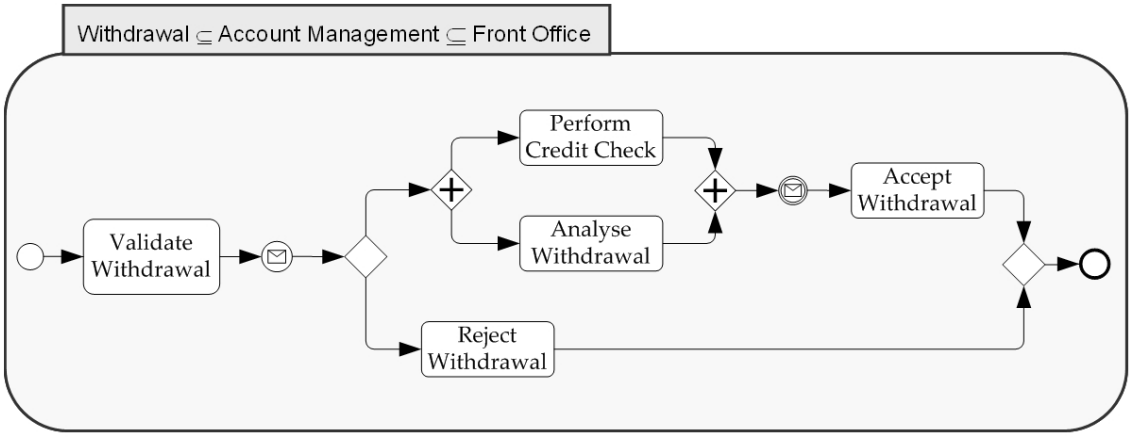

Fig. 9. A Semantic BPMN Extension. (© 2008, Claus Pahl. Used with permission.)

The provided modelling constructs in BPMN do not support domain modelling. In order to provide traceability between the informational view and the process views at business level, and subsequently at software levels, we have extended BPMN to allow traces with domain model elements and application components through an ontology-based model. The traces provide flow dependencies information among services that guide the technical services composition. Fig. 9 shows an extract of the semantic extension of the BPMN notation. Central to ontologies at this layer is the intrinsic specification of process behaviour in the ontology language itself. Behaviour specifications based on the descriptions of necessity and possibility are directly accessible to logic-based methods; behaviour-related inference of service properties is possible. Using the service ontology model from the previous section, we add to BPMN:

- the description of effects as pre/post-conditions, which includes guard conditions that allows acceptance to be decided if a check has been performed successfully:

PerformedCreditCheck() = true -> Accept Withdrawal . WithdrawalAccepted()

- a domain model taxonomy that relates concept terms used in the process model to the domain ontology – here *Withdrawal*, *AccountManagement* and *FrontOffice* are BML-level information element organised into a hierarchy:

  $Withdrawal \sqsubseteq AccountManagement \sqsubseteq FrontOffice$

- textual representation (and formal semantics) of processes in terms of the connector-based role language, which would result in

  Validate Withdrawal ;
  (((Perform Credit Check + Analyse Withdrawal) ; *Accept Withdrawal)*
  $\times$ *Reject Withdrawal)*

We have only considered core BPMN constructs here in order to illustrate the principles of ontology-based process modelling. We will consider further concepts later on.

4.3 Process Patterns

Business patterns identify the interaction and structure between users, business processes, and data. Specific integration and composition patterns at a more technical level address enterprise application integration and capture reliable solutions. Patterns as abstractions of quality implementations are discussed from two perspectives:

- ontology-based business pattern representation to capture common business domain and process solutions; business patterns identification guides the services definition;
- ontology-based SOA pattern representation to capture quality IT and software architecture solutions; SOA patterns instantiation improves architecture designs.

The pattern identification and matching process is based on ontology-based matching.

Process patterns are essentially common workflow patterns. These can be operator-oriented, e.g. a multi-choice pattern that allows the selection of a number of options instead of an exclusive selection based on the basic choice operator (Aalst, Hofstede, Kiepuszewski, and Barros, 2003). The other category consists of application context-oriented and often more complex patterns derived from the business context. The bank account process has a common set of account usage activities at its core that can be represented in the form of a pattern:

```
process AccountProcess
        service
                Balance (no:int) : real
                Lodgement (no:int,sum:real) : void
                Transfer (no:int,dest:int,sum:real) : void
        connector
                 Balance+Lodgement+Transfer
```

Patterns are formulated in the same way as the process connectors. The abstraction that patterns represent is reflected through a subsumption relationship (Baader, McGuiness, Nardi and Schneider, 2003) that we expect to hold between a pattern and a concrete process. Using subsumption instead of instantiation allows us to capture pattern instantiation (e.g. for the operator-oriented ones), but also pattern refinement (e.g. for the domain-specific ones). Process model elements can belong to more than one pattern.

