Application of Design Patterns in Service Oriented Architecture

Roman Šelmeci*
Institute of Informatics, Information Systems and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 842 16 Bratislava, Slovakia
selmeci.roman@gmail.com

Abstract

Developing system on principles of Service Oriented Architecture (SOA) is not an easy task. Among many techniques used in software development process, Model Driven Development is quite popular. When applying this paradigm properly, we are able to write and implement computer programs quickly, effectively and at minimum cost. Sometimes, this can be quite difficult and specific problems may occur, especially in case when we want to know if system meets the requirements of SOA. The objective of this extended abstract is to investigate whether utilization of informal SOA Design Patterns could offer a solution to these problems. Patterns are transformed into a machine acceptable form, which enable identification of pattern instances in system models. Object oriented analysis and the theories of categories and graphs are used for (semi)formalization of design patterns. According to our results, the method proposes a better utilization of SOA Design Patterns in the modelling of new or existing systems.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.10 [**Software Engineering**]: Design; D.2.11 [**Software Engineering**]: Software Architectures

Keywords

Service Oriented Architecture, design patterns, theory of category and graph, design pattern models, pattern oriented development, SOA Design Patterns

1. Introduction

Principles of Service Oriented Architecture (SOA) bring many advantages into software development [7]. However, its application can fetch along some problems especially when models and Model Driven Development (MDD) is involved. Using the models is not an easy task [9]. We face several elementary problems:

- how to unify different models (or at least how to find common vocabulary)
- how to utilize this unification for effective support of the MDD process and
- how to support models verification (debugging)

According to our opinion, one concept which could bring required results and improvements is an application of design patterns within MDD. Natural requirements of every engineering discipline − reusing some "good practices" of the given field − are presented in design patterns.

SOA Design Patterns are currently published only in informal text form. This kind of pattern description brings some limitations[1]:

- users of the pattern must be aware of the existence of the pattern
- they have to know how to apply it,
- it is highly probable that every user of pattern creates a slightly different solution,
- it is difficult to manually modify all areas of the solution which were affected by the pattern and
- manual application of the pattern may bring a mistake.

Therefore, design patterns representation is critical to their successful application.

The structure of this extended abstract is as follows. In the second section, we will briefly introduce existing approaches for SOA Design Patterns utilization and open problem are introduced. In the third section, we will propose our method for application of SOA Design Patterns in MDD. In the fourth section, we will describe our experiments and the last fifth section contains conclusion.

*Recommended by thesis supervisor: Assoc. Prof. Viera Rozinajová
To be defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on August 20, 2018.

2. Background

Existing research sources offer only few existing approaches for application of SOA Design Patterns. Authors in [16, 14, 15] propose a method for formalization of limited patterns group with application of SoaML and Event-B. SoaML is used for (semi)formal representation of patterns in models and Event-B is used as a method for verification if pattern in models are correctly used. This approach enables modeling structural and behavioral aspects of patterns. Another approach uses ontology [12]. Authors transform patterns into ontology which suggests pattern according to user requirements in form of questions and corresponding answers. Both of these approaches has some limitations. The first one has limited application only for message oriented patterns and the second one does not allow to verify system models but only suggests patterns according to user requirements. In addition, user has to apply pattern by himself/herself. The last approach [4] declares representation of SOA Design Patterns only by Service Component Architecture (SCA) standards and patterns are formalized by utilization of their Domain Specific Language (DSL) for "rules cards" based on SCA metrics. A rule describes a metric, a relationship, or a combination of other rules using set of custom operators. This approach supports identification of pattern variants in SCA model but does not support modeling with pattern involved.

SOA Design Patterns are influenced by patterns from other areas. Therefore, we studied also several approaches from object oriented paradigm, service iteration patterns and enterprise integration. After having done an analysis of different methods, we came to the following conclusion:

Approaches for the object oriented design patterns are mostly focused on generating code in specific programming language. SOA Design Patterns often appear in a higher abstract form, which mostly can not be directly converted into executable programming language code.

Petri's or Open nets used for Service interaction patterns are closely linked to a concept of services communicating through messages and connected channels. SOA Design Patterns publication contains many categories which are not concerned with messages, these approaches are not suitable for all of them.

SOA Design Pattern are published in informal text way. Therefore, if we want to reach better utilization of computers with its manipulation, we need to define at least some (semi)formal representation of patterns. If we would like to support modelling of architecture in SOA based systems, one of the most important sections of design pattern definition is a structure. Our main goal is to propose an approach for better application of SOA Design Patterns [7].

