Business Model Driven Service Architecture Design for Enterprise Application Integration

Veronica Gacitua-Decar and Claus Pahl

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

Abstract. Increasingly, organisations are using a Service-Oriented Architecture (SOA) as an approach to Enterprise Application Integration (EAI), which is required for the automation of business processes. This paper presents an architecture development process which guides the transition from business models to a service-based software architecture. The process is supported by business reference models and patterns. Firstly, the business process models are enhanced with domain model elements, application architecture elements and business-level patterns. Afterwards, business reference models and patterns are exploited for identification of software services and their dependencies. The subsequent activities are focused on the transformation of the enhanced business processes toward a service-based architecture that solves the application integration problem.

Key words: Service oriented architecture, enterprise application integration, business process management, reference model, business patterns, software architecture patterns.

1 Introduction

Business process management (BPM) aims to improve productivity, product quality, and operations of an enterprise [1]. BPM encompasses methods, techniques, and tools to support the analysis, design, implementation and governance of operational business processes [2]. Software applications are built or acquired to provide specialized functionality required by 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 supporting the operation of business processes [3]. Increasingly, enterprises are using Service-Oriented Architectures (SOA) as approach to EAI [4]. SOA has the potential to bridge the gap between business and technology, improve reuse of existing applications and interoperability with new ones. Software services can be composed to provide a more coarse grained functionality and to automate business processes [5]. However, if new applications are often created without a structured architectural design, integrating these into a coherent architecture closely aligned with the business domain becomes a

significant challenge. On top of specific architectures, architecture abstractions such as reference models, patterns, and styles have been used to allow reuse of successfully applied architectural designs, improving the quality of software [6, 7]. The continual rise of abstraction in software engineering approaches is a central driver of this work, placing the notion of patterns at business domain and focusing on its subsequent transformation to a software architecture.

The main contribution of this paper is a software architecture approach which provides a tractable and consistent transition from business models to software service architectures.

- The architecture approach is structured in layers. They separate aspects and aid with the maintainability of the architecture. Explicit connections between elements of different layers provide advantageous traceability characteristics, essential for change management. A modelling technique capturing the layered architecture provides coherence between business models and the software architecture. Standardized notation at business and software level [8, 9] promotes a broad use of the approach.
- A potential fully automated architecture development process is presented. Automation encourages reduction of human errors and an increase in the quality of products. Three core activities are performed along the process: modelling, identification and transformation.
- Along the process, architecture abstractions such as reference models and patterns are exploited for identification of software services and their dependencies. A derived advantage of incorporating architecture abstractions is the positive contribution over the maintainability of the architecture. Patterns are aside of the architecture and remain valid as long as no large changes occur at business and application level. Another advantage of our approach is its independence of commercial infrastructure. Large software providers offer support for service-centric solutions based on their own reference architectures, often dependant on technology.

This article is structured as follow. Firstly, a layered architecture structuring application integration problem and the required architectural concepts are presented in section 2. Section 3 introduces a case study and explains the layered architecture development process. Section 4 discusses our approach. Related work and conclusions are presented in sections 5 and 6 respectively.

2 Layered Architecture

A *software architecture* is the design of a system describing its design elements, their organisation, allocation and collaboration, and also it characterizes the behaviour of the system [10]. Architecture abstractions, such as patterns, constrain the design elements and their relations and can be distinguished on top of specific architectures.

This paper uses a layered architecture to structure the application integration problem. An incremental transformation from models at business level to a

service architecture is done using business reference models and patterns. Note that business process modelling, domain modelling and service development are activities outside the scope of this paper. However, they are framing our approach. Fig. 1 depicts the architecture layers, their elements and complementary architectural concepts are utilized in this paper.

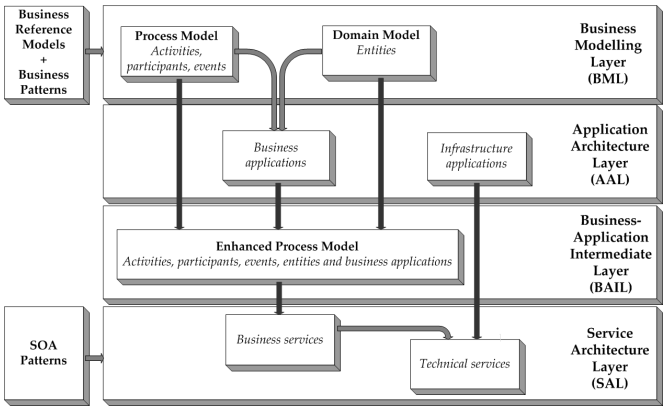

Fig. 1. Layered architecture structuring the EAI problem.