The aim of the pattern support is to identify occurrences of known patterns in a business process. This matching requires technical support, in particular for the formal effect and connector descriptions. Service-based software systems are based on a central state concept; additional concepts for auxiliary aspects such as the pre- and poststate-related descriptions are available. Services are behaviourally characterised by transitional connector roles (state changes) and descriptional roles (auxiliary state descriptions).

The identification of patterns in processes can be implemented based on a graph-theoretic approach, focusing on graphs as the structural backbone of ontologies. The algorithms are based on a graph-centric approach and they consider the restricted vocabulary of different vertical business domains and hierarchical pattern matching. The validation of matching can be based on techniques such as refinement.

In order to support process pattern matching at the SOA level through ontology technology, we need to extend the ontology language. We can make statements about service processes, but we cannot refer to the data elements processed by services. The role (or relationship) expression sublanguage needs to be extended by *names* (representing data elements) and parameters (which are names passed on to services for processing). We can make the bank account *Transfer* service description more precise by using a data variable (*sum*) in pre- and postconditions and as a parameter:

$$balance >= sum \text{ -> } Transfer(sum) \,.\, balance = balance@pre - sum$$

decreasing the pre-execution *balance* by *sum*. (Pahl, 2007) describes the ontological details of the names inclusion in service process ontologies. An illustration of adding a *PolicyApplication* data item is presented in Fig. 10. This also indicates the intended distributed nature of the application into interacting Request and Notification processes.

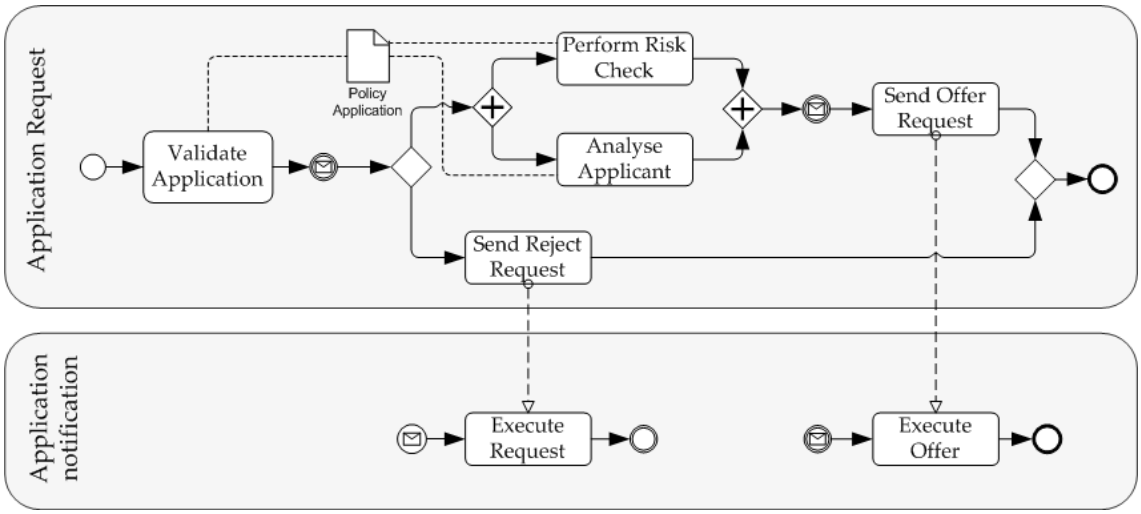

Fig. 10. A Semantic SOA Process Model Example. (© 2008, Claus Pahl. Used with permission.)

