

# One approach to partial formalization of SOA design patterns using production rules

Roman Šelmeci

Slovak University of Technology in Bratislava,  
Faculty of Informatics and Information Technologies  
Ilkovičova 3, 842 16 Bratislava, Slovakia  
Email: selmeci@fit.stuba.sk

Viera Rozinajová

Slovak University of Technology in Bratislava,  
Faculty of Informatics and Information Technologies  
Ilkovičova 3, 842 16 Bratislava, Slovakia  
Email: rozinajova@fit.stuba.sk

**Abstract**—Service oriented architecture (SOA) is nowadays one of the dominant styles in developing new information systems. These information systems often have complex models, which can contain mistakes, or are described by informally. In order to minimize mistakes and to create formal models, patterns as components of software development could be used - according to Model driven development (MDD) principles. Design patterns in SOA have been identified by T. Erl [1]. However, they are represented in form which is suitable for humans, but not for computers. In context of machine processing formal representation of patterns would be advantageous. In this paper we present our approach to partial formal representation of SOA design patterns using production rules. This partial formal representation is useful in searching for mistakes (antipatterns) in models and will enable creating formal models (with patterns) from informal documents.

## I. INTRODUCTION

A PARADIGM of Service oriented architecture (SOA) which is nowadays quite popular, brings flexibility, scalability and faster system development [2].

However, an effective system development consistent with principles of SOA is sometimes quite difficult. One concept, which could - according to our opinion - bring promising results in this context is the concept of design pattern. Adequate motivation for using patterns can be found in [3], [4], [5], [6].

The main representative of the design patterns in SOA can be considered patterns published in [1]. This publication contains a lot of patterns (we can find 83 there), which describe together the best design practices in development of SOA based systems.

Patterns are globally represented by an informal documentation. To work better with the patterns and to be able to have greater benefit of using them (for instance to utilize artificial intelligence techniques in processing them), it would be profitable if the patterns were at least partly converted into the formal representation which is suitable for computer processing. Patterns as components of the software development could be used in MDD or in DSM (Domain specific modeling). In MDD the patterns could be elements of a platform independent model. Patterns could serve us for building a model with higher abstraction level and additional value. In the DSM the patterns would form the basis for the domain specific language (DSL) used for solution specification

in the platform independent form. This language would be useful for experts (architects) who can describe a solution using language which is familiar for them and then use code translator for creating solutions in final platform (which they do not know). As far as SOA design patterns are concerned, we are not aware of research which would deal with a formal representation of SOA design patterns.

In context of functional requirements definition in MDD many activities must be done. Among them definition of boundaries, structure and domain model are very important. Our interest is in creating at least partial formal representation of SOA design patterns and also in identifying patterns and antipatterns in models/descriptions of SOA based system. We believe that this will facilitate better definition of boundaries (objects in pattern are inputs to a solution), better structure (objects in pattern and their relationships define structures), and better domain models (patterns represent high-level abstraction, objects in pattern represent low-level abstraction).

The rest of the paper is structured as follows. Related work is given in section II. In section III, we describe our approach to partial formalization of SOA design patterns and their utilization in process of pattern identification in SOA based solutions. Experiments with our approach and their evaluation are given in section IV. We conclude with suggestion for future work in section V.

## II. RELATED WORK

Theme of formal representation of patterns and application of patterns in software development is not new. There exist more approaches describing how to convert informal pattern specification into a form which would be suitable for machine processing and would allow the automatization of a pattern application in the life cycle of software development.

Important role in the environment of the enterprise architecture and the integration of enterprise applications have Enterprise Integration Patterns [7]. Authors in [8], [9] were inspired also by these patterns. These authors aim to use patterns as platform independent elements in models of integration solution. In [10] a pattern language for process execution and integration design in SOA was defined. This work was elaborated further in [11]. Authors use pattern language and they enhance it by so-called pattern primitives. By means of

pattern primitives they want to achieve that the patterns which are described only in informal form could be used in MDD. Pattern primitives are so an abstract interface for different participants in solution, which patterns provide. For the purpose of MDD support in SOA they create DSLs for every process model. These DSLs are created through metamodel which is based on pattern primitives. In [12] managing architectural decision model is presented. Authors propose formal definition of architectural decision model which supports SOA design by verifying integrity constraints.

We have investigated the possibility of using some of existing principles for partial formal representation of SOA design patterns. We came to the following conclusion:

In the area of enterprise applications integration effort is not focused on a formal way of work with patterns and pattern interpretations. Consequently, individual approaches represent and work with patterns in different ways. We do not consider different approaches as a drawback but we think that better results could be reached if one unified standard for pattern representation could be used.

