PATTERN-BASED BUSINESS-DRIVEN ANALYSIS AND DESIGN OF SERVICE ARCHITECTURES

Veronica Gacitua-Decar and Claus Pahl
School of Computing, Dublin City University, Glasnevin, Dublin 9, Ireland.
vgacitua@computing.dcu.ie, cpahl@computing.dcu.ie

Keywords: Service-oriented architecture, Enterprise application integration, Business patterns, SOA patterns, Architectural analysis, Architectural design, Architecture change.

Abstract: Service architectures are an increasingly adopted architectural approach for solving the Enterprise Application Integration (EAI) problem originated by business process automation requirements. In previous work, we developed a methodological framework for the designing of service architectures for EAI. The framework is structured in a layered architecture called LABAS, and is distinguished by using architectural abstractions in different layers. This paper describes the pattern-based techniques used in LABAS for service identification, for transformation from business models to service architectures and for architecture modifications.

1 INTRODUCTION

Service-oriented Architectures (SOA) have been considered a promising architectural approach for Enterprise Applications Integration (EAI). Software services are the building blocks for SOA, and they can be composed to provide a more coarse grained functionality and to automate business processes. The designing of SOA requires a systematic method in order to generate quality-aware, modifiable and business-IT aligned service architectures. The development of service architectures for EAI encompass the analysis of the involved business processes, business information models and applications architecture (Erl, 2004).

Software Patterns are considered in the software community as architectural abstractions that represent encapsulated practical knowledge. The instantiation of architectural and design patterns on a particular architecture design have an impact over the quality attributes of that architecture (Bass et al., 2004). At business level, business reference models and patterns also provide encapsulated knowledge, representing for example best practices, standard business processes, standard information models, among others (Fettke and Loos, 2006).

In previous work (Gacitua-Decar and Pahl, 2008), we developed a pattern-based and business model-driven architecture development framework for the designing of service architectures for EAI. The framework is structured in a layered architecture. The objective of this paper is to summarise the notations and techniques required for such a development framework. The contribution of the paper is the pattern-based techniques for service identification, service architecture development, architecture modification, and transformation from business models to service architectures used in our framework.

2 LITERATURE REVIEW

Different pattern-based techniques have been proposed for analysis, design and evolution of architectures. In (Zdun, 2007), a systematic method to select patterns using language grammars and design space analysis is introduced. Architectural decisions are triggered by a design problem. In (Kim and Khawand,), an approach to specify the problem domain of design patterns is described. The contribution advance in the direction of automatic evaluation of pattern applicability. Often, patterns are not used in an isolated way. Sets of patterns are normally part of organised collections named pattern languages. Pattern languages allow regulated combinations that extends the reach of individual patterns (Buschmann et al., 2007). Architecture transformations are the main architectural concerns after the implementation of a software. Discovering the instances of architectural and design patterns, and providing techniques to modify the architecture in a controlled way, are two important activities for software maintenance. Since the nineties, pattern discovering techniques have been proposed to recover patterns from source code. Several approaches start extracting the classes model from the source code, and subsequently mining the patterns from that model. A review of pattern discovery tech-

niques can be found in (Dong et al., 2007). Non controlled changes in the architecture might interfere with the previously applied design patterns. In (Zhao et al., 2007) a graph-transformation approach to pattern level design validation and evolution is presented. In (Gomes et al., 2003) a set of patterns operators are introduced to design architectures for applications in grid environments. In (Pahl et al., 2007), an ontological-based approach for modelling architecture styles is presented. As such patterns, styles are architecture abstractions. Style modifications and combinations among styles are introduced. Relations between quality requirements and modelling of styles are investigated.

3 LAYERED ARCHITECTURE

The integration problem is structured as a layered architecture named LABAS (Layered Architecture for Business, Applications and Services). An incremental transformation from models at business level to a service architecture is supported by the use of business reference models and patterns. Fig.1 depicts the architecture layers, their elements and involved architecture abstractions.

3.1 Layers in LABAS

Layers separate aspects of the integration application problem. Aspects separation improves the architecture maintainability. Explicit connections between layer elements provide beneficial traceability characteristics, essential for change management.

Business Modelling Layer (BML) is a container for business elements and provides the process models and domain models that represent the context of the business process automation problem. Models in BML are expressed in an enhanced BPMN notation. We have developed a UML profile for the BPMN notation. Most BML constructs are mapped to UML 2.0 activity diagrams constructs. Additionally, the BPMN notation is enhanced by domain model elements.

