Domain Model Definition for Domain-Specific Rule Generation Using Variability Model

Neel Mani, Markus Helfert, Claus Pahl, Shastri L Nimmagadda and Pandian Vasant

Abstract The business environment is rapidly undergoing changes, and they need a prompt adaptation to the enterprise business systems. The process models have abstract behaviors that can apply to diverse conditions. For allowing to reuse a single process model, the configuration and customisation features can support the design improvisation. However, most of the process models are rigid and hard coded. The current proposal for automatic code generation is not devised to cope with rapid integration of the changes in business coordination. Domain-specific Rules (DSRs) constitute to be the key element for domain specific enterprise application, allowing changes in configuration and managing the domain constraint with-in the domain. In this paper, the key contribution is conceptualisation of the do-main model, domain model language definition and specification of domain model syntax as a source visual modelling language to translate into domain specific code. It is an input or source for generating the target language which is do-main-specific rule language (DSRL). It can be applied to adapt to a process constraint configuration to fulfil the domain-specific needs.

N. Mani (✉) · M. Helfert
School of Computing, ADAPT Centre for Digital Content Technology,
Dublin City University, Dublin, Ireland
e-mail: neel.mani@computing.dcu.ie

M. Helfert
e-mail: markus.helfert@computing.dcu.ie

C. Pahl
Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
e-mail: Claus.Pahl@unibz.it

S. L. Nimmagadda
School of Information Systems, Curtin Business School (CBS), Perth, WA, Australia
e-mail: shastri.nimmagadda@curtin.edu.au

P. Vasant
Department of Fundamental and Applied Sciences, Universiti Teknologi PETRONAS,
Perak, Darul Ridzuan, Malaysia
e-mail: pvasant@gmail.com

I. Zelinka et al. (eds.), *Innovative Computing, Optimization and Its Applications*, Studies in Computational Intelligence 741,
https://doi.org/10.1007/978-3-319-66984-7_3

1 Introduction

We are primarily concerned with the utilisation of a conceptual domain model for rule generation, specifically to define a domain-specific rule language (DSRL) [1, 2] syntax, its grammar for business process model and domain constraints management. We present a conceptual approach for outlining a DSRL for process constraints [3]. The domain-specific content model (DSCM) definition needs to consider two challenges. The first relates to the knowledge transfer from domain concept to conceptual model, where model inaccuracies and defects may have been translated because of misunderstandings, model errors, human errors or inherent semantic mismatches (e.g., between classes). The other problem relates to inconsistency, redundancy and incorrectness resulting from multiple views and abstractions. A domain-specific approach provides a dedicated solution for a defined set of problems. To address the problem, we follow a domain model language approach for developing a DSRL and expressing abstract syntax and its grammar in BNF [4] grammar. A domain-specific language (DSLs) [5–7] refers to an approach for solving insufficient models by capturing the domain knowledge in a domain-specific environment. In the case of semantic mismatches/defects, a systematic DSL development approach provides the domain expert or an analyst with a problem domain at a higher level of abstraction.

DSL is a promising solution for raising the level of abstraction that is easier to understand or directly represent and analyse, thus, attenuating the technical skills required to develop and implement domain concepts into complex system development. Furthermore, DSLs are either textual or graphical language targeted to specific problem domains by increasing the level of automation, e.g. through rule and code generation or directly model interpretation (transform or translate), as a bridge, filling the significant gap between modeling and implementation. An increase in effectiveness (to improve the quality) and efficiency of system process is aimed at rather than general-purpose languages that associated with software problems. Behavioral inconsistencies of properties can be checked by formal defect detection methods and dedicated tools. However, formal methods may face complexity, semantic correspondence, and traceability problems. Several actions are needed for implementation of any software system. These are from a high-level design to low level execution. The enterprises typically have a high level of legacy model with various designs in a domain or process model. Automatic code generation [8–11] is a well-known approach for getting the execution code of a system from a given abstract model. The Rule is an extended version of code since code requires compiling and building, but the rule is always configurable. Rule generation is an approach by which we transform the higher-level design model as input and the lower level of execution code as output. It manages the above mentioned constraint.