Approach in [11] establishes formal pattern representation for process integration through metamodels in UML notations. These metamodels are used as bases for domain specific language. The language is however specifically fixated on these patterns and it is not possible to apply it within SOA design patterns.

Proprietary approaches and methods for formal pattern representation in different areas do not conform to our goal - SOA design patterns. Therefore we prefer more general approach to formalizing SOA design patterns.

### III. PARTIAL FORMALIZATION OF SOA DESIGN PATTERN USING PRODUCTION RULES

According to [13] for the success of any method of formal pattern representation there are a few requirements, which are important: (1) preciseness, (2) flexibility and (3) tool support. Using rules for a partial formal representation of SOA design patterns could provide us with a couple of advantages: partial formal representation of patterns will be in a declarative form and in an adequate format. This will enable us to utilize tools for their elaboration. Thus we will be able to reach desired independence on the concrete implementation in the final product.

In order to reach our goal - SOA design patterns partial formalization, we defined transformation process from informal text pattern specification to rule based partial formal pattern representation. Results of this process are production rules and processes of pattern identification, which can be used in expert systems.

We have realized and consecutively experimented with first 14 SOA design patterns. In this paper we chose the Canonical Schema pattern for the purpose of demonstrating activities, which we have performed.

#### A. Transformation process from informal text pattern specification to rule based partial formal pattern representation

We defined transformation process in three steps. The descriptions of individual process steps follow.

1) *Pattern specification*: Pattern can be specified in different ways. We defined pattern structure as 5-tuple:

$$Pattern = \langle Name, Icon, P, I, F \rangle \quad (1)$$

where

$$P = \langle Summary, Draft, Example \rangle \quad (2)$$

$$I = \langle Requirement, Problem, Solution, Context, Impacts \rangle \quad (3)$$

$$F = \langle Effort, Application, Specialities, Relationships \rangle \quad (4)$$

Set  $P$  (Formula (2)) defines presentation specification, set  $I$  (Formula (3)) defines identification specification and set  $F$  (Formula (4)) defines application specification of patterns.

2) *Object oriented analysis*: Production rules are defined in form of *If ... Then ...* Each section contains objects and definition of their required states. In consequence, it is necessary to define objects which together create cores of individual patterns specifications. We use object oriented analysis of the textual pattern specification to identify objects in patterns. We use objects identified in patterns - for the first time in vocabulary expansion and for the second time in pattern formalization rules.

In the phase of object oriented analysis we use UML class diagrams for representing identified domain objects and their relationships. UML diagrams help us to visualize transformed part of pattern specification. Example of UML class diagram with identified objects in specification of Canonical Schema problem is in Figure 1.

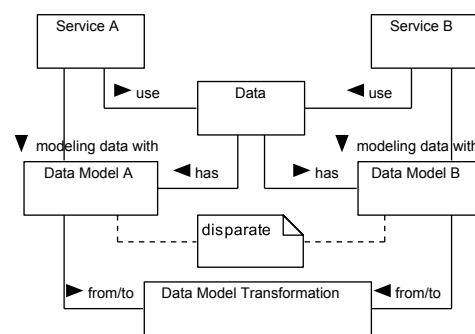


Fig. 1. Sample of some identified objects and their relations in problem definition of Canonical Schema Pattern.

Figure 1 describes a situation when data transformation is needed because two different data models are used for the same data by two services (problem definition in Canonical Schema pattern).

3) *Pattern production rules and identification process:*

Subsequently we create production rules based on identified objects, object relationships and pattern specification from step 2. Production rules (assigned to corresponding group) describe patterns in one of their state - problem, solution, context, and impacts.

Then we define process for pattern candidate/pattern identification. This process is necessary and specifies the moment, when the certain group of rules is executed. General overview of rules activation process is in Figure 2. Input of this process is formed by pattern rules and objects identified in descriptions/models of SOA based solutions. Output of this process is represented by identified patterns, pattern candidates and patterns impacts. Pattern candidates serve us in searching places in the solution which interfere with good established practices of designing system. Pattern impacts define which areas in solution and how are these areas affected by application of pattern.