2.1 Architecture Abstractions

The incremental transformation from business models to a service architecture is realised with the help of architecture abstractions such as reference models and patterns. Architectural abstractions are exploited for identification of software services and their dependencies.

Business Reference Model. According with [10] a reference model is a standard decomposition of a know problem into parts that cooperatively solve the problem. They arise from experience and together with architectural patterns can constitute reference architectures. A *business reference model* is a standard decomposition of the business domain, normally provided by standardization organizations. We use reference models as medium to identify business services.

Business and SOA Patterns. In the software domain, architectural and design patterns are common abstractions on top of specific architectures. Patterns have been broadly adopted as a medium to reuse architectural design knowledge and improve quality of software [6, 7]. Design patterns [6] are considered as micro-architectures solving recurring design problems contributing to the overall system architecture. *SOA patterns* solve design problems in the context of service oriented architectures. Some influential efforts such as [11] focus on patterns as abstractions capturing and describing business modelling problems and their corresponding solutions so that the solutions can be reused. Similarly to the work presented in [11] and [6], we consider *business patterns* as micro-

models solving recurring business problems contributing to the overall business model.

2.2 Layers

The layers of the architecture are the Business Modelling Layer (BML), the Application Architecture Layer (AAL) and the Service Architecture Layer (SAL). An intermediate layer (BAIL) containing enhanced process models is also considered. Layers separate involved aspects of the integration application problem, improving the maintainability of the architecture [10]. Explicit connections between layer elements provide beneficial traceability characteristics to our approach, essential for change management.

Business Modelling Layer (BML). The BML constrains and drives the integration of software applications. It has two main models: the business process model and the domain model. While business process models capture the dynamics of the business, domain models capture structural relations between business concepts. Among process modelling notations, Business Process Modelling Notation (BPMN) [8] has been incrementally adopted over the last few years. We have adopted BPMN as notational basis for business models and their transformation to a service architecture. We have also adopted class diagrams of UML 2.0 [9] for domain modelling due to their suitability for our approach.

Application Architecture Layer (AAL). This layer constitutes the technical scenario of the integration application problem and acts as architectural constraint for technical service definition and composition. AAL has either and inter- and intra- organizational scope, since business processes can span across organizations. In order to define applications in architectural terms (and subsequently software services), we borrow the component and connector view from [12]. This view defines a system as a set of components. Each component has a set of ports modelling its interfaces, which enables the interaction with other components through connectors. We adopt the notation of component diagrams of UML 2.0 [9] for models in this layer.

Business-Application Intermediate Layer (BAIL). In this layer, process models are enhanced with elements of the domain model and applications. Creation of models in the BAIL is the first step toward the development of the service architecture. The notation used is inherited from the previous layers (UML 2.0 and BPMN).

Service Architecture Layer (SAL). SAL is a container of software services structured in a service architecture. Software services, called only services in the rest of the paper, are software components capable to of performing a set of offered tasks. Services might delegate their responsibility of performing tasks to other services or applications and they can be composed to provide a more coarse grained functionality. In this paper we discriminate between business

and technical services. Business services abstract activities or business entities from the BML into the SAL. Technical services abstract functionality and data provided by the AAL into the SAL, as well as, functionality required to manage technical issues such as security, messaging, etc. We adopt the UML 2.0 notation for components diagrams [9] for representing services and their connections.

3 Layered Architecture Development Process

Development of service architectures requires consideration of business and technical aspects. We follow an intermediate approach using elements from the business and application layers as starting point for the development of the service architecture. The aim is to provide an EAI approach which maintains coherence between the business domain and its supporting software. The layered architecture development process has the potential of been fully automated. A modelling technique capturing the layered architecture provides coherence between business models and the resultant software architecture. Along the process, architecture abstractions are exploited for identification of software services and their dependencies. Incorporation of architecture abstractions contributes positively over the maintainability of the architecture, since they are aside of the service architecture and remain valid while no large changes occur at the business and applications level.