We structure the modelling and DSL principles in Sect. 1. In Sect. 2, we discuss the State of-the-Art and Related Work. We give an overview of the global intelligent content processing in a feature-oriented DSL perspective, which offers the domain model and language definition in Sect. 3. Then, we describe the ontology-based con-

ceptual domain model in Sect. 4. The description of the domain model and language expressed in terms of abstract and concrete syntax are given in Sect. 5. As a part of DSRL, the general design and language are presented in Sect. 6. Section 7 provides details regarding implementation of a principal architecture of DSRL generation and how it translates from the domain model to the DSRL. We discuss analysis and evaluation of the rule in Sect. 8. Finally, we conclude our work with future scope. Throughout the investigation, we consider the concrete implementation as a software tool. However, a full integration of all model aspects is not aimed at, and the implementation discussion is meant to be symbolic. The objective is to outline the principles of a systematic approach towards a domain model used, as source model and the domain specific rule language, as a target for content processes.

2 Related Work

The web application development is described as a combination of processes, techniques and from which web engineering professionals make a suitable model. The web engineering is used for some automatic web application methodologies such as UWE [12], WebML [13] and Web-DSL [14] approaches. The design and development of web applications provide mainly conceptual models [15], focusing on content, navigation and presentation models as the most relevant researchers expressed in [16, 17]. Now, the model driven approach for dynamic web application, based on MVC and server is described by Distante et al. [18]. However, these methods do not consider the user requirement on the variability model. To simplify our description, we have considered the user requirement and according to the need of the user, the user can select the feature and customize the enterprise application at the dynamic environment.

A process modeling language provides syntax and semantics to precisely define and specify business process requirements and service composition. Several graph and rule-based languages have been emerged for business process modeling and development, which rely on formal backgrounds. They are Business Process Modeling Notation (BPMN) [19], Business Process Execution Language (BPEL)/WS-BPEL, UML Activity Diagram Extensions [20], Event-Driven Process Chains (EPC) [21], Yet Another Work ow Language (YAWL) [22], WebSphere FlowMark Definition Language (FDL) [23], XML Process Definition Language (XPDL) [24], Java BPM Process Definition Language (jPDL) [25], and Integration Definition for Function Modeling (IDEF3) [26]. These languages focus on a different level of abstraction ranging from business to technical levels and have their weaknesses and strengths for business process modeling and execution. Mili et al. [27] survey the major business process modeling languages and provide a brief comparison of the languages, as well as guidelines to select such a language. In [28], Recker et al. present an overview of different business-process modeling techniques. Among the existing languages, BPMN and BPEL are widely accepted as de facto standards for business process design and execution respectively.

Currently, there is no such type of methodology or process of development for creating a rule-based system in a web application (semantic-based). Diouf et al. [29, 30] propose a process which merges UML models and domain ontologies for business rule generation. The solution used for semantic web has ontologies and UML; to apply to the MDA approach for generating or extracting the rules from high level of models. Although, the proposed combination of UML and semantic based ontologies is for extracting the set of rules in target rule engine, they only generate the first level of the abstraction of the rules.

Our approach provides the systematic domain-specific rule generation using variability model. The case study uses intelligent content processing. Intelligent content is digital that provides a platform for users to create, curate and consume the content in dynamic manner to satisfy individual requirements. The content is stored, exchanged and processed by a dynamic service architecture and data are exchanged, annotated with metadata via web resources.

3 Business Process Models and Constraints

We use the intelligent content (IC) [31] processing as a case study in our application. The global intelligent content (GIC) refers to digital content that allows users to create, curate and consume content in a way that fulfills dynamic and individual requirements relating to information discovery, context, task design, and language. The content is processed, stored and exchanged by a web architecture and the data are revised, annotated with metadata through web resources. The content is delivered from creators to consumers. The content follows a certain path that consists of different stages such as extraction and segmentation, named entity recognition, machine translation, quality estimation and post-editing. Each stage, in the process, comes with its challenges and complexities.