Application Architecture Layer (AAL) is a container for applications components supporting the business processes in BML. AAL is organized in a process-wide applications architecture. Applications might be owned by different process roles in BML. The applications architecture is modelled with AAL elements of the LABAS profile. AAL constructs are mapped to UML 2.0 component diagrams constructs.

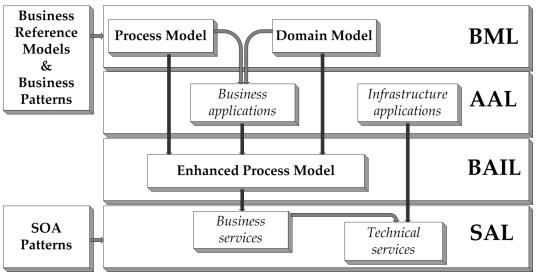

Figure 1: Layered Architecture (LABAS).

Business-Applications Intermediate Layer (BAIL) integrates the elements from BML and AAL in an enhanced business process model. An explicit traceability model relates BML and AAL elements. The traces in BAIL are an integral part of the LABAS profile and follows the trace-tagged traceability metamodel from (Baelen and Berbers, 2007).

Service Architecture Layer (SAL) is a container for software services and is organised in a service architecture that solves the applications integration problem. Service architecture models are expressed with SAL elements of the LABAS profile. The LABAS metamodel and the LABAS profile for SAL are established upon a proposal for a UML Profile and Metamodel for Services (UPMS) in the OMG[1]. Services are categorised in two main types. *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.

3.2 Architectural Abstractions

Domain-Specific Business Reference Models are standard decompositions of a business domain. Reference models arise from experience, and together with business patterns, they can constitute business reference architectures.

Business patterns. They are considered as micro-models detailing standard decompositions of reference models. Two types of business patterns are considered: *process patterns* and *domain patterns*. Note that process and domain patterns have a linguistic and a structural dimension, however process patterns add a behavioural dimension. In this paper,

[1]available at http://www.omg.org/cgi-bin/doc?ad/2007-11-02

we refer only to structural aspects of patterns.

SOA patterns are software design patterns in the context of service architectures. They have a three-part representation with context, problem and solution descriptions. Well-known design patterns, such as the GoF patterns (Gamma et al., 1993), are normally described in a textual form. Formalisations of patterns have been introduced to fulfil requirements of pattern recovery and automatic code generation techniques. In LABAS, SOA patterns enhance the business-driven service architecture derived from successive transformations from models in BML to SAL, into a service architecture that incorporates design solutions for technical aspects, such as service invoking, service composition, security, among others.

Pattern Catalogues. In LABAS, business and SOA patterns are implemented and organised in pattern catalogues. Each pattern in a pattern catalogue contains information organised in a pattern template. The template includes a textual explanation of pattern intent, motivation, participants, consequences, among others, but also models using elements of the LABAS profile. The latter approach allows the exportation of the pattern catalogue as a XMI file, promoting the use of patterns as tool-supported modelling constructs. Information of quality attributes associated to patterns is also included in a section of the pattern template.

4 PATTERN-BASED TECHNIQUES

This section describes the pattern-based techniques used in LABAS. The techniques aim to facilitate the activities performed by business analysts and software architects to transform a business model into a service architecture.

4.1 Business Service Identification

Business patterns are utilised to facilitate the recognition of reusable portions of the business model, setting boundaries for the definition of reusable business services. Business patterns from reference models are a common denominator among organisations in a specific domain, and also within the same organisation that is changing over time. Changeability is related to the ease of an architecture to change, but also with the ability of the architecture to remain invariant after a change agent acts (Ross et al., 2008). The definition

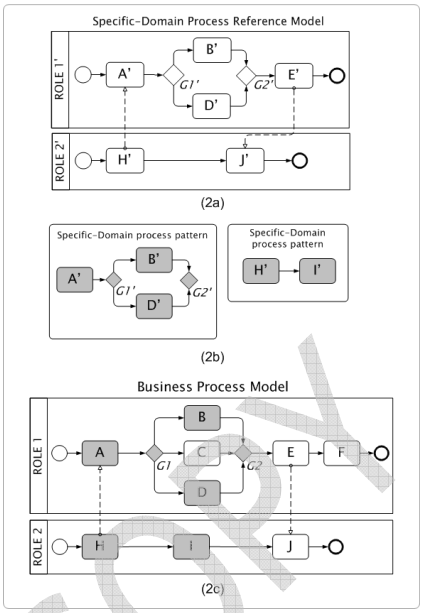

Figure 2: Business process patterns in a process model.

of the business services in LABAS takes into account the latter characteristic.