3.1 Case Study

The case study involves a *billing and payment process* representing a typical process where consumers and businesses (C2B) interact. Fig. 2 shows the high level business process. Three participants (roles) are exhibited at this level, *customer*, *banks network* and *utility company*. Periodically, a utility company bills their customers with an amount of money corresponding to the consumption of delivered services. Customers receive their bills and decide the payment. A payment[1] on the date due will eliminate the debt of the customer, otherwise the debt is accumulated. Remittance information is sent by the bank's network to the customer and the biller after the payment transaction is completed.

3.2 Description of Development Activities

The main activities of the development process are: *modelling*, *identification* and *transformation*. *Modelling* activities enhance models from one layer adding new elements and relations to the models in the same layer or anther layer. *Identification* activities are performed to identify business and technical services; and also to identify suitable business reference models and patterns. *Transformation* activities apply a set of rules to transform an original model in a new model which incorporates components emerged during the identification activities.

[1] For the sake of simplicity this example shows only a bank transfer as a medium of payment.

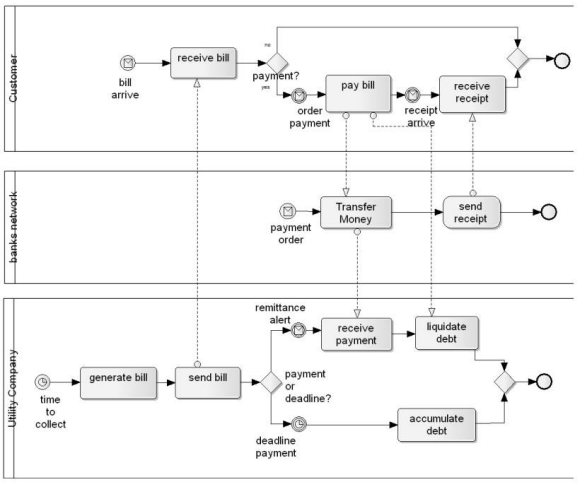

Fig. 2. Billing and payment process.

3.3 Development Process

The presented development process is a systematic approach for designing service-centric solutions for EAI. It consists of a set of activities structuring information from the business and IT sides of an enterprise. External information from the business and software community is also added in the form of business reference models and patterns. Process participants can be related with common roles of the IT industry. Business analysts or enterprise architects are suitable for business modelling and business services definition. Software architects are suitable for modelling at AAL, BAIL or SAL. Fig.3 depicts the development process. Numerated activities are briefly explained in the rest of this section.

Business Model Analysis and Augmentation.

Step 1. At the beginning of the development process the business process models are enhanced with elements of the domain model and business applications. Firstly, the high level process model is decomposed into lower level activities. Subsequently, domain model elements are related with the lowest level activities and applications manipulating those domain elements are added. Fig. 4 shows an example of the enhanced process model for the *generate invoice* activity, which belongs to the high-level process model of Fig. 2.

Step 2. This activity is mainly human centric, and involves the identification of an appropriate business reference model and business patterns. Business reference models facilitate the recognition of reusable portions of the business model, setting boundaries for definition of reusable business services. Additionally, business patterns provide information for early identification of dependencies between services. We have selected the electronic bill presentment and payment (EBPP) reference model for the case study. It was published by NACHA, which represents more than 11,000 financial institutions [13]. Based on EBPP, we realise that a process participant is play-ing the role of mediator. Specifically,