4.4 A Technique for Process Pattern Matching

A matching technique needs to be supported by a comparison construct. We already mentioned a refinement notion as a suitable solution. Simulation is the other notion that

we apply. This definition, however, needs to be based on the support available in description logics. Subsumption is here the central inference technique. Subsumption is the subclass relationship on concept and role interpretations. We define two types of matching. These correspond to the two role types – effect and connector:

- Service effect (semantics). For individual services, we define a refinement notion based on the design-by-contract principle, i.e. weaker preconditions (allowing a service to be invoked in more states) and stronger postconditions (improving the results of a service execution). For example

  true -> Transfer(sum) . balance = balance@pre – sum

  matches, i.e. refines

  balance >= sum -> Transfer(sum) . balance = balance@pre – sum

  as it allows *balance* to be negative due to a weaker, less restrictive precondition *true*.
- Process connector (structure and behaviour). For service processes, we define a simulation notion based on sequential process behaviour. A process matches another process if it can simulate the other's behaviour. For example, the expression

  Login; !(BalanceEnq+Transfer); Logout

  matches, i.e. simulates

  Login; !BalanceEnq; Logout

  since the *Transfer* service can be omitted. The provider needs to be able to simulate the process pattern requested by a potential user. Our constructive support for subsumption determination is based on a simulation notion for complex role expressions of the ontology language (Pahl, 2007).

Both forms of matching are sufficient criteria for subsumption between the respective constructs. Matching of effect descriptions is the prerequisite for the assembly of services in architectures and the composition of services to processes. Matching guarantees the proper interaction between composed services in the actual process.

If necessary, a matching variant can be introduced that includes name equality (for individual services) as a matching criterion. For instance, *Login; !(BalanceEnquiry+Transfer); Logout* does not match, i.e. does not simulate *Login; !BalEnq; Logout* due to name inequality, even if the respective semantic effect specifications match. In terms of the case study from Section 3.3, this allows patterns at both BAIL and SAL to be modelled and matched against respective processes (Fig. 7).

5 Transformations and Alignment

Structured, pattern-based transformations are the solutions to the IT alignment and traceability problem in relation to business architecture models. Not only functionality, but also the quality of implementations is critical to achieve business goals associated to business models. Therefore, we discuss quality in the context of architectural patterns.

5.1 Metamodels and Structural Connectivity

In order to define the transformation between business and technology layer – more concretely between BAIL and SAL layer – we define metamodels for each layer. The metamodels are captured as an architectural pattern language. At the core of the transformation definition is an architecture metamodel ontology that defines the central architectural concepts. These concepts are five core types of architectural elements:

Architecture, *Service*, *Connector*, *Effect*, *Operation*

These are derived from a general concept called *Element* that captures all architectural notions. Services and connectors are core elements of architecture descriptions. Services encapsulate computation and connectors represent communication between the services. Services can communicate through operations. Connectors connect to these operations (operation activation on the client side and operation execution of the server side). Architectures are compositions of services and connectors. Services are described in terms of their effect. Connectors refer to the operations they relate to. This formalises at metamodel level the notions of effect and connectivity. The vocabulary of the five elements is formally defined in terms of a simple logical formulation.

$$\mathit{Service} \sqcup \mathit{Connector} \sqcup \mathit{Effect} \sqcup \mathit{Operation} \sqcup \mathit{Architecture} \sqsubseteq \mathit{Element}$$

and

$$\begin{aligned} \mathit{Architecture} &= \exists\, \mathit{hasPart} \,.\, (\mathit{Service} \sqcup \mathit{Connector} \sqcup \mathit{Effect} \sqcup \mathit{Operation}) \\ \mathit{Service} &= \mathit{Element} \sqcap \exists\, \mathit{hasInterface} \,.\, \mathit{Effect} \\ \mathit{Connector} &= \mathit{Element} \sqcap \exists\, \mathit{hasInterface} \,.\, \mathit{Operation} \end{aligned}$$

The predicates *hasPart* and *hasInterface* are predefined relationships between architecture elements. An architecture has parts such as service, connector, effect or operation. The existential qualifier describes that these components might exist. In terms of architecture models, these elements are types, i.e. meta-level constraints for a concrete architecture.