Business patterns identification could be a human-performed activity, but our aim is to provide techniques to support business analysts and architects with algorithms to automate the identification of business patterns in a business model.

Business Pattern Matching. In LABAS, business models, architectures, business patterns and SOA patterns are represented as graphs. Graphs are a suitable formalisation, since they can capture both: structure and behaviour, and also allow abstractions such as patterns to be related to architectures. The pattern matching technique is based on the matching of the graph (G_{PAT}) representing the pattern, over the graph G_{BM} representing the business model. In order to identify a business pattern, and consequently a business service, an algorithm searches for the sub-graph G_{PAT} within the graph G_{BM}. Fig.2 shows a simplified schema of a reference process model (2a) with process patterns (2b), and a business model containing those patterns (2c).

4.2 Technical Service Identification

Identification of technical services is slightly different to business service identification. In this case, the enhanced process models in BAIL are used to identify common flow structures across the process model. The process model is decomposed until atomic activities are reached. The atomic activities of interest have

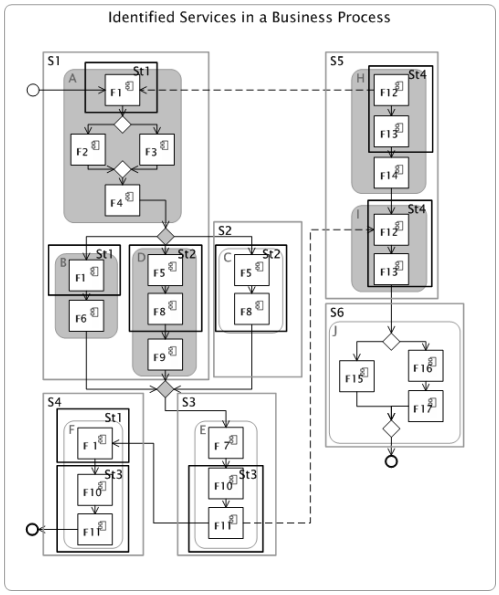

Figure 3: Business and Technical services identified in a enhanced business process model.

a one-to-one relation with the functionality provided by applications in AAL. The Fig.3 shows the process model[2] of Fig.2c, where the activities are enclosing sets of application components that provide the functionalities $F1$ to $F17$. S1 to S6 correspond to business services. S1 and S5 were defined through business pattern matching. St1 to St4 are technical services that can be reused by business services. Technical services encapsulate common flow structures with invocations to functionality provided by application components in AAL[3].

The identification of technical services across process models pursue the fundamental concept of reuse in SOA. The identification of the common control flow structures is support by graph partitioning techniques. Note that the graph representing the enhanced process model has information about their elements types, thus control flow structures involving certain elements types can be further categorised as different technical services types, for instance: data aggregation, calculations, among others.

[2]Note that the proper modelling notation for models in LABAS is not used here because of space considerations. Examples using the LABAS modelling notation can be accessed in (Gacitua-Decar and Pahl, 2008)

[3]In order to simplify the illustration of technical services in the Fig.3, only simple flow structures are depicted. More complicate structures involving e.g. decisions, splits, among other control flow structures, may also be used.

4.3 Pattern-based Architecture Modelling and Architecture Change

Software patterns have been used to allow reuse of successfully applied designs, improving the quality of the software. Analogously, business reference models and a notion of patterns at business level, provide a medium to reuse successful business models. Both, business and software patterns might be applied in practice, only based on the knowledge of business analysts and architects. In the LABAS approach, designers are supported by a repository of experience-based solutions in the form of pattern catalogues, and by implemented techniques allowing the use of those pattern catalogues.

Pattern instantiation. Instantiate a pattern in a model (called here: host model) is a basic task required in pattern-based architecture modelling. Pattern instantiation allows augmentation of a model or architecture through the incorporation of that pattern into the model or architecture. Pattern instantiation involves the creation of pattern elements and relations into the host model, and/or the merging of pattern elements with host model elements. What elements in the host model are merged can be decided by the designer or *suggested*, as is explained below. In LABAS, patterns can be instantiated at BML and SAL to augment or to enhance models in these two layers.