We set following requirements for new method:

- Propose a transformation process of SOA Design Patterns from informal text representation into new computer acceptable form.
- Include support for representation and identification of SOA Design Patterns which enable its application during modeling of system.
- Include support for modification suggestions of system's model according to rules from pattern application in order to remove well known bad design structures.
- Enable evaluation of design quality according to identified design patters in models.

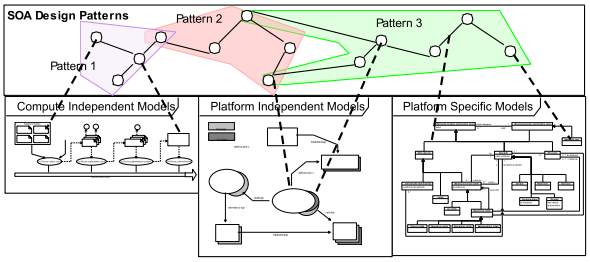

Figure 1: Overview of proposed approach. Pattern participants and relationships are used for annotating models on different levels of abstraction. This enables identification of pattern instances and eventually correctness checking of whole design with patterns.

A language independence in modeling language and also supporting tool is main criterion for method implementation.

3. Our Method for SOA Design Patterns Application

Rigorous and precise structural modeling of SOA design patterns could be reached by a graph theory and category theory.

Graphs, as bases for structural modeling of patterns, support quite easily patterns transformation to required model notation. Graph algorithms and databases, which could bring flexibility and tool support to application of patterns, can be also used. This utilization of graphs also supports describing (nested) variable sub-models as well as inter-pattern synchronization across several diagrams or models synchronization [11]. Graph grammar could be also used for (semi)automatic transformation of models.

Our approach is based on approaches from [2, 6] and object oriented analysis. It defines common vocabulary for annotating system models with extra information and enables reusing of "good practices" from patterns in development.(Fig. 1).

The authors in [2] propose a language-independent approach for visual patterns modeling which could be used in MDD. They applied their approach to a formalization of object-oriented patterns and provide simple examples of how to apply their approach to other kinds of design patterns like work-flow patterns and enterprise integration patterns (EIP). However, the authors did not apply their approach to SOA Design Patterns [7]. Authors also do not propose method for identification of pattern in existing system models and method for modification of the model according to patterns.

First of all, we defined transformation process (Fig. 2) in order to represent pattern in more computer acceptable form. Input for the process is an informal textual design

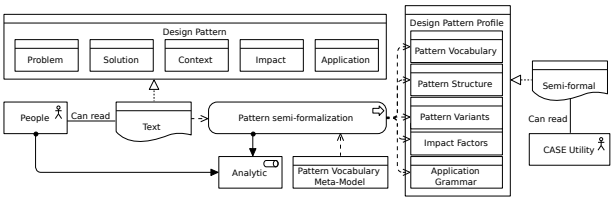

Figure 2: Overview of transformation process.

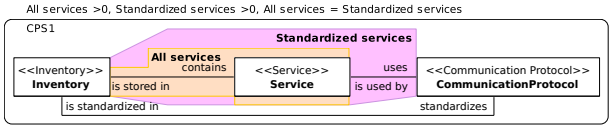

Figure 3: Example of Canonical Protocol Pattern variant definition.

pattern specification and its output is a pattern profile suitable for computer processing.

The descriptions of an individual process steps follow:

1. Identification of pattern participants and theirs roles with object oriented analysis of the informal textual pattern specification.
2. Definition of pattern vocabulary and structure.
3. Identification of possible pattern variants.
4. Definition of new pattern profile with sections for problem and solution variants, pattern context, pattern impacts and pattern application.

Pattern as well as its instance can have many variations ([13]). We need to find out how we could describe these variations as precisely as possible. Authors in [2] propose so-called *variable pattern* for purpose of capturing variation of design pattern instances. Individual pattern variants are modeled according to structural constrains from created pattern vocabulary. Fig. 3 shows an example of one variant for the *Canonical Protocol Pattern*. Figure contains variant for situation when all services in inventory use standardized communication protocol.

Some patterns like *Enterprise Inventory pattern* can be applied only in specific *context* - within small to medium size organization with legacy systems and sufficient resources. All pattern variants of this pattern must conform this conditions. However, if we add these conditions directly into each diagram, diagram could be unreadable and also could contains cross-cutting concerns. Pattern context is defined as extra diagram and synchronized with individuals variants.