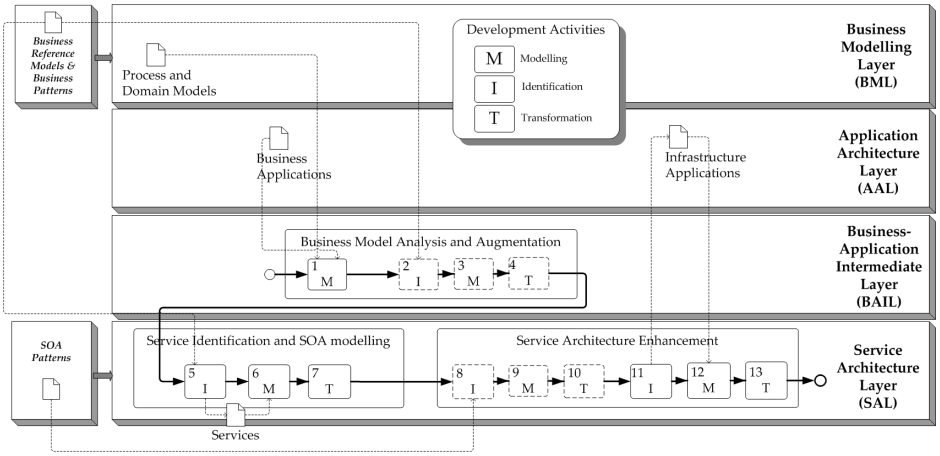

Fig. 3. Layered architecture development process

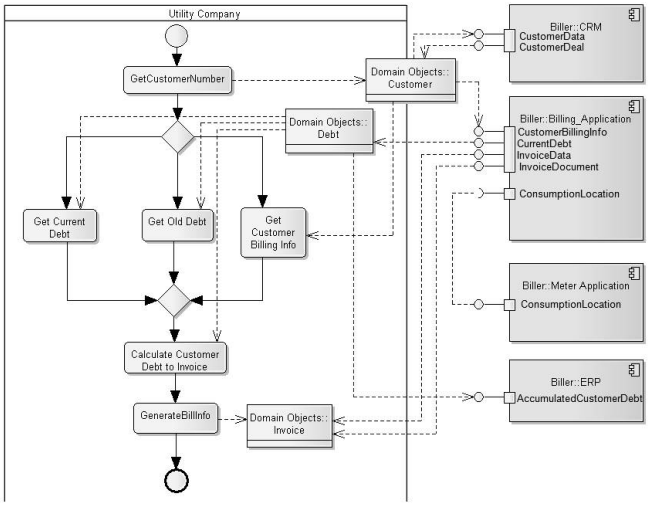

Fig. 4. Generate invoice activity with elements of the domain model and applications.

we identify a *customer service provider* mediating between customers and the utility company. This relation is analogous to the *mediator pattern* from [6]. The relation is illustrated in Fig. 5.

Step 3-4. These two steps involve modelling and transformation activities. They are performed to convert business layer models into models including elements and relations from business patterns. Fig. 6 shows an example where the Customer Service Provider element and the mediator and colleague elements were added to the original domain model after incorporation of the *mediator pattern* (see Fig. 5).

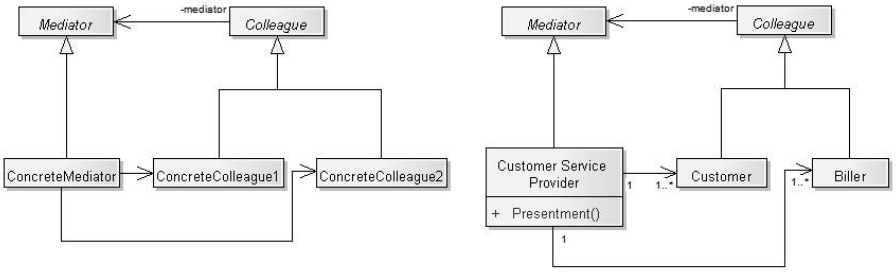

Fig. 5. The mediator pattern and analogy within the billing and payment model.

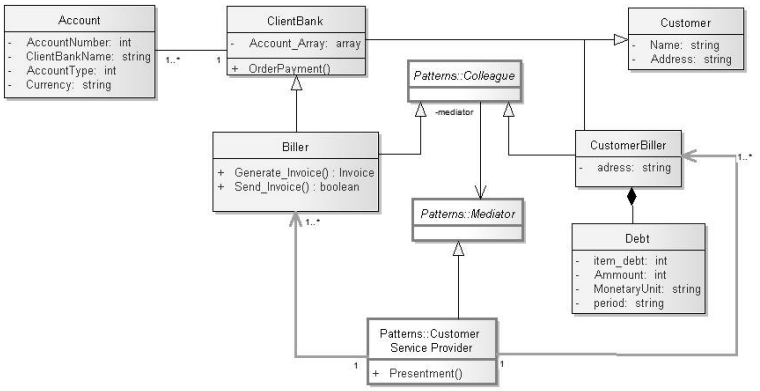

Fig. 6. Extract of domain model for billing and payment model.

