# Knowledge Representation of Software Design Patterns: A Model Transformations Perspective

Himesha Wijekoon[1], Boris Schegolev[2] and Vojtěch Merunka[3,4]

[1]Department of Information Engineering, Faculty of Economics and Management, Czech University of Life Sciences Prague, Prague, Czech Republic; e-mail: wijekoon@pef.czu.cz
[2]Department of Information Engineering, Faculty of Economics and Management, Czech University of Life Sciences Prague, Prague, Czech Republic; e-mail: schegolev@pef.czu.cz
[3]Department of Information Engineering, Faculty of Economics and Management, Czech University of Life Sciences Prague, Prague, Czech Republic; e-mail: merunka@pef.czu.cz
[4]Department of Software Engineering, Faculty of Nuclear Sciences and Engineering, Czech Technical University in Prague, Prague, Czech Republic; e-mail: vojtech.merunka@fjfi.cvut.cz



**Abstract.** Software design patterns help software developers to design robust and easy to maintain systems as they could be used to solve already identified problems in less time. These software design patterns are mostly textual descriptions which are introduced from books as catalogues. While senior developers tend to know about these patterns by experience, the novice developers need to refer necessary books to get knowledge about them. Therefore, it will be a great help for software development if there are tools to detect and recommend design patterns in software designs. The ultimate advantage will be to automatically apply a design pattern over an initial design to improve it. This will be a model transformation task under model driven architecture. As an initial step towards this goal, in this paper a survey has been done to study the existing knowledge representation techniques used for software design patterns. These representations help automatic reading and processing of software design patterns in order to build necessary CASE[1] tools. Finally, the best design pattern specification techniques are recommended for pattern-based model transformation.

**Keywords.** Knowledge Representation; Software Design Patterns; Model Transformations; Design Pattern Specification; Model-driven Engineering.

[1] Computer Aided Software Engineering


Proceedings of the 9th International Conference on Information and Communication Technologies in Agriculture, Food & Environment (HAICTA 2020), Thessaloniki, Greece, September 24-27, 2020.

## 1 Introduction

A single software program can be designed differently by different software developers. While one design is readable, robust and easy to maintain the other can be complex, hard to read and change. The quality of the design also depends on the experience of the developer. To overcome these issues software design patterns are used extensively in software engineering. Further software design patterns speed up the development without compromising the design quality. These patterns are documented and published by experienced and respected programmers [1, 2, 3, 4].

However there exists some issues about the usability of the design patterns. One of these is the difficulty for the software developers to learn about all necessary design patterns [5]. It is also hard to memorize them all and it is inconvenient to refer these as and when you design software. To overcome these issues, researchers suggest that there should be some tool support regarding the application of software design patterns. For an example some researchers have created recommendation systems to suggest necessary design patterns [6, 7]. There are also studies carried out to automatically detect the existence of design patterns in software designs/code [5, 8, 9].

But the main challenge in this regard has been to model and represent design patterns in a machine-readable form. This paper presents a survey of such knowledge representation techniques used to represent design patterns with an emphasis towards model transformations. Finally, the best option is chosen to be used for authors' ultimate goal, which is to automate/semi-automate application of a design pattern upon a given software model/design.

## 2 Background

There is no common specification for documenting design patterns as different forms have been used so far by pattern authors. Fowler mentions some of these forms such as GoF, Portland Coplien, POSA and P of EAA [10]. GoF Form is the most comprehensive and nicely structured form among these. However, GoF patterns are quite large with many pages of descriptions.

Software developers usually first come up with the models for the business case and then manually refine the models applying the selected software design patterns. For example, software engineers initially design the class diagram for the underlying business case. Then they refine the class diagram by applying necessary design patterns. However, developers should be initially aware of a particular design pattern and its applicability in order to benefit from it. A developer is also responsible for properly implementing the suggested design pattern in his context.

Model transformation techniques can be used to automate or semi-automate this process. They can be utilized to validate the manually refined models as well. Model-driven engineering (MDE) is considered as a well-established software development methodology that uses abstraction to bridge the gap between the problem space and the software implementation [11, 12]. Model transformations constitute the essence of MDE [13].

## 3 Existing Knowledge Representation Techniques for Software Design Patterns

### 3.1 Ontology based Approaches

These approaches are based on the concepts and technologies of semantic web such as Web Ontology Language (OWL) [14] and Resource Description Framework (RDF) [15]. The major contribution in this category for coming up with a representation for software design patterns is from Dietrich and Elgar [16, 17]. They have come up with an OWL ontology named Object Design Ontology Layer (ODOL) [18]. Then they have extended the ontology by adding pattern refinement [18]. Dietrich and Elgar have made a prototype of a Java client to show how ODOL can be used. This prototype software can access the ODOL based pattern definitions which are published online and can detect patterns in Java programs.