Pattern variant has unique *impacts* on the design. If designer knows these impacts, he/she can choose design with required characteristic and quality. Impact can influence some of participants in instance of pattern variant (or only subset of them) in specific time and duration.

The application of pattern is defined as a transformation process from problem variable pattern into solution variable pattern with utilization of typed attributed graph transformation system (GTS) and graph grammar [6].

Now, we have method to describe SOA design patterns in more (semi)formal representation. However, if we want to reach a better application of the SOA design patterns in MDD, we need to find solutions for another problems. The first problem is concerned with a format variability of the system models. Each stakeholder in process of SOA based system development could use different notification for description of his/her parts. Transformation of SOA Design pattern into computer acceptable form makes it possible to use a common vocabulary for annotation of objects in different models which results in easier understanding of these labels.

If we combine all pattern vocabularies from all transformed patterns, the result is meta-model for modeling all parts of SOA based system in which elements from transformed patterns are participating (and pattern instances can be identified). Our meta-model also allows us to check structural correctness of the system or to create graph grammar that conforms to the input meta-model ([10]). Utilization of meta-model enables semi-automatic modification of the model by applying transformation rules defined in patterns profiles. Pattern are added to the existing models with utilization of pattern-annotated model [3].

The second problem is how to detect pattern variant in system design to find any possible problems in it. Moreover, elements forming design patterns need to be located, interpreted and connected in the right manner. How to do it?

Authors in [2] do not offer algorithm for detection of design pattern instances because they manually select desired pattern and create its instance during modelling of the system. They only propose algorithm for pattern satisfaction. It performs a depth-first traversal of the *Emp* tree, counting the occurrences of the variable parts along the way and finally check if these occurrences satisfy the given constraint of variable parts. Some steps of this algorithm rely on graph pattern matching. Expansion set is a core part in the process of a pattern satisfaction. Model satisfies a variable pattern in case some pattern expansion is found in this model.

Unlike the approach described in the [2, 3], we do not use pattern instances during designing phase, instead we assume that SOA patterns originate from SOA principles [7]. Therefore, we use several system models that are annotated with patterns participants roles and then we look for possible pattern instances. We offer to apply sets of paths for description of variable patterns and unification of a detected path in the pattern-annotated model for detection of variable pattern instances.

In contrast to original approach in [2], designer with our method can start with modeling the SOA based system without any knowledge about all existing patterns. Designer only has to use pattern participants vocabulary. The correctness of the system is verified in (semi)automatic manner with utilization of a knowledge base created from formalized SOA Design Patterns. This results into the benefit that less experienced designer without knowledge about all design patterns could be able to design the system and reach almost the same design quality as a designer who knows all patterns.

4. Experiments and Evaluation

We set up testing environment and conducted several experiments which were focused on different areas:

4.1 Pattern Transformation to Proposed Profile and Creation of a Knowledge Base of Patterns

We experimented with the following SOA design patterns: Canonical Protocol, Canonical Schema, Domain Inventory, Enterprise Inventory, Entity Abstraction, Logic Centralization, Policy Centralization, Process Abstraction, Process Centralization, Rules Centralization, Service Layer, Service Normalization, Schema Centralization, and Utility Abstraction, Redundant implementation. We transformed all these patterns and stored them in the knowledge base prototype. All of these patterns were be able to transform into new profiles.

4.2 Definition of Domain Specific Language for Annotation of SOA Solution with Pattern Vocabulary

Our approach is language independent and can be implemented in different forms. We used it in models with utilization of UML stereotypes but also for experimenting and testing we created a DSL named *soa:Pt* for creating pattern annotated models. In our case, language is focused on modelling of SOA based system domain. As an internal DSL is *soa:Pt* built on top of hosting language Clojure [1]. Semantic model [8] of this language is a meta-model created by combination of all individual pattern meta-models. Cardinality between domain objects is not used because it is defined in individual pattern variants. Each time a new pattern is added to meta-model, new language structure is also added. *soa:Pt* controls if allowed relationships used among domain objects (where inherited relationships are included) fulfills language semantic model.

4.3 Identification and Reconstruction of Pattern Variant Instance with Utilization of Graph Database

Our aim is not to define a new algorithm for detection of path in attributed graph, but to use already existing solution for detection of path in an attributed graph, therefore we use Neo4j[2] - graph database which offers declarative language *Cypher* for path definition and detection. In order to initialize database with data, *soa:Pt* statements are translated into Neoj4 Cypher query.