Service Identification and SOA Modelling.

Step 5. At this stage services are identified. This activity involves decomposition of the overall integration problem, starting at business level, to subsequently considers architectural constrains imposed by applications. Service design principles [4] such as loose coupling, abstraction, reusability, autonomy, among others are considered. Based on the EBPP reference model, *bill creation*, *presentment* and *payment* processes were considered to be exposed as business services. Since customer is a central concept, we also identify the *customer* business service. *Customer* service abstracts the information of customers from different data sources in the biller side. Sources include a customer relationship management (CRM) application; an enterprise resource planning (ERP) application; and two custom built applications -billing and metering-. Transfer money activity of Fig. 2 is decomposed in three main activities: the initial payment from the customer side, the clearing activity which manages taxes and other charges for transactions between financial institutions, and the final settlement activity. Based on the latter, we define three more fined grained services composing the payment service: *pay* service, *clearing* service and *settlement* service. Two technical services -*tariff*

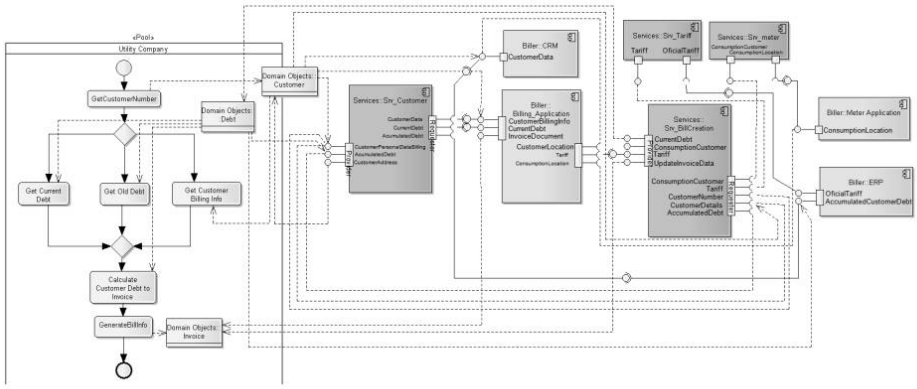

Fig. 7. Generate bill activity with elements of the domain model, applications and services.

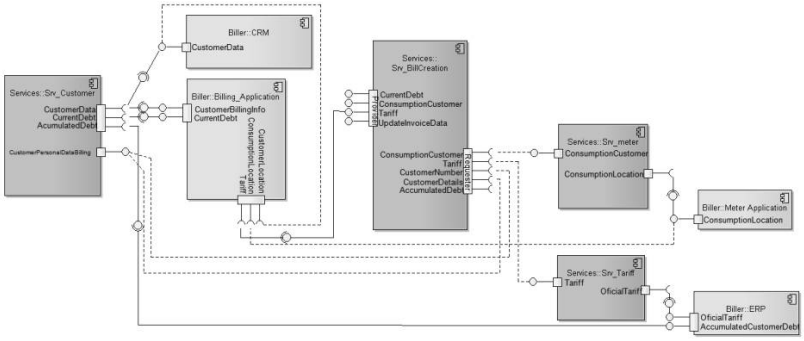

Fig. 8. Software architecture with services and applications.

and *meter* services- are derived for abstracting the tariff rules embedded into the ERP application, and the information about customer consumption managed by the meter application.

Step 6. This step incorporates services identified in the previous step into the enhanced process model. An example is shown in Fig. 7, where services and their dependencies are added into the model of Fig. 4. Note that service dependencies reduce the space of possibilities for service composition to the context provided by the BAIL.

Step 7. In this activity the enhanced process model is transformed towards a software architecture with services and applications. Dependencies between elements of the BML and AAL are hidden. Subsequently, elements of the BML are also hidden. Fig. 8 shows the resultant software architecture after transformation of the process model of Fig. 7.

Service Architecture Enhancement.