Initially ODOL has only supported structural properties of the design patterns. Later, Di Martino and Esposito have further extended ODOL to support dynamic behaviour of the design patterns and also to support cloud patterns [9]. They have revised original ODOL to support pattern categorization. Structure of this augmented ODOL is shown in Figure 1. Further OWL-S [18] has been used to describe behaviour of the pattern. OWL-S defines how the different participants communicate and relate to each other dynamically. Then, they have used this representation of design patterns to come up with a rule-based procedure to automatically recognize design patterns in UML diagrams. Each design pattern is individually converted to a set of first-order logic rules in Prolog by the use of Thea framework [19]. In parallel, the UML model which needs to be analyzed is converted to XMI (XML Metadata Interchange) with the use of existing tools and then converted to Prolog facts using their own tool. Then using Prolog based inference rules specified they could check whether a respective design pattern exists in the UML model under investigation.


**Fig. 1.** Augmented ODOL [8]

ODOL has been also used to represent design patterns in an attempt to automate the Sequence Diagram generation when realizing a particular design pattern [20]. In this case, the OWL based representation of design patterns are converted into Java Expert System Shell (Jess) facts by using a conversion tool, called SweetRules. Then the Jess rule engine is used for the execution of design pattern rules to produce an output sequence diagram.

Ontology based representations of design patterns have following aspects [16].

- Formal definition of patterns
- Machine readable representation
- Modular design that supports the separation of schema and instances
- Compatibility with standard web

### 3.2 Formal Mathematical Logic based Approaches

Formal approaches represent design patterns based on mathematics and formal logic. One of the earliest such approaches is Language for Pattern Uniform Specification (LePUS) [21]. This has only covered the structure of design patterns. On the contrary, Distributed Co-operation (DisCo) specification supports the behavioural aspect of patterns [22]. Taibi and Ngo, who have been inspired from both LePUS and DisCo have come up with Balanced Pattern Specification Language (BPSL) [23]. BPSL combines the First Order Logic (FOL) and Temporal Logic of Actions (TLA) to support both structural and behavioural aspects of design patterns.

Jeon et. al. have also come up with their own formal specification of design patterns [24]. Inference rules are derived for design patterns and then these rules are saved as

Prolog rules. This Prolog rule base is used to find candidate spots in Java program code to apply a certain design pattern.

Formal approaches have been emerged in order to remove ambiguity, support reasoning about patterns and facilitate automation. However as per Khwaja and Alshayeb, formal specifications are not popular in the software industry [25]. They mention that formal specifications are hard to use with less tool support while requiring strong mathematical background from the user.

### 3.3 UML Notation based Approaches

Design Pattern Modelling Language (DPML) is a visual language that can be used to represent software design patterns [26]. DPML can be used along with UML as it also supports instantiation of the design patterns into UML design models. However, DPML does not support dynamic aspects of design and meta-data of design patterns. Other drawbacks are the lacking tool support and notational differences from UML.

Role Based Metamodeling Language (RBML) is another UML based pattern specification language [27]. RBML specifies the pattern solutions as a specialization of the UML metamodel. RBML provides 3 types of specifications; Static, Interaction and StateMachine in order to handle both structural and behavioural aspects of the design patterns. Each RBML specification is an instance of the RBML metamodel. A tool called RBML-Pattern Instantiator (RBML-PI) has been also created to generate a UML model from an RBML pattern specification.

These approaches are complex and require dedicated tool support which is not abundant. Therefore, not many applications could be found in the literature leveraging these specifications. Further these approaches fail to represent the textual descriptions which describe design patterns.

### 3.4 XML Notation based Approaches

Khwaja and Alshayeb have come up with Design Pattern Definition Language (DPDL) based on XML (Extensible Markup Language) to share design patterns between developers [25]. They have reviewed most of other techniques mentioned above and have chosen XML as it is popular and easy to understand. DPDL supports both structural and behavioural views of the design patterns. High level schema of DPDL is shown in Figure 2.

