ISSN: 0974-5823 Vol. 7 No. 09 September, 2022

International Journal of Mechanical Engineering

Domain-specific Rule Language Editor Using Model-Driven Architecture in SOA

B. Mythily[1] and Dr. R. Bhavani[2,*]

[1]*Department of Computer Science and Engineering*
Suguna College of Engineering, Nehru nagar, Kalapatti Road, Coimbatore-14, Tamil Nadu, India.
Email: mythilyabhi@gmail.com
[2,*]*Department of Computer Science and Engineering*
Suguna College of Engineering, Nehru nagar, Kalapatti Road, Coimbatore-14, Tamil Nadu, India.
Email: bhavaniaucse@gmail.com

Abstract. Domain-specific Rules are a crucial component of domain-specific enterprise applications as they enable change configuration and domain constraint management. Businesses that construct their systems using the process-driven service-oriented architecture (SOA) paradigm realize business processes by orchestrating services to manage the process's business activities. In order to facilitate the development of SOA, this study suggests an integrated model-driven solution method for domain-specific rule generation. Additionally, one of our main motivations for using the Decisional Model Notation (DMN) and Service Oriented Architecture Rule Language (SOARL) standards in creating various services is to study the standard models at every defined level.

Keywords: Service Oriented Architecture (SOA), variability model, decision making, rule generation

1. Introduction

A Domain-Specific Language (DSL) is a programming/modeling language that introduces domain-specific notations to improve reliability, productivity, portability and and maintainability for domain/end users [1]. Domain-specific rule language (DSRL) governs a set of rules that make data more process-free in a certain domain [2]. The core component of domain-specific enterprise applications, Domain-specific Rules (DSRs) let configuration modifications and manage domain constraints within the domain [3]. Our main focus is on the use of a conceptual domain model for rule generation in SOA, specifically for generating a DSRL [4] grammar, syntax, and domain constraint management.

The Rule is an expanded kind of code since it can always be customized unlike code, which needs to be compiled and built. We use the lower level execution code as the output and the higher level design model as the input in a process called rule generation. A process modeling language offers semantics and syntax to precisely define and express the service composition and business process needs. For the development and modeling of business processes, rule-based languages and a number of graphs have evolved that depend on formal training. For the creation of business rules, Diouf et al. [5,6] provide a method that combines UML models and domain ontologies. DecisionKing was created as an Eclipse7 platform plugin. Eclipse is an open source, Java-based programming environment. It is a plugin framework that offers a number of services [7] on top of which all plugin extensions are built as a foundation for various utilities.

Enterprise information systems are now frequently implemented in businesses using SOA concept. Different functions are designed to be encapsulated as services in SOA-based systems. A process engine that coordinates these services to carry out the various tasks that make up a business process is also introduced by a process-driven SOA [8]. Software extension and evolution are made easier by SOA [9] when business requirements change. According to Fazziki et al. [10], a model-driven approach (MDA)can be utilized to fill the gap between business needs and service-oriented architecture. The method represents business processes in a language known to developers using the SOAML meta model and BPMN. For real-time decision making in SOA 2.0, MEdit4CEP is a model-driven approach [11]. Masood et al. [12] develop a semantic performance-oriented decision support system (SPODSS) for SOA.

2. Model Driven Architecture (MDA)

MDA concepts, which are associated with the transformation of models in domain-specific applications, were used to build our SOA approach. Platform Specific Models (PSM) are required for the prototyping of the application and for its development. It is necessary to have tool support to automate the development process. To enable the use of an end-to-end automated procedure while also supporting user feedback, the tool should have default value options. Greater flexibility and result customisation are provided as a result. The PSM gives a general overview of SOA's constituent parts and the options available to organizations for SOA implementation.

These models are acquired through PIM transformation, which also adds relevant technical data to platforms. These models offer the framework needed to make rule generation easier. The MDA method is frequently employed for sophisticated and intricate model generators. The MDA four-level model structure is used to design the DSRL generator's architecture. The S3 is the Syntax Definition Formalism (SDF), which is the SDF's grammar, at the highest level. This level, which is defined and adheres to itself, is also known as a Computational Independent Model (CIM). The mapping rules between the CIM level (use case, DMN), which can be used to align the SOA's decision perspective.