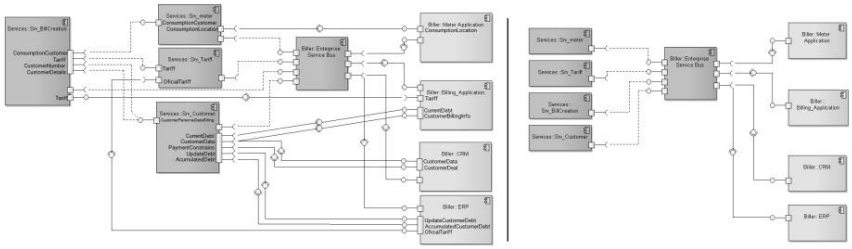

Fig. 9. Service-centric architecture with business and infrastructure applications.

Step 8-10. These three steps focus on the identification and incorporation of applicable SOA patterns for service architecture implementation. When implementing a service architecture, specific components supporting aspects such as service invoking, com-position, security, among others facilities are required. After identification activities, elements of SOA patterns are actually modelled into the service architecture. Trans-formation activities provide consistency to the final service architecture. In the case study for example, technical services in the biller side might be implemented based on the enterprise service bus (ESB) pattern [14], which facilitates exposition of software components on a centralized bus and handles messaging between services, among other facilities.

Step 11. In order to implement the previously incorporated SOA patterns, identification of concrete software infrastructure is required. For instance, the ESB pattern mentioned in the previous step could be implemented with concrete commercial ESB infrastructure.

Step 12. At this stage, the identified infrastructure and required connections are modelled into the AAL. Fig. 9 (left side) shows the ESB component added to the architecture of Fig. 8. Note that redundant relations could appear after inclusion of infrastructure elements.

Step 13. The last step of the development process generates a service-centric architecture where redundant relations appeared in the previous activity are hidden. Fig. 9 (right side) shows the resultant architecture after hiding redundant relations.

4 Discussion

Service-centric architectures have received considerable attention over the last few years. However, the focus has been mainly on improving technological infrastructure and run-time issues [15]. Design-time aspects for development of service architectures have received less consideration. Practical work, indicates that successful implementations of service oriented solutions in enterprises require systematic approaches and maintainable architectures as outcomes [4].

In this paper we provide an architectural approach which provides a service-centric solution for EAI. Coherence between business levels and the software architecture was demonstrated through a case study. Potential automation of activities in the development process was illustrated by means of a structured and guided step by step process, with modelling, identification and transformation activities.

Maintainability is implicitly demonstrated by means of the traceability characteristics provided by explicit connections between elements of the layered architecture. In order to evaluate modifiability, Architecture-Level Modifiability Analysis (ALMA) [16]can be used. ALMA analysis includes change scenario elicitation, evaluation and interpretation. Likely change scenarios come from changes in business processes, domain models and applications. Changes in process and domain models are expressed in terms of creation or elimination of elements and connections in the intermediate layer (BAIL). Mayor changes in process models can affect the definition of business services. Reference models would assist the identification of new business services. Changes in elements of the AAL directly affect the definition of technical services. Those changes might involve adaptation of the implementation of business services. Despite the different nature of the changes, the explicit connections between elements in all layers make traceable those alterations. This characteristic in our approach contributes positively to the modifiability of the final service-centric software solution.

The architectural approach presented in this paper has emerged from empirical work, together with the analysis of a wide rage of systems and methods for development of large scale systems. The authors have been involved in service based solutions projects for mining, governmental and e-learning domains, where coherence between business and software, and maintainability of architectures were key aspects.

5 Related Work

Methodologies such as the presented in [20] and [5] provide a similar guide for building service centric architectures. We go beyond, incorporating modelling support for preserving coherence between business and software aspects. We also incorporate architectural abstractions enhancing maintainability characteristics of the final software solution for EAI. In [14] the authors use patterns and patterns primitives for process-oriented integration of services, however patterns are at service composition level. This kind of patterns might be included as SOA patterns during the last activities of our presented development process. In [20] service identification is driven by analysis of use cases. We use business reference models for decomposition of the business domain, abstracting the identification of services from a particular snapshot describing a temporal situation of the business. Note that selected reference models in our approach are application independent. Software companies can also provide business service definitions based on reference models, however they often relate this definition to their own software applications offer, e.g. [21]. Authors in [15] introduces a framework with

reusable architectural decision models as design methodology for service realization. Architectural decisions in our approach, such as the election of a particular reference model or patterns can be complemented with the framework provided in [15].