5.2 Architectural Patterns and Quality

Defining a process or a pattern is actually done by extending the basic vocabulary of the metamodel layer from the architecture ontology. Again, we use the subsumption relationship rather than instantiation. This allows us to keep all descriptions at one ontological level, thus simplifying the language framework. This metamodel defines the structural elements of processes. Their semantics and process behaviour are defined through the service process ontology. This approach also allows us to introduce the specific types that form an architectural pattern by defining its structural connectivity, which explains a link to quality considerations. The approach shall be illustrated using the hub-and-spoke architectural pattern. This pattern abstracts a system that manages a composition from a single location, the hub. We start with an extension of the hierarchy of architecture types in order to introduce style-specific services and connectors:

$$\mathit{HubSpokeService} \sqsubseteq \mathit{Service} \quad \text{and} \quad \mathit{HubSpoke} \sqsubseteq \mathit{Connector}$$

A standard connector, *HubSpoke*, connects hubs and spokes. Further constraints could limit the number of hubs to one:

$$HubSpokeArchitecture \quad = \quad =1\ hasPart.\ Hub$$

with $HubSpokeArchitecture \sqsubseteq Architecture$. Spokes can be instantiated in any number. These new elements shall be further detailed and restricted to express their semantics. *Hub* and *Spoke* connectivity through input and output ports is defined as follows:

$$Hub \quad = \quad =10\ hasOperation \sqcap \exists\ hasOperation\ .\ Input$$
$$Spoke \quad = \quad =1\ \ hasOperation \sqcap \exists\ hasOperation\ .\ Output$$

This explains that hubs receive up to 10 incoming requests from spokes. A spoke is only connected to one hub. *Hub* and *Spoke* are services of a hub-and-spoke architecture. Each of these services is characterised through the number and types of operations using so-called predicate restrictions on a numerical domain and the usual concept descriptions. The expression $\leq$n is used to express *hasOperation*.(n | n $\leq$ 1). In addition to these structural conditions that define connections between the service types, a number of semantic constraints can be formulated. Disjointness requires the services to be truly different: $Hub \sqcap Spoke = \bot$; completeness requires hub-and-spoke services to be made up of only the two specified types: $HubSpokeService = Hub \sqcup Spoke$.

We define an architectural pattern to be a subtype of the architecture type.

$$PipeFilterPattern \sqsubseteq Architecture$$
$$PipeFilterPattern = \exists\ hasPart.\ (\ PipeFilterService \sqcup PipeFilterConnector \sqcup Operation\ )$$

Architectural patterns can be linked to quality attributes. Some of the advantages of the hub-and-spoke architectural pattern for Web service implementations in terms of quality aspects are, according to (Barrett, Patcas, Murphy and Pahl, 2006):

- Composition is easily maintainable, as composition logic is all contained at a single participant, the central hub.
- Low deployment overhead as only the hub manages the composition.
- Composition can include externally controlled participants. Web service technologies, for instance, would enable the reuse of existing service components.

The main disadvantages of this architectural pattern are:

- A single point of failure at the hub provides poor reliability and availability.
- A communication bottleneck at the hub results in restricted scalability. SOAP messages have considerable overhead for message deserialisation and serialisation.
- The high number of messages between hub and spokes is sub-optimal.

Quality is important to achieve business goals and this quality link can be the driver to select and enforce suitable patterns.

5.3 Transformation Templates

We return to the transformation problem, which we address using pattern-dependent transformations. During the design of service-centric architecture solutions for the EAI

problem, business process models are annotated with information of matched business patterns. Matched business patterns define the boundaries of coarse-grained and business-centric services. Each matched pattern has a predefined transformation providing a process-centric service description. A set composed of the pattern, the transformation and the associated service description is named *transformation template*. Similar ideas are presented in (Baresi, Heckel, Thöne and Varro, 2006). While they follow a structural view, we use a process-centric view.

Transformation templates are the building blocks of an incremental transformation from the enhanced business process models (BAL) that contextualise the integration problem into a service-centric solution (SAL) aligned with the business and applications levels. A transformation template refines an enhanced business process model into a service architecture description. Each element's type from an enhanced business process description has a refined version as an element of a service architecture. Particular transformation templates are defined through transformations at metamodel level. The main refinements of elements from BAIL to elements of the service architecture are shown in Table 1. It addresses mappings between the main architectural elements:

- Service & Operation: mapping of service boundaries (pattern-based abstractions)
- Behaviour: mapping of processes and their elements
- Connectivity: mapping of process connectors
- Effect: mapping of in- and output elements and associated effect specifications

The relations among BAIL elements are preserved by the transformation. These relations provide information about the flow dependencies between business services.

Service implementations are constrained by the applications supporting the business processes. Traces between business model elements and applications support the refinement of business services into more concrete services based on the functionality of the applications (Fig. 4). The refinement of service connectivity processes and patterns is verifiable through subsumption – available for both the process and the architecture connectivity ontology. Corresponding theory has been developed in (Pahl, 2007).