At the S2 level, also known as the Platform Independent Model (PIM) level, we create the DSRL metamodel, which is the grammar of DSRL with ECA (Event Condition Action) described in SDF. At level S3, the metamodel complies with it. We define DSRL models of configuration application at the S1 level. The Platform Specific Model (PSM), which consists of entities and definitions, is what this is called. In the SOA's decision-making process, the model complies with the metamodel at level S2. We designate the setup of BPM customisation at the S0 level, which consists of DSR and XML rules that reflect the models at the S1 level.

3. Rule Language Editor

The user is guided by a specialized rule editor while designing decision-to-decision dependencies in SOARL. One of the most crucial features that is essential for user acceptability is code completion. Additionally, the syntax highlighting tool in the rule language editor improves the readability of the script in SOARL decisions. Long-term, this feature benefits the programmers who must keep the variability models up to date. Additionally, we offer on-the-fly syntax checking services that alert the user to typos and other errors that can be avoided. features including (1) error viewer, (2) error markers and messages, (3) comments, and (4) syntax errors.

To make the process of modeling in SOARL easier, DecisionKing offers an expression editor (with syntax highlighting and auto completion). To build more sophisticated expressions and query the value of decisions, in addition to the conventional operators, we offer the following actions (grammar shown in Algorithm 1).

```
RuleLangCompiler = { Rule }.

Rule   = ( "if" Expression "then" {Action } "endif") | Action
Action   = [ ActionFunctionName "(" Parameters ")" ] "
Parameters = [Expression { ", Expression ) ]
Function   "contains" | "isTaken" | "max" | "abs"
ActionFunctionName " setValue" | "reset" | "selectOption" |
"deSelectOption" | "allow" | "disAllow"
Expression   = AndExpr {" ‖|* AndExpr }
AndExpr   = EqlExpr { " &" EqlExpr }
EqlExpr   = RelExpr{(" ==*| "! =  *) RelExpr}
RelExpr   = AddExpr {(" < "|" > n|" <=| " >= " ) AndExpr }
AddExpr   = MulExpr {(+*|  * −) MulExpr }
MulExpr   = Unary{(" ∗ "|"/"|"%") Unary }
Unary   = {"+*|" − "| + ! "} Primary
Primary = (Literal | ident | Function "("Parameters ")
Literal   numberLiteral | stringliteral | true | false | null
 (** END of Grammar **)
```

Algorithm 1: DecisionKing's rule language syntax is used to convey decision dependencies [13]

1. selectoption(a, b) – In an enumeration decision, the deselect option (a, b) is used to select or deselect the alternative b.
2. setvalue (a, b) is an assignment function, this assigns the decision a the value b. Because the version of JBoss Rule Engine we used (version 4.0) only enabled function calls in the actions defined as a part of the rules, setValue was used instead of the standard assignment operator " $=$ ".
3. allow (a, b), disallow (a, b) are employed in enumeration decisions a to broaden or narrow the set of possible values for the set.
4. contains (a, b) is a set operator that can be used to perform $\subset, \subseteq, \in$ operations in enumeration decisions. The set to be compared with the one selected a's value is set b.
5. reset (a) is employed to reverse the decision made. Reversing a decision also resets all of its rule-modelled implications.
6. isTaken (a) is employed to determine whether the user has already made a decision.

The exact decision logic that was applied to arrive at the decision is described at the decision logic level. The decision logic can be represented in a number of ways, such as through an algorithm, an analytical model, or a decision table. Decision analysis involves these three steps:

1. Identify the Decisions : According to SOARL, decisions can be identified using terms like "forecast," "calculate," "selection," "determine," "choose," "validate," "assess," and, most importantly, "decide" based on the BPMN and the use case. These supporting procedures or duties are regarded as candidate decision-making services.
2. Analyze the Decision Conditions : Create a Decision Requirements Graph by specifying the knowledge and data that are necessary for making decisions. Important decision-making components and their dependencies in SOARL are shown in a notation provided by the DMN decision requirements. Decisions, input data, and sources of business expertise are a few of the crucial components.
3. Specify Decision Logic Level : The DMN decision logic level (DLL) indicates when a complete logic expression should be specified, which may be sufficient for automation. There may be a business model, value expression, or knowledge model for each decision in the diagram for the DMN decision requirements depending on the specifics of SOARL. Decision table specifications, business rules, algorithms, or analytical models might be included in a business knowledge model.