The target of the rule language (DSRL) is an extendable process model notation for content processing. Rules are applied at processing stages in the process mode.

The process model that describes activities remains at the core. It consists of many activities and sub-activities of reference for the system and corresponds to the properties for describing the possible activities of the process. The set of activities constitutes a process referred to as the extension of the process and individual activities in the extension are referred as instances. The constraints may be applied at states of the process to determine its continuing behaviour depending on the current situation. The rules combine a condition (constraint) on a resulting action. The target of our rule language (DSRL) is a standard business process notation (as shown in Fig. 1).

The current example is a part of digital content (processing) process model as shown in Fig. 1, a sample process for the rule composition of business processes

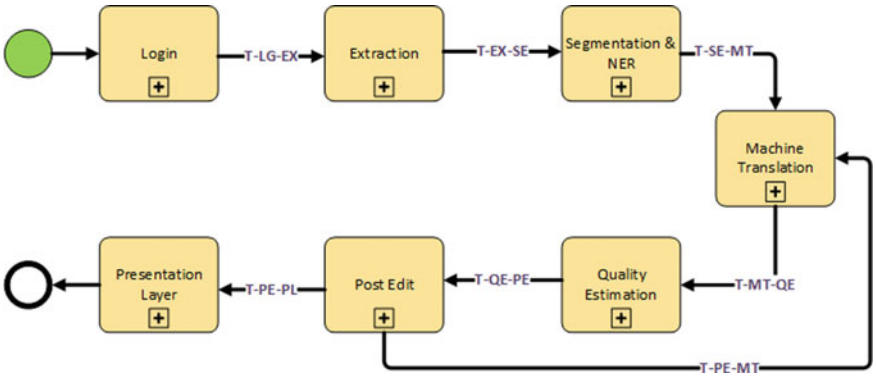

Fig. 1 Process model of global intelligent content

and domain constraints that conduct this process. The machine language activity translates the source text into the target language. The translated text quality decides whether further post-editing activity is required. Usually, these constraints are domain-specific, e.g., referring to domain objects, their properties and respective.

4 Ontology-Based Conceptual Domain Model

We outline the basics of conceptual domain modelling, the DSRL context and its application in the intelligent content context. The domain conceptual models (DCM) are in the analysis phase of application development, supporting improved understanding and interacting with specific domains. They support capturing the requirements of the problem domain and, in ontology engineering. A DCM is a basis for the formalized ontology. There are several tools, terminologies, techniques and methodologies used for conceptual modelling, but DCMs help better understanding, representing and communicating a problem situation in specific domains. We utilise the conceptual domain model to derive at a domain-specific rule language.

A conceptual model can define concepts concerning a domain model for a DSL, as shown in the class model in Fig. 2, which is its extended version described in [3]. A modeling language is UML-based language defined for a particular domain, defining relevant concepts as well as a relation (intra or inter-model) with a metamodel. The metamodel consists of the concrete syntax, abstract syntax and static semantics of the DSL. The abstract syntax defines modelling element such as classes, nodes, association, aggregation and generalisation, and relationships between the modelling elements [3].

A DSRL reuses domain model elements or define new modelling element depending on the domain concept and its relations with other elements. Modelling of elements is done with two fundamental types: concepts and relations. The concepts are