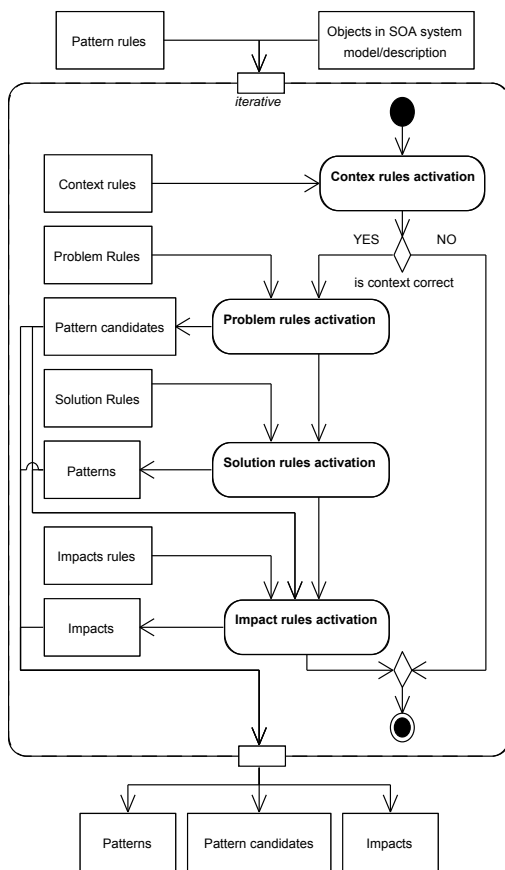


Fig. 2. Overview of rules activation process for identifying patterns, pattern candidates and pattern impacts.

For our prototype implementation we have chosen JBoss Drools, as it has good community and tool support, it is easy accessible and it is an open-source product. Example of JBoss Drools production rules used to identify the Canonical Schema pattern candidate according to problem of data model transformation is Rule 1. This rule is assigned to groups which

**Rule 1** Canonical Schema - problem.

```
rule "Model transformation"
  ruleflow-group: "problem"
  when
    $data: Data()
    $s1: Service($data in data)
    $s2: Service($data in data, this != $s1)
    $m1: DataModel(for == $data)
        from $s1.data_models()
    $m2: DataModel(for == $data)
        from $s2.data_models()
    DataModelTransformation(
      (m1 == $m1 && m2 == $m2) ||
      (m1 == $m2 && m2 == $m1))
  then
    insert(
      new CanonicalSchemaCandidate(
        $data, $s1, $s2));
end
```

describe problem state of Canonical Schema pattern (defined with ruleflow-group). Rule 1 defines problem (and identifies candidate for Canonical Schema pattern) by searching for services, which are using different data models with transformation requirement for the same data.

IV. EXPERIMENTS AND THEIR EVALUATION

All rules for 14 patterns were inserted into the JBoss Drools knowledge base. Then we created JUnit tests. JUnit tests were created in order to simulate adequate environment in which patterns and pattern candidates have to be identified. JUnit tests were inspired by case studies from [1]. For example correct application of Canonical Schema in case study is described as: For the order record, a single Order schema is defined, and it is agreed that all order data passed between Alleywood Java service, Tri-Fold ERP service, and Tri-Fold .NET service will comply to the document structure and validation rules established by this schema.

Case studies are in informal textual form, so we use once more object oriented analysis to identify objects in them. Each identified object was matched to existing term from created vocabulary and inserted into working memory of JBoss Drools prototype system. In Canonical Schema case study of three services (Alleywood Java, Tri-Fold ERP, Tri-Fold .NET), one data (Order) and data model (Order.xsd) were identified. Each partial formalized pattern definition in knowledge base was able to identify corresponding pattern or pattern candidate according to objects in working memory.

Combining same partial UML class diagram from transformation of 14 patterns we got diagram in the Figure 3. This diagram represents a set of domain objects, which can be used for modelling of solution based on SOA principles. The benefit is easy understanding, because they are derived from well known and accepted patterns. Domain experts and

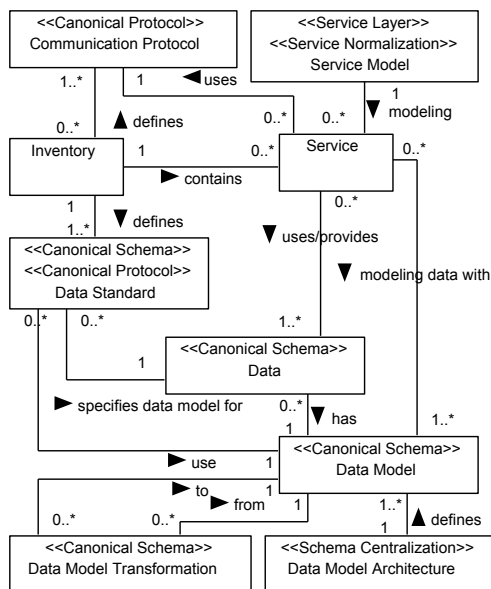


Fig. 3. Diagram of domain objects from some inventory patterns, their relationship and pattern relevancy.