4. SOA Fixed Validation of generated rule

First, this method examines whether the characteristics that the end user selected are converted into rules in SOARL or not. We use the under and over-generation of SOARL rules to validate this issue. Second, we can claim that the decision of the feature model can be easily confirmed in a SOA method since the grammar of the rule is based on operational and functional of the models via validation.

- Under generation is defined as the absence of an instance in SOARL (event, action, etc.) either during or after the generation process.

- Over generation—In the SOARL methodology, this would be recognized as an additional portion of syntactic and semantic data.

We can therefore validate the resulting rule with some additional or less information discovered during the model transformation using this SOARL approach. This results in a flexible and simple-to-reconfigure environment for complicated systems, combined with the reuse of current opaque behavior(s).

The MDA idea can offer a multi-conceptual framework that enables a domain user to develop application models, business logic concepts, and rule generation for a target model or platform via transformations. The domain expert would be able to address and concentrate on domain engineering and domain-related problems that are unique to the application domain by employing the MDA approach. Model-based design and automated rule generation are novel concepts, thus the proposed solution cannot be evaluated using a common system. As a result, SOARL is possible to specify how each separate model that will be taken into account during the process is produced. We verify the output type of the generation in terms of accuracy, comprehensiveness, effectiveness, and efficiency.

Conclusion

In this research, we suggested a syntax definition for domain model language for rule generation leveraging domain variability to the SOA development process that can be used to achieve a model-driven and integrated solution for service-oriented architectures. By transforming conceptual models into general domain-specific rule languages that are transferable to other domains, we investigate future research that focuses on how to define the DSRL concerns abstract and concrete syntactical description with grammar construction across many domains.

References

[1] Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. ACM computing surveys (CSUR), 37(4), 316-344.

[2] Mani, N., Helfert, M., & Pahl, C. (2016). Business process model customisation using domain-driven controlled variability management and rule generation. International Journal on Advances in Software, 9(3&4), 179-190.

[3] Mani, N., Helfert, M., Pahl, C., Nimmagadda, S. L., & Vasant, P. (2018). Domain Model Definition for Domain-Specific Rule Generation Using Variability Model. Modeling, Simulation, and Optimization, 39-55.

[4] Mani, N., & Pahl, C. (2015). Controlled variability management for business process model constraints. ICSEA 2015, The Tenth International Conference on Software Engineering Advances. IARIA XPS Press.

[5] Diouf, M., Maabout, S., & Musumbu, K. (2007). Merging model driven architecture and semantic web for business rules generation. International Conference on Web Reasoning and Rule Systems. Springer.

[6] Musumbu, K., Diouf, M., & Maabout, S. (2010, July). Business rules generation methods by merging model driven architecture and web semantics. In 2010 IEEE International Conference on Software Engineering and Service Sciences (pp. 33-36). IEEE.

[7] Fayad, E. M., Schmidt, C. D., & Johnson, R. E. (1999). Building application frame works: object-oriented foundations of framework design. New York, NY, USA: John Wiley & Sons, Inc.

[8] Zdun, U., Hentrich, C., & Dustdar, S. (2007). Modeling process-driven and service-oriented architectures using patterns and pattern primitives. ACM Transactions on the Web (TWEB), 1(3), 14-es.

[9] Erl, T. (2005). Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall.

[10] El Fazziki, A., Lakhrissi, H., Yetognon, K., & Sadgal, M. (2012, November). A service oriented information system: a model driven approach. In 2012 Eighth International Conference on Signal Image Technology and Internet Based Systems (pp. 466-473). IEEE.

[11] Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. (2015). MEdit4CEP: A model-driven solution for real-time decision making in SOA 2.0. Knowledge-Based Systems, 89, 97-112.

[12] Masood, T., Cherifi, C. B., & Moalla, N. (2021). A machine learning approach for performance-oriented decision support in service-oriented architectures. Journal of Intelligent Information Systems, 56, 255-277.

[13] Wallner, Stefan. (2008). Integration of a Rule Language in a Tool Suite for Software Product Line Engineering. Linz: Master's Thesis, Johannes Kepler University.