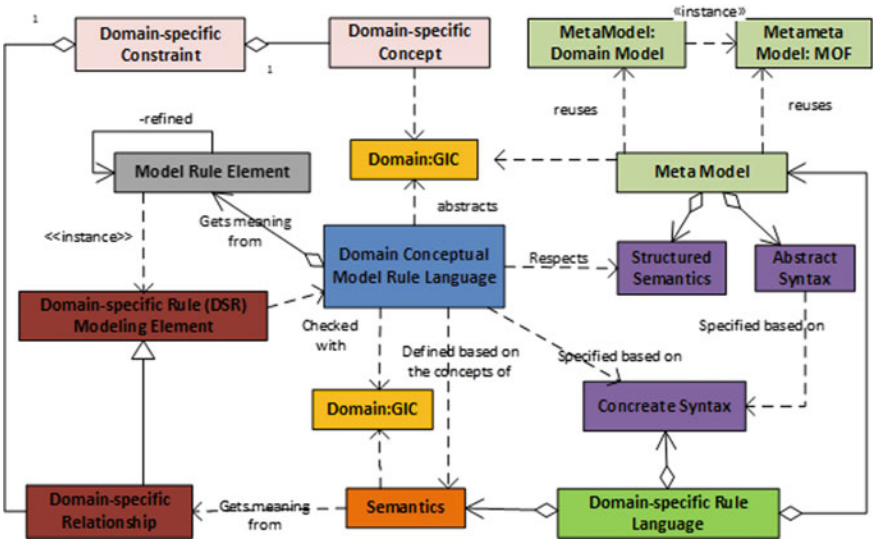

Fig. 2 A domain model based DSRL concept formalization

based on domain concepts such as entity, state, action, location, risk, menu and relations are used to connect elements. The relations are also of two types: generalisation and association. Aggregation is a special type of an association. The association has domain-specific relations like conditional flow, multiplicity and aggregation. Furthermore, relations may have properties such as symmetry, reflexivity, equivalence, transitivity and partial order.

5 Language Definition of a Domain Model

Domain model serves as the very basis of all types of Business Applications that run on a Domain, both individual as well as Enterprise applications. The objective is to define the language for Domain Model and recognises the internal data structures or schema used in it. It is easy to transform or translate the graphical Domain Model into Textual Rule language for a particular domain. In this scenario, the objective data structure refers to the storability of a domain model in a vulnerable environment, as in the case with a rule language. This is because the target of a language for mapping the translated domain model's knowledge into XML schema of DSRL that follows the rule paradigm.

1	*Domain*	*::=*	*<Domain model>*	*Domain definition*
2	*Concept*	*::=*	*<Concept>*	*Concept definition*
3	*Class*	*::=*	*<Attributes>,< Operation>, < Receptions>, <Template Parameters>, < Component>, <Constraints>, <Tagged Values>*	*Class Definition*
4	*<Relations>*	*::=*	*<Association>|<DirectedAssociation>|<Reflexive Association>|<Multiplicity>|<Aggregation>|<Composition>|<Inheritance/Generalization>|<Realization>*	*Class relationships*
5	*<Association>*	*::=*	*'→' | '✱' | '::'*	*Structural relationship between objects (classes) of different type*
6	*<Type>*	*::=*	*<BuiltinType>|<UCase Ident >|<EnumType>*	*Domain model type concept type or extended type enumeation type list type*
7	*<PrimitiveTypes >*		*<String>,<Integer>,<Boolean>, ...,<Date>*	*Domain model primitive (built-in) types*

Fig. 3 Syntax definition of domain model language

5.1 Language Description

Metamodeling is used to accomplish specifications for the abstract syntax. We introduce the Domain Model language by analyzing its syntax definition (Fig. 3 shows in EBNF notation). The language with its basic notions and their relations are defined with structural constraints (for instance to express containment relations, or type correctness for associations), multiplicities, precise mathematical definition and implicit relationships (such as inheritance, refinement). The visual appearance of the domain specific language is accomplished by syntax specifications, which is done by assigning visual symbols to those language elements that are to be represented on diagrams.

5.2 Syntax

For describing the language in general, the rule language checks various kinds of activities. The primary requirement is to specify the concept of the syntax (i.e. abstract and or concrete syntax) and develop its grammar. The semantics is designed to define the meaning of the language. The activities are completed by concepting, designing and developing a systematic domain-specific rule language systems, defining the functions and its parameters, priorities or precedence of operators and its values, naming internal and external convention system. The syntaxes are expressed with certain rules, conforming to BNF or EBNF grammars that can be processed by rules or process engine to transform or generate the set of rules as an output. The generated rules follow the abstract syntax and gram-mar to describe the domain

concepts and domain models because both the artefacts (abstract syntax and grammar) are reflected in the concrete syntax.

5.3 Abstract Syntax

The abstract syntax refers to a data