Suggestion of applicability of a Pattern. Pattern instantiation can be done only after the recognition of the associated design problem. Inexperienced designers might not be aware that a pattern can be applied to improve the quality of their designs. If the pattern problem is expressed in terms of elements and relations of the host model, the recognition of a pattern problem could adopt a similar approach as the pattern matching strategy explained in section 4.1. The pattern problem is formalised as a sub-graph, which is a subset of the graph that represents the business model or architecture. Thus, the design problem can be systematically searched, and once localised, it can be suggested to the analyst or designer for the subsequent instantiation of the associated pattern solution.

Pattern suggestion and pattern instantiation are geared in a pattern problem-solution pair. The instantiation of the pattern solution into a model with a design problem (pattern-problem), is formalised as a graph transformation rule. The transformation rule allows the transformation from a graph that represents the model or architecture with a design problem, into a graph that represents the model or

architecture with the instantiated pattern-solution. Note that the modelling of the pattern problem is a key issue for the automatic suggestion of the applicability of a pattern (Kim and Khawand,).

Pattern Comparison. A design problem could have more than one pattern solution associated. In this case, two or more patterns require a comparison. The comparison is supported in LABAS with information about quality attributes associated to the pattern. This information is encapsulated in the pattern consequences section of the pattern template.

Pattern Modification. The description of a pattern solution is a generic description. The instantiation of a pattern might require the modification of the generic pattern solution into one that adjust to the actual model. The adjustments should not interfere with the objectives and consequences of the pattern. The preservation of the pattern properties requires that only allowed modifications can be done. Thus, allowed modifications over patterns make use of a set of pattern modification techniques. Basic techniques involve for example, the increasing or decreasing of the instances of pattern elements, and the increasing or decreasing of pattern elements. After modification of the pattern, validation techniques are applied.

Pattern Combination. Often, patterns are not applied in a isolated way. They are combined to reach a larger scope. Different types of pattern combinations might occurs. For instance, patterns can be *unified* or *embedded*. In the latter case, one or more patterns are subsets of the pattern with larger scope. Combination of two patterns could interfere with the expected contributions that each pattern provide separately. An important issue in pattern combination is to verify that the individual pattern consequences are preserved after the patterns combination. This is difficult to ensure before implementation, but some indication at design-time can be provided with the analysis of possible interferences between associated pattern quality attributes.

Discussion. Requirements for combining patterns could exceed the capabilities of simple techniques such as the union or the embedding techniques mentioned above. Only as an illustration, we use an analogy with relational algebra. Basic operations as projection and selection in relational algebra are not enough in some practical uses for data base queries. Composition of operators is a solution to the restrictions of the basic operations. Analogously, combination of pattern techniques provides a medium to satisfy more complex requirements imposed by architecture modelling and modifications.

Several issues can also be discussed from a practical point of view. For instance, available business models, architecture designs and patterns might have different levels of details. However, the increasing use of enterprise architecture frameworks, such as the well-known Zachman framework (Sowa and Zachman, 1992), have encouraged the development and maintenance of business and software architecture models, together with associated reference architectures and reference models. We can assume that models, architectures and their associated patterns exist, and with same level of granularity. Note that this paper leaves out of its scope, process simulation and linguistics considerations for pattern matching and pattern identification techniques. However, the integration of the behavioral and linguistic dimensions could follow similar directions as in (Ehrig et al., 2007) and (Martens, 2005).

4.4 TRANSFORMATIONS

The methodological framework based on LABAS (section 3) explains how to systematically transform a business model into a service architecture. Patterns are actively involved during this transformation. Since a one-to-one transformation from business processes into services is not realistic, a multi-step transformation approach is adopted. Firstly, pattern-based identified business services, which are documented in the enhanced business process model of BAIL by means of tagged values, are transformed into service elements in SAL. Relations among BAIL elements are preserved after the transformation, providing in this manner, information about the flow dependencies between business services. Subsequently, technical services, also documented with tagged values in BAIL elements, are transformed into service elements in SAL. Technical services also have flow dependencies inherited from the BAIL model.