Table 1. Mapping at Metamodel Level for Transformation Templates.

BAIL (business-oriented)	SAL (business-oriented)
Service & Operation: Abstraction	
Process pattern (business process service boundary)	*Business process-centric service* (port operations)
Behaviour: Process elements	
Process description of business process service boundary	Business *service interface protocol*
Process steps of a business process service boundary	*Business process-centric service interface operation*
Connectivity: Connector elements	
Sequence flow	*sequence orchestration connector* (sequential composition within a specific business service provider)
Message flow	*Sequence choreography connector* (sequential composition of business services)
Gateways XOR, OR, AND	*Exclusive choice* (⊕), *choice* (+),

	concurrency (×) *connectors*, respectively
Indication of *Loop* in a process step or sequence flow connection to an "upstream" process step.	*Iteration connector* (!) for operations associated with process steps of the loop framed by the sequence flow connection to an "upstream" process step.
Effect: In/outputs, pre/postconditions	
Inputs and preconditions of the inbound business process step of a business process service boundary. • Inbound business process step is the initial process step(s) framed by the business service boundary defined by a business process pattern. • Inputs are defined by domain model elements traced with process steps. • Preconditions are defined for each process step.	*Inputs and preconditions* of business process-centric service. Respective elements of business process-centric services: • identified service • input elements • preconditions
Outputs and postconditions of outbound business process step of a business process service boundary. • Outbound business process step is the final process step(s) framed by the business service boundary defined by a business process pattern. • Outputs are defined by domain model elements traced with process steps. • Postconditions are defined for each process step.	*Outputs and postconditions* of business process-centric service. Respective elements of business process-centric services: • identified service • output elements • postconditions

As Fig. 4 shows, the transformation template defined based on the mappings in Table 1 addresses the process models only. Another important component of transformation is the application model. Applications are existing system components that provide business and infrastructure support for the implementation of the SAL design. Central aspects defining their transformation that reflect AAL elements shall be outlined:

- Capabilities (abstraction). An application operation can be traced to a process step in a business process service boundary. In order to provide output and postconditions, an application component traced to a process step might depend on other application components. Those components are also part of the mapping to SAL.
- Behaviour (process elements). Process-centric descriptions of application component capabilities are derived from a refinement of the described process in a business process service boundary. New inputs, outputs, preconditions and postconditions from the technical view in BAIL are introduced to the business view at the same layer. They are mapped to a refined business service interface protocol, which considers constraints from the applications.
- Effects (in/outputs and pre/postconditions). Inputs and preconditions of the application component operations are traced back to business process steps within a business process service boundary. Inputs and preconditions of interface operations of the application participant's interfaces and the outputs and postconditions of the application operations are traced to the business process steps within a business process service boundary.

- Connectivity (connector elements). Process-centric application component connectors (sequential composition, iteration, choice, and concurrency) are mapped to sequential composition (;), iteration (!), choice (+), exclusive choice (⊕) and concurrency (×), respectively.

Several model-driven development approaches have followed a strategy of direct translation from business modelling constructs to software constructs, e.g. direct transformation from BPMN to BPEL constructs. However, business models could contain sections that cause deadlocks and other problems for the process execution (Koehler, Hauser, Küster, Ryndina, Vanhatalo, and Wahler, 2008). In LABAS, the transformation from business models to service architectures is based on pattern-based transformation templates. An advantage of using transformation templates is that they can be designed to provide error-free transformations. Nevertheless, this requires an initial step to refine the business process model to match the business model section of the transformation template. In (Ouyang, Dumas, Hofstede, and Aalst, 2007), a control-flow pattern approach for BPMN-to-BPEL translation is presented. The transformation templates in LABAS follow a similar approach, but also consider application domain-specific patterns. The transformations can be semi-automated where the tool takes the burden of carrying out the transformation and the software architect determines, based on a automated pattern discovery, the relevant patterns to be applied and preserved.

6 Discussion

We use an ontology-based notion of patterns to link business processes and service architectures. Our survey of SOA methodologies and tools has identified this as a weakness in current enterprise modelling. As mentioned in the background section, current support for service architecture definition from business process neglects this aspect. Often, only reference models for specific domains are suggested to map business processes onto architectures. In many cases, these are geared towards a provider's tool landscape. In order to achieve an independent, automatable solution, a more structured and systematic approach to architecture integration needs to be taken.