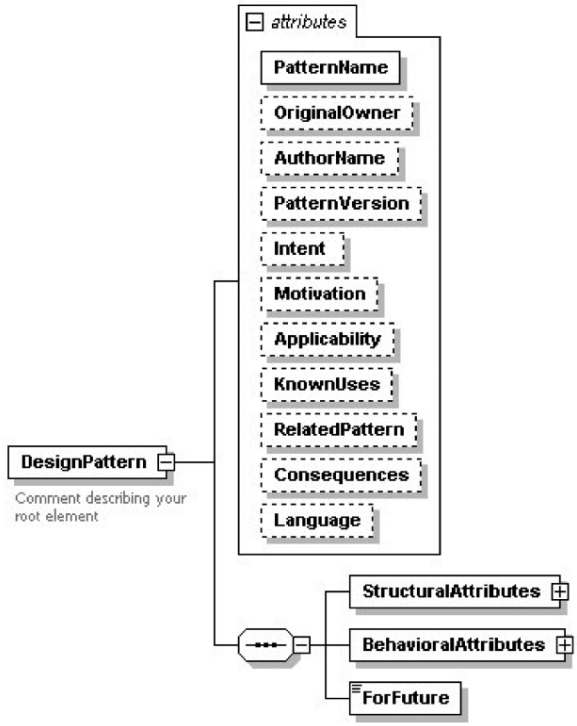


**Fig. 2.** DPDL high-level schema [25].

## 4 Discussion

Meta-Object Facility (MOF) is an Object Management Group (OMG) standard for model-driven engineering [28]. The MOF 2.0 Queries Views and Transformations (QVT) provides a standard for expressing model transformations. Judson et. al. have come up with an approach to automate pattern-based transformations leveraging the MOF 2.0 standards in meta-model level [29]. Overview of their approach is depicted in Figure 3. Initially, Source Model is converted into a meta-model named Source Pattern. Source Pattern is transferred into the Target Pattern using the Transformation Schema and Transformation Constraints. Hence the Transformation Pattern consists of three parts: Source Pattern, Transformation Schema, and Transformation Constraint. Finally, the Target Model is derived from Target Pattern.

The main drawback with their approach is the necessity to prepare a Source Pattern for each design pattern to be applied in a specific scenario. However, it is nice if the freedom is given to the developer to specify the Source Pattern. For an example the developer can choose artefacts from a UML model and ask the transformation engine to transform them by applying a certain design pattern. For an example he/she can mark a certain class in the UML diagram and apply Singleton pattern over it. Then

transformation engine should be robust enough to transform the source model accordingly.

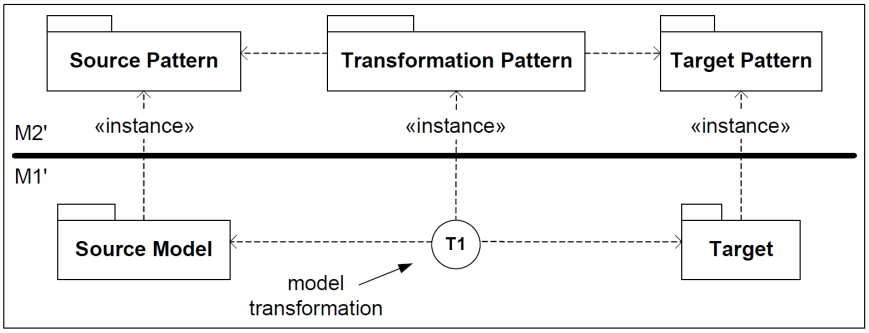


**Fig. 3.** Transformation Overview [29]

However major challenge in this regard is to dynamically generate Transformation Schema and Transformation Constraints for a specific application of a design pattern. This will be not feasible unless the design patterns are represented in a machine-readable format. ODOL + OWL-S seems to be the best choice as per the review in section 4 and it is popularity. The represented design patterns will be not limited to a specific purpose as the definitions can be shared online for public access. Another advantage of this is the fact that OWL supports different syntaxes such as RDF/XML, OWL2/XML and Manchester. Therefore the developers get ample options to adjust according to their tool stack. DPDL also cannot be left out as it stands out with ease of use and shorter learning curve as a second choice.

## 5 Conclusion

In this paper, existing knowledge representation techniques for software design patterns were reviewed with a focus on using them for model transformations. Introductions about software design patterns and model transformations were also included to provide a complete overview of the subject domain. In the discussion, the importance of design pattern specification techniques for pattern based model transformations was presented with example usage. Finally, the best design pattern specification techniques were recommended for the tasks of model transformation.

## References

1. Grand, M. (1998). Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML, Wiley.
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley.
3. AWS cloud design patterns. Available from: http://en.clouddesignpattern.