technical experts can use these objects in process of system development. UML class stereotype is used for classifying objects according to SOA design pattern in diagram (alike technique used in [14]). If one object was identified in more than one SOA design pattern, then object has more stereotypes (one for each pattern). Only Inventory and Service have no stereotype, because these objects belong to all patterns we have already formalized. One object can be identified in several SOA design patterns because authors also specified relationships among patterns. These relationships specify how one pattern influences another pattern. SOA design patterns relationships could be useful in creating the DSL, because they specify how patterns could be combined and how this combination influences each pattern. Common objects used in different patterns can represent shared understanding of modelled solution among patterns. These common objects also represent places in a solution where different patterns "work" together. Each pattern manipulates with parameters of objects from their own perspective and consequently models different aspect of solution. All these activities can be expressed by pattern language defined in suitable DSL.

## V. CONCLUSION AND FUTURE WORK

In this paper we have presented our proposal for partial formal representation of SOA design patterns by production rules. We brought in also several approaches for formal pattern presentation from different areas of software engineering. The paper also contains an example of objects and production rules for partial formal representation of Canonical Schema Pattern. Compared to [15] we have created a vocabulary of domain objects with higher level of precision for modelling one part of SOA environment - Service inventory. In comparison with other mentioned approaches, our method enables not only

partial formal representation of SOA design patterns but also the definition of processes for detection of patterns/candidates for pattern.

In our further work we plan to continue in extending the knowledge base of SOA design patterns. The base for our future work is an idea of using design patterns as model components of a software solution. SOA system documentation contains documents in many different styles and formats, so we will create domain specific language for modelling SOA systems design. Keywords in this DSL will be names of domain objects identified in the process of SOA patterns formalization. We believe that using this DSL the following goals can be reached: automatic manipulation with domain objects and their relationships and automatic detection of patterns and candidates in the solution.

## ACKNOWLEDGMENT

This work was partially supported by the Slovak Research and Development Agency under the contract No. APVV-0208-10 and the Scientific Grant Agency of Slovak Republic, grant No. VG 1/1221/12.

## REFERENCES

- [1] T. Erl, *SOA Design Patterns*. Prentice Hall, 2009.
- [2] —, *SOA Principles of Service Design*. Prentice Hall, 2007.
- [3] C. Alexander, *The Timeless Way of Building*. Oxford University Press, 1979.
- [4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [5] M. Fowler, *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1997.
- [6] L. Ackerman and C. Gonzalez, *Patterns-Based Engineering: Successfully Delivering Solutions Via Patterns*. Addison-Wesley Professional, 2010.
- [7] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2003.
- [8] R. Z. Frantz, R. Corchuelo, and J. Gonz ales, "Advances in a DSL for Application Integration," in *Web Application Integration (ZOCO) in JISBD*, vol. 2, 2008, pp. 54–66.
- [9] T. Scheibler and F. Leymann, "From Modelling to Execution of Enterprise Integration Scenarios: The GENIUS Tool," in *Kommunikation in Verteilten Systemen (KiVS)*, ser. Informatik aktuell, K. David, K. Geihs, and W. Brauer, Eds. Springer Berlin Heidelberg, 2009, pp. 241–252.
- [10] C. Hentrich and U. Zdun, "A Pattern Language for Process Execution and Integration Design in Service-Oriented Architectures," in *Transactions on Pattern Languages of Programming I*. Springer, 2009, pp. 136–191.
- [11] U. Zdun and S. Dustdar, "Model-driven and pattern-based integration of process-driven SOA models," *International Journal of Business Process Integration and Management*, vol. 2, no. 2, pp. 109–119, 2007.
- [12] O. Zimmermann, J. Koehler, and F. Leymann, "Managing architectural decision models with dependency relations, integrity constraints, and production rules," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1249–1267, Aug. 2009.
- [13] N. Soundarajan and J. O. Hallstrom, "Precision, Flexibility, and Tool Support: Essential Elements of Pattern Formalization," in *Design Pattern Formalization Techniques*, T. Taibi, Ed. IGI Publishing, 2007, pp. 280–301.
- [14] J. Dong, S. Yang, and K. Zhang, "Visualizing design patterns in their applications and compositions," *IEEE Trans. Softw. Eng.*, vol. 33, no. 7, pp. 433–453, Jul. 2007. [Online]. Available: <http://dx.doi.org/10.1109/TSE.2007.1012>
- [15] L. Tang, J. Dong, T. Peng, and W.-T. Tsai, "Modeling enterprise service-oriented architectural styles," *Service Oriented Computing and Applications*, vol. 4, no. 2, pp. 81–107, 2010.