An ontology-based framework needs an initial investment. Clearly, the benefits that we outlined in the Introduction and the beginnings of the technical sections, needs to make the additional efforts acceptable. In our approach, we expect architectures and patterns to be formulated explicitly, in a notation based on ontologies. We have addressed this concern through the following measures:

- a profile for UML activity diagrams is available, described in (Pahl, 2008), which shields the ontology notation and would not require specific expertise from a developer or software architect who is familiar with a common, de-facto standard such as UML – this reduces the skills gap that developers might face;
- higher degrees of automation in terms of service and pattern identification, architecture generation, and architecture traceability – this provides direct return on investment in terms of the explicit architecture specification;
- maintenance support through explicit and documented traces between business processes and service architectures – this is the benefit from investing into explicit pattern identification and transformation.

A central aim of this chapter is to identify the weaknesses in state-of-the-art in business process-driven SOA and demonstrate potential of ontologies in this context. Our contribution in this respect is a framework consisting of an architectural model (LABAS), an ontology language for service process description based on (WSPO) and techniques for pattern modelling, matching and transformation. Although a full development environment implementing these languages and techniques is still in progress, prototype components for this environment demonstrate central benefits. In particular we have worked on a Jena-based tool to process ontologies, combined with the FaCT description logic reasoner; a range of pattern identification algorithms that detect existing patterns in process descriptions based on different degrees of matching; and a transformation tool that extracts connectivity and process aspects from an architecture model and generates a Web service process (in WS-BPEL and WSDL).

These efforts have supported so far the feasibility of the approach and demonstrate that essential development steps can be automated – either fully or embedded into an interactive process with the developer. A specific aspect that we have already mentioned and that contributes to automation is decidability. Our past investigations into the theoretical aspect confirm that the ontology language presented is decidable. However, extensions or variations of the language might change this property, and consequently impact the automation degree. Another aspect in addition to automation is usability. The critical component is the service process ontology, which might cause problems for developers for two reasons: firstly, unfamiliarity with ontologies in general and, secondly, the fact that a non-standard interpretation (or use) of ontological modelling is applied (processes as roles). To overcome this, and increase the usability, we have developed an extension of UML based on a profile for UML activity diagrams.

7 Future Trends

A number of developments will affect the application, but also further research and development of the proposed framework. In particular, we discuss improved maintenance (traceability), support for process ontologies, and model-driven development concerns (transformation customisation) here:

- Traceability Automation. A semantically enhanced solution for Enterprise Application Integration for business processes in service-oriented environments needs to provide a framework that links business and service processes. Consequently, the need to look at service modelling and process transformation emerges. Explicit traces were a key suggestion. However, to fully support traceability and to possibly derive traces automatically, more work needs to be done in terms of ontology integration and mapping, which our use of domain process patterns and architectural pattern ontologies demonstrates.
- Service and Process Ontologies. While ontology-based frameworks to describe structural, static knowledge has been widely explored, the representation of behaviour is less well advanced. In particular, the reasoning support has focused on static relationships such as subsumption and composition. However, representing dynamics directly would require the interpretation of relationships as dynamic dependencies, denoting for example state change. The WSPO framework, which we have used here, is a step in this direction. WSPO and SWSF/FLOWS are more recent process-oriented service ontologies that reflect the need to address

composition ontologically. However, this work also demonstrates challenges. For instance, decidability is difficult to achieve, but a necessity for automated support.

- Customisation of Transformations. In terms of transformation, we have focussed on traceability and preservation. However, model-driven development as an emerging software development approach demonstrates the need to address the customisation of transformation in order to deal with specific requirements in terms of for example quality, platform, or language. In particular, directly quality-driven transformation is a promising idea that would allows resulting architectures to be tailored towards selected quality criteria. We have indicated how patterns and qualities can be linked, but more work is needed to influence quality in complex multi-pattern systems.

8 Conclusions

Knowledge representation and management is increasingly important in all aspects of information technologies. Knowledge can play a particularly central role in the context of EAI and service-oriented software development. The emergence of the Web as a development and deployment platform for software emphasises this aspect. We have structured a knowledge space based on different ontological techniques for service and application integration in service-oriented architectures. Processes and their behavioural properties and patterns as abstractions were the primary aspects.