4. Cloud Design Patterns. Available from: https://docs.microsoft.com/en-us/azure/architecture/patterns/.
5. Alsheiksalem, O. & Qattous, H. (2017). An Expert System for Design Patterns Recognition. IJCSNS International Journal of Computer Science and Network Security, VOL.17 No.1, January 2017.
6. Pavlič, L., Podgorelec, V. & Hericko, M. (2014). A Question-Based Design Pattern Advisement Approach. Computer Science and Information Systems. 11. pp. 645-664. 10.2298/CSIS130824025P.
7. Palma, F., Farzin, H., Guéhéneuc, Y. & Moha, N. (2012). Recommendation system for design patterns in software development: An DPR overview. 2012 Third International Workshop on Recommendation Systems for Software Engineering (RSSE), Zurich, pp. 1-5.
8. Di Martino, B. & Esposito, A. (2015). A rule-based procedure for automatic recognition of design patterns in UML diagrams. Software Practice and Experience. 46. 10.1002/spe.2336.
9. Panich, A. & Vatanawood, W. (2016). Detection of design patterns from class diagram and sequence diagrams using ontology. 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), Okayama, pp. 1-6.
10. Fowler, M. Writing Software Patterns. Available from: https://www.martinfowler.com/articles/writingPatterns.html
11. Stahl, T., Voelter, M. & Czarnecki, K. (2006). Model-driven software development: technology, engineering, management. John Wiley & Sons.
12. Whittle, J., Hutchinson J. & Rouncefield, M., (2014). The State of Practice in Model-Driven Engineering, IEEE Software, vol. 31, no. 3, p. 79-85, May-June.
13. Sendall, S. & Kozaczynski, W. (2003). Model transformation: the heart and soul of model driven software development. IEEE Softw. 20(5), p. 42–45.
14. Web Ontology Language (OWL), W3C Recommendation, Available From: http://www.w3.org/TR/OWL/.
15. Resource Description Framework (RDF), W3C Recommendation, Available From: http://www.w3c.org/RDF/.
16. Dietrich, J. & Elgar, C. (2005). A formal description of design patterns using OWL, in: Proceedings of the ASWEC 2005, IEEE Computer Society.
17. Dietrich, J. & Elgar, C. (2007). Towards a Web of Patterns. Journal of Web Semantics, pp. 108-116, June 2007.
18. Mark, B., Jerry, H., Ora, L., Drew, M., Sheila, M., Srini, N., Massimo, P., Bijan, P., Terry, P., Evren, S., Naveen, S. & Katia, S. (2004) OWL-s: Semantic markup for web services. Available from: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122. World Wide Web Consortium.
19. Vassiliadis, V. Thea A web ontology language-owl library for [swi] prolog. Available from: http://www.semanticweb.gr/thea/.

20. Shakya, B. & Nantajeewarawat, E. (2013). A design pattern knowledge base and its application to sequence diagram design. 2013 International Computer Science and Engineering Conference (ICSEC), Nakorn Pathom. pp. 179-184.
21. Raje, R.R. & Chinnasamy, S. (2001). eLePUS - a language for specification of software design patterns. In SAC '01: Proceedings of the 2001 ACM symposium on applied computing. ACM: Las Vegas, Nevada, United States. pp. 600–604.
22. Mikkonen, T. (1998) Formalizing Design Patterns. In Proceedings of the 20th international conference on Software engineering (ICSE '98). IEEE Computer Society, USA, pp. 115–124.
23. Taibi, T. & Ngo D.C. (2003). Formal specification of design pattern combination using BPSL. Information and Software Technology 2003; 45(3). pp. 157–170.
24. Jeon, S., Lee, J. & Bae, D. (2002). An automated refactoring approach to design pattern-based program transformations in Java programs. Proceedings of the Ninth Asia-Pacific Software Engineering Conference. pp. 337 - 345. 10.1109/APSEC.2002.1183003.
25. Khwaja, S. & Alshayeb, M. (2013). Towards design pattern definition language. Software: Practice and Experience, 43(7), pp. 747–757.
26. Mapelsden, D., Hosking, J. & Grundy, J. (2002). Design pattern modelling and instantiation using DPML. In CRPIT '02: Proceedings of the Fortieth International Conference on Tools Pacific. Australian Computer Society: Inc.: Sydney, Australia. pp. 3–11.
27. Kim, D. (2007). Role-Based Metamodeling Language for Specifying Design Patterns. In Design Pattern Formalization Techniques. IGI Global: Pennsylvania, USA. pp. 183–205.
28. OMG: Meta Object Facility Specification, version 2.5.1.
29. Judson, S.R., France, R. & Carver, D.L. (2003). Specifying model transformations at the metamodel level.