5 CONCLUSIONS

In this paper we have outlined the necessary notational elements and pattern-based techniques used in our methodological framework for developing service architectures for EAI. Traditionally, the creation of architectures have only focused on structural descriptions. Instead, the focus in this paper has been on processes and constrained architectural descriptions. The continual rise of abstraction in software engineering approaches was a central driver, placing the notion of

patterns at business domain level and focusing on its subsequent transformation to a service architecture. The LABAS architecture and its associated methodological framework have as an ultimate goal, the creation of service architecture solutions for EAI with improved changeability characteristics, while maintaining coherence between the business model and the software architecture. Explicit traceability between elements of different layer in LABAS contribute to the coherence between the business and the software levels. The improved changeability characteristics of the architecture solutions are achieved by using architectural abstractions. Their use is enabled through the pattern-based techniques described in this paper. The techniques are utilised for software service identification, for business model to service architecture transformations and for architecture modifications.

Our future plans include the use of the Architecture-Level Modifiability Analysis (ALMA) method (Bengtsson et al., 2004) to evaluate the architecture solutions created with LABAS. In (Gacitua-Decar and Pahl, 2008) we demonstrate the use of LABAS and discuss the use of ALMA. We also consider the formalisation and implementation of the pattern-based techniques described in this paper. We will investigate sematic and behavioral aspects in patterns. The implementation of techniques is planned to be part of a plug-in for a standard UML modelling tool. The plug-in is complemented with a LABAS profile, compliant with the LABAS metamodel. Additionally, a simplified pattern catalogue at business level, and a SOA pattern catalogue will be developed for evaluation purposes.

REFERENCES

Baelen, V. v. and Berbers, J. (2007). Traceability as input for model transformations. In *ECMDA Traceability Workshop (ECMDA-TW)*, Haifa, Israel.

Bass, L., Clements, P., and Kazman, R. (2004). *Software Architecture in Practice*. Addison-Wesley Professional, second edition.

Bengtsson, P., Lassing, N., Bosch, J., and van Vliet, H. (2004). Architecture-level modifiability analysis (alma). *Journal of Systems and Software*, 69(1-2):129–147.

Buschmann, F., Henney, K., and Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. Wiley and Sons.

Dong, J., Zhao, Y., and Peng, T. (2007). Architecture and design pattern discovery techniques - a review. In *International Conference on Software Engineering Research and Practice (SERP)*, pages 621–627, USA.

Ehrig, M., Koschmider, A., and Oberweis, A. (2007). Measuring similarity between semantic business process models. In *APCCM2007*, volume 67, pages 71–80, Australia.

Erl, T. (2004). *Service-oriented architecture: Concepts, Technology, and Design*. Prentice Hall.

Fettke, P. and Loos, P. (2006). *Reference Modeling for Business Systems Analysis*. IGI Publishing.

Gacitua-Decar, V. and Pahl, C. (2008). Business model driven service architecture design for enterprise application integration. In *ICBIIT2008*.

Gamma, E., Helm, R., Johnson, R. E., and Vlissides, J. M. (1993). Design patterns: Abstraction and reuse of object-oriented design. In *ECOOP93*, volume 707 of *LNCS*, pages 406 – 431. Springer.

Gomes, M. C., Rana, O. F., and Cunha, J. C. (2003). Pattern operators for grid environments. *Sci. Program.*, 11(3):237–261.

Kim, D.-K. and Khawand, C. E. An approach to precisely specifying the problem domain of design patterns. *J. of Visual Languages and Computing*, 18(6):560–591.

Martens, A. (2005). Simulation and equivalence between bpel process models. In *Proc. of the Design, Analysis, and Simulation of Distributed Systems Symposium (DASD05)*.

Pahl, C., Giesecke, S., and Hasselbring, W. (2007). An ontology-based approach for modelling architectural styles. In *ECSA 2007*.

Ross, A., Rhodes, D., and Hastings, D. (accepted 2008). Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value. *Journal of Systems Engineering*.

Sowa, J. F. and Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture. *IBM Syst. J.*, 31(3):590–616.

Zdun, U. (2007). Systematic pattern selection using pattern language grammars and design space analysis. *Software Practice and Experience*, 37(9):983–1016.

Zhao, C., Kong, J., Dong, J., and Zhang, K. (2007). Pattern-based design evolution using graph transformation. *J. of Visual Languages and Computing*, 18(4):378–398.