We have developed a process-oriented, layered architecture based on an ontological model. The composition of process-oriented services based on ontological descriptions is a central activity. While some of the underlying techniques, for instance for matching, are already used in areas such as component-based software development, it is necessary to used widely accepted languages and techniques specific to the Web platform for Web services-based software development. Explicit, machine-processable knowledge is the key to future automation of development and integration activities. In particular, ontologies have the potential to become an accepted format that supports such an integration and automation endeavour for the SOA platform. Comprehensive tool support for all integration activities, however, remains difficult to achieve. We have demonstrated here the benefit of ontology techniques to support selected activities in the SOA-driven EAI context.

References

Aalst, W. M. P., Beisiegel M., Hee, K. M. V., Konig D., & Stahl C. (2007). An SOA-based architecture framework. *International Journal of Business Process Integration and Management, 2(2)*, 91 - 101.

Aalst, W. M. P., Hofstede, A. H. M., Kiepuszewski, B., & Barros, A.P. (2003). Workflow Patterns. *Distributed and Parallel Databases, 14(1)*, 5-51.

Allen, R., & Garlan, D. (1997). A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology, 6(3)*, 213–249.

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services – Concepts, Architectures and Applications*. Berlin, Germany: Springer-Verlag.

Baader, F., McGuiness, D., Nardi, D., & Schneider, P.P. editors (2003). *The Description Logic Handbook*. Cambridge, UK: Cambridge University Press.

Baresi, L., Heckel, R., Thöne, S., & Varro, D. (2006). Style-based modeling and refinement of service-oriented architectures. *Software and Systems Modeling, 5(2)*, 187-207.

Barrett, R., Patcas, L.M., Murphy, J., & Pahl, C. (2006). Model Driven Distribution Pattern Design for Dynamic Web Service Compositions. In *International Conference on Web Engineering ICWE06* (pp. 129 - 136). Palo Alto, US: ACM Press.

Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice* (2nd Edition). SEI Series in Software Engineering. Boston, MA, US: Addison-Wesley.

Bernus, P., Nemes, R., & Schmidt, G. (2003). *Handbook on Enterprise Architecture (International Handbooks on Information Systems).* Heidelberg: Germany: Springer-Verlag.

Daconta, M.C., Obrst, L.J., & Smith, K.T. (2003). *The Semantic Web*. New York, US: Wiley.

Dijkman, R. M., & Dumas, M. (2004). Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems (IJCIS). Special Issue on Service Oriented Modeling, 13(4)*, 337-368.

Djurić, D. (2004). MDA-based Ontology Infrastructure. *Computer Science and Information Systems (ComSIS)*, *1(1)*, 91–116.

Erl, T. (2004). *Service-oriented architecture: Concepts, Technology, and Design*, Upper Saddle River, NJ, US: Prentice Hall.

Erradi, A., Anand, S., & Kulkarni, N. (2006). SOAF: An Architectural Framework for Service Definition and Realization. *Proceedings of the IEEE International Conference on Services Computing* (pp. 151-158). IEEE Computer Society.

Fettke, P., & Loos, P. (2006). *Reference Modeling for Business Systems Analysis*, Hershey, PA, US: IGI Publishing.

Foster, H., Mukhija, A., Uchitel, S., & Rosenblum, D. (2008). A Model-Driven Approach to Dynamic and Adaptive Service Brokering using Modes. *Proceedings of 6th International Conference on Service Oriented Computing ICSOC 2008* (pp. 558-564).

Gacitua-Decar, V. & Pahl, C. (2008a). Pattern-Based Business-Driven Analysis and Design of Service Architectures." *Proceedings of the 3rd International Conference on Software and Data Technologies* (pp. 252-257).

Gacitua-Decar, V., & Pahl, C. (2008b). Service Architecture Design for E-Businesses: A Pattern Based Approach. *Proceedings of the 9th International Conference on*

Electronic Commerce and Web Technologies EC-Web08, LNCS 5183 (pp. 41-50). Berlin: Germany: Springer-Verlag.

Gamma, E., Helm, R., Johnson, R. E., & Vlissides, J. M. (1993). Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Proceedings of the 7th European Conference on Object-Oriented Programming* (pp. 406-431), Berlin: Germany: Springer-Verlag.

Garlan, D., & Schmerl, B. (2006). Architecture-driven Modelling and Analysis. *Proceedings of the eleventh Australian workshop on Safety critical systems and software - SCS '06* (pp. 3-17). Melbourne, Australia, Australian Computer Society, Inc. 248: 3-17.