4.4 Use Cases Testing

We had two types of use case tests. Firstly, we defined several tests for each pattern inspired by case studies from [7]. These tests simulate adequate environment in which pattern variants have to be identified and applied. The second type of testing was performed on a real project. This project is relatively new and its development started from scratch. Service Oriented Architecture principles and micro-services are first-class citizens in this project. The project was developed in the organization, which started up just a short time before its beginning. There is no

[1]Clojure, http://clojure.org/

[2]Neo4j.org, neo4j: World's Leading Graph Database, http://www.neo4j.org/

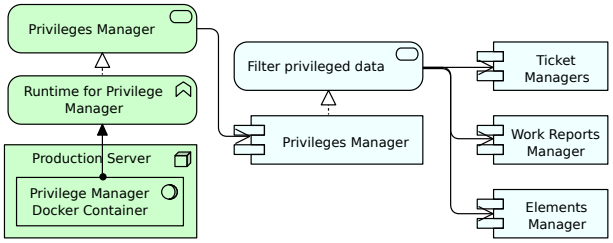

Figure 4: One instance of Privilege Manager is used by several services.

legacy software, its culture is very flexible and development is based on agile methodology. Core development team consists of 5 people. In the first few months of development, the focus was given to prototyping and no architectural documentations was created. Services were created taking into consideration the principles of SOA, but without knowledge of SOA Design Pattern (most developers have a little knowledge about them). After some time we started to create models and documentation of created services in UML diagrams and ArchiMate. These diagrams were taken and rewritten into *soa:Pt*. One of system requirements was:

Business requires to manage user access to element data according to his/her privileges. Data *Privilege Manager* were developed for this purpose. This agnostic service is used in several service compositions (etc. with Ticket Manager, Work Reports Manager and Elements Manager). Only one instance of Data filtering service was deployed on production servers (Fig. 4).

How do we know that this design fulfills all requirements [5] for manageable, efficient and robust system based on SOA principles? We ran our pattern detection method on system models and i.e. we were able to identified this:

Problem was identified in services for managing user privileges and filtering element data. Data Filtering service is used in several service compositions, but only one service instance was deployed on production services. Problem pattern variant for *Redundant Implementation* was identified in actual design. According to the identified design pattern variant, that service is a single failure point for any other services. New design (Fig. 5) was created where *DF Load-balancer* acts as a service facade for additional instances of service. According to design profile we know that this new design brings new impacts of complexity of infrastructure and increases requirements on service government.

We were capable to remove several identified problems from system with utilization of pattern application rules and improve its design quality.

4.5 Evaluation

According to these tests, we are able to identify variants of SOA Design Patterns in existing models only with application of pattern vocabulary, knowledge base created during process of (semi)formalization of SOA design patterns and graph database which contains data from these models. In addition, we are able to suggest design modification which can remove well known problems described in SOA design patterns. Moreover, design quality can be calculated from impacts of identified pattern variants in system.

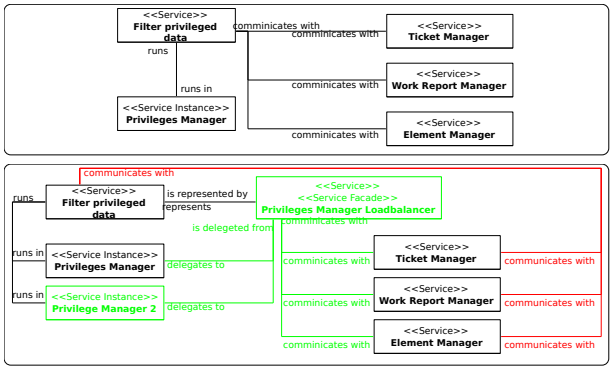

Figure 5: Redundant Implementation example. Top side shows identified problem and bottom side proposed solution (red color represent remove modification and green add).

5. Conclusion

In this extended abstract we have presented a new approach for better application of SOA Design Patterns in the process of SOA solutions modelling. Transformation of existing informal pattern definitions into computer acceptable form with application of category theories and graph theories have been presented. Our design pattern profile was also presented. Pattern profile contains information about impacts of individuals pattern variant, which can be in the future used for calculating overall score of system quality according to identified pattern variants. Application of design patterns with utilization of graph grammar and transformation system are also parts of the pattern profile. Graph grammar supports semi-automatic calculation of alternative solution from which identified instances of pattern problems are removed. Transformation of several SOA Design Patterns lets us to create knowledge base of patterns and define vocabulary which can be used in annotating of SOA solution models. On the top of this vocabulary, we created simple DSL which has a support for syntax verification of models and unify system model into graph representation. This unified representation of the whole system design is used for searching instances of pattern variants stored in knowledge base. We have introduced a new way of description and detection of pattern variants with application of graph paths and existing graph database language. And finally, we experimented with several SOA design patterns in synthetic use cases as well as in the real project. Application on the real project leads to identified problems areas of creating system and permits us to fix it. According to these results we believe that we fulfill our main goals in dissertation thesis.