6 Conclusion

Methodologies for EAI based on SOA are still maturing. In practice, most approaches start from the application level to afterwards adjust their designs to business levels.

In this paper we have presented an architectural approach for designing service-centric solution for EAI. Our approach is driven by models at business level and take into account the constraints imposed by applications. Coherence between business models and its supporting software was provided by modelling techniques. The used modelling notation provided graphical support and consistency inherited from modelling languages. Business reference models and patterns were used for guiding the identification of software services and early recognition of dependencies between services.

Some of the Activities of the layered architecture development process as the potential of been automated. Automation of transformation activities shall increase quality of products, since human centric errors would avoid. Automation of pattern identification would reduce analysis time, helping specially to neophyte architects and business analysts. Our future work includes automation of transformation and identification activities. Graph based formalization shall give the basis for consistency.

References

1. Lee, R.G. and Dale, B.G.: Business process management: a review and evaluation. BPM J., 4, 214–225. (1998)
2. van der Aalst, W.M.P., ter Hofstede, A.H.M., and Weske, M., Business Process Manage-ment: a survey. In: van der Aalst, W.M.P., et al., (eds.) Int. Conf. on BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Berlin (2003)
3. Hohpe, G. and Woolf, B.: Enterprise integration patterns. Addison-Wesley (2004)
4. Erl, T.: Service-oriented architecture: Concepts, Technology, and Design. Prentice Hall (2004)
5. Papazoglou, M.P. and van den Heuvel, W.J.: Service-Oriented Design and Development Methodology. Int. J. of Web Engineering and Technology (IJWET), 2, 412–442. (2006)
6. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
7. Monroe, R.T., Kompanek, A., Melton, R., and Garlan, D.: Architectural Styles, Design Pat-terns, and Objects. IEEE Software, 14, 43–52. (1997)
8. Business Process Modeling Notation Specification 1.0. BPMI - OMG (2006)
9. Unified Modeling Language: Superstructure. Version 2.1.1. OMG (2007)

10. Bass, L., Clements, P., and Kazman, R.: Software Architecture in Practice. Addison-Wesley (2004)
11. Eriksson, H.-E. and Penker, M.: Business Modeling with UML: Business Patterns at Work. John Wiley and Sons, Inc. (1998)
12. Garlan, D. and Schmerl, B., Architecture-driven Modelling and Analysis. In: Cant, T., (ed.) SCS' 06. ACM Int. Conf. Proc. Series, vol. 248, pp. 3–17. Melbourne, Australia (2006)
13. NACHA - The Electronic Payments Association, http://www.nacha.org
14. Zdun, U., Hentrich, C., and Dustdar, S.: Modeling process-driven and service-oriented architectures using patterns and pattern primitives. ACM Trans. Web, 1, 14. (2007)
15. Zimmermann, O., Koehler, J., and Leymann, F.: Architectural Decision Models as Micro-Methodology for Service-Oriented Analysis and Design. In: Lbke, D. (ed.) SEMSOA 2007. Hannover (2007)
16. Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H.: Architecture-level modifiability analysis (ALMA). J. of Systems and Software, 69, 129–147. (2004)
17. C. Pahl. A Formal Composition and Interaction Model for a Web Component Platform. ICALP'2002 Workshop on Formal Methods and Component Interaction. Malaga, Spain. Elsevier. Electronic Notes in Theoretical Computer Science. 2002.
18. C. Pahl and Y. Zhu. A 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. 2005.
19. C. Pahl, S. Giesecke and W. Hasselbring. An Ontology-based Approach for Modelling Architectural Styles. In European Conference on Software Architecture ECSA2007. Springer-Verlag, LNCS Series, 2007.
20. Arsanjani, A.: Service-oriented modeling and architecture, http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
21. Enterprise Services for Electronic Bill Presentment and Payment https://www.sdn.sap.com/irj/sdn/wiki?path=/display/ESpackages/Home