Gašević, D., Devedžić, V., & Djurić, D. (2004). MDA Standards for Ontology Development – Tutorial. *In 4th International Conference on Web Engineering ICWE2004,* Galway, Ireland. Retrieved December 9, 2008, from http://afrodita.rcub.bg.ac.yu/~gasevic/tutorials/ICWE2004/

Grønmo, R., Jaeger, M.C., & Hoff, H. (2005). Transformations between UML and OWLS. In A. Hartman and D. Kreische, editors, *Model-Driven Architecture – Foundations and Applications LNCS* 3748 (pp. 269–283). Berlin: Germany: Springer-Verlag.

Koehler, J., Hauser, R., Küster, J., Ryndina, K., Vanhatalo, J., & Wahler, M. (2008). The Role of Visual Modeling and Model Transformations in Business-driven Development. *Electronic Notes in Theoretical Computer Science 211*, 5-15. Amsterdam, The Netherlands: Elsevier Science Publishers B. V.

Kozen, D., & Tiuryn, J. (1990): Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, *Vol. B* (pp. 789–840). Amsterdam, The Netherlands: Elsevier Science Publishers.

Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., & Fensel, D. (2005). Web Service Modeling Ontology. *Applied Ontology, 1(1)*, 77–106.

McIlraith, S., & Martin, D. (2003). Bringing Semantics to Web Services. *IEEE Intelligent Systems*, *18(1)*, 90–93.

McIlraith, S., & Son. T.C. (2002). Adapting Golog for Composition of Semantic Web services. *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning*, (pp. 482-493).

Object Management Group (2003a). MDA Model-Driven Architecture Guide V1.0.1. OMG. Retrieved September 9, 2008 from www.omg.org/docs/omg/03-06-01.pdf

Object Management Group (2003b). Ontology Definition Metamodel - Request For Proposal (OMG Document: as/2003-03-40). OMG. Retrieved September 9, 2008 from http://www.omg.org/docs/ad/03-03-40.pdf

Ouyang, C., Dumas, M., Hofstede, A.H.M., & Aalst, W.M.P. (2007). Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research (JWSR), 5(1)*, 42-62.

OWL-S Coalition (2002). DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proceedings of the First International Semantic Web Conference ISWC 2002*, LNCS 2342, (pp. 279–291). Berlin: Germany: Springer-Verlag.

Pahl, C. (2007). An Ontology for Software Component Matching. *International Journal on Software Tools for Technology Transfer (STTT), Special Edition on Foundations of Software Engineering*, *9(2)*, 169-178.

Pahl, C. (2008). Semantic Model-Driven Development of Web Service Architectures. *International Journal of Web Engineering and Technology (IJWET), 4(3)*, 386-404.

Pahl, C., & Zhu, Y. (2006). Semantical Framework for the Orchestration and Choreography of Web Services. *International Workshop on Web Languages and Formal Methods WLFM'05*. Newcastle upon Tyne, UK. Elsevier ENTCS Series.

Pahl, C., Giesecke, S., & Hasselbring, W. (2007). An Ontology-based Approach for Modelling Architectural Styles. *First European Conference on Software Architecture ECSA 2007* (pp. 60-75). Berlin: Germany: Springer-Verlag.

Papazoglou, M. P., & van den Heuvel, W. J. (2006). Service-Oriented Design and Development Methodology. *Int. J. of Web Engineering and Technology, 2(4)*, 412-442.

Payne, T., & Lassila, O. (2004). Semantic Web Services. *IEEE Intelligent Systems, 19(4)*, 14-15.

Plasil, F., & Visnovsky, S. (2002). Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, *28(11)*, 1056–1075.

Semantic Web Services Language (SWSL) Committee (2006). Semantic Web Services Framework (SWSF). Retrieved September 9, 2008 from http://www.daml.org/services/swsf/1.0/

Steen, M.W.A., Strating, P., Lankhorst, M.M., Doest, H.W.L., & Iacob, M.E. (2005). In Z. Stojanovic and A. Dahanayake (Eds.). *Service-Oriented Enterprise Architecture. Service-Oriented Software System Engineering: Challenges and Practices* (pp. 132-154). Hershey, PA, US: Idea Group.

Zdun, U., & Dustdar, S. (2007). Model-driven and pattern-based integration of process-driven SOA models. *International Journal of Business Process Integration and Management, 2(2)*, 109 - 119.