Acknowledgements. The work reported here was partially supported by the Slovak Research and Development Agency under the contract No. APVV-0208-10; the Scientific Grant Agency of Slovak Republic, grant No. VG 1/1221/12; Operational Programme, ITMS 26240220039, co-funded by the ERDF and STU Grant scheme for Support of Young Researchers.

References

[1] L. Ackerman and C. Gonzalez. *Patterns-Based Engineering: Successfully Delivering Solutions Via Patterns*. Addison-Wesley Professional, 2010.

[2] P. Bottoni, E. Guerra, and J. de Lara. A language-independent and formal approach to pattern-based modelling with support for composition and analysis. *Information and Software Technology*, 52(8):821–844, Aug. 2010.

[3] P. Bottoni, E. Guerra, and J. Lara. Formal Foundation for Pattern-Based Modelling. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, FASE '09, pages 278–293, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] A. Demange, N. Moha, and G. Tremblay. Detection of soa patterns. In S. Basu, C. Pautasso, L. Zhang, and X. Fu, editors, *Service-Oriented Computing*, pages 114–130, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] J. Dick, E. Hull, and K. Jackson. *System Modelling for Requirements Engineering*, pages 57–92. Springer International Publishing, Cham, 2017.

[6] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] T. Erl. *SOA Design Patterns*. Prentice Hall PTR, 2009.

[8] M. Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 1st edition, 2010.

[9] R. France, B. Rumpe, and M. Schindler. Why it is so hard to use models in software development: observations. *Software & Systems Modeling*, Oct. 2013.

[10] L. Fürst, M. Mernik, and V. Mahnič. Converting metamodels to graph grammars: doing without advanced graph grammar features. *Software & Systems Modeling*, Sept. 2013.

[11] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, and T. Engel. Model synchronization based on triple graph grammars: correctness, completeness and invertibility. *Software & Systems Modeling*, Jan. 2013.

[12] L. Liu, P. Miao, L. Pavlic, M. Hericko, and R. Zhang. An ontology-based advisement approach for soa design patterns. In L. Uden, L. S. Wang, J. M. Corchado Rodríguez, H.-C. Yang, and I.-H. Ting, editors, *The 8th International Conference on Knowledge Management in Organizations*, pages 73–84, Dordrecht, 2014. Springer Netherlands.

[13] D. Riehle. Lessons Learned from Using Design Patterns in Industry Projects. In J. Noble, R. Johnson, P. Avgeriou, N. Harrison, and U. Zdun, editors, *Transactions on Pattern Languages of Programming II*, volume 6510 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2011.

[14] I. Tounsi, M. Hadj Kacem, and A. Hadj Kacem. Building correct by construction soa design patterns: Modeling and refinement. In K. Drira, editor, *Software Architecture*, pages 33–44, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] I. Tounsi, M. H. Kacem, A. H. Kacem, and K. Drira. Transformation of compound soa design patterns. *Procedia Computer Science*, 109:408 – 415, 2017. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.

[16] I. Tounsi, M. H. Kacem, A. H. Kacem, K. Drira, and E. Mezghani. Towards an approach for modeling and formalizing soa design patterns with event-b. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 1937–1938, New York, NY, USA, 2013. ACM.

Selected Papers by the Author

R. Šelmeci, Rozinajová. One approach to partial formalization of SOA design patterns using production rules In *FEDCSIS - Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 1381–1384, Wroclaw, Poland, 2018. Piscatawey : IEEE.

R. Šelmeci, Rozinajová. SOA Design Patterns - can they improve the

process of Model Driven Development? In *SCC 2013 Proceedings of the IEEE 10th International Conference on Services Computing*, pages 753–754, Santa Clara, California, 2013. Los Alamitos : IEEE Computer Society.

R. Šelmeci, Rozinajová. Towards More Effective Service Modelling Utilizing SOA Design Patterns. In *International Journal of Web Services Research (IJWSR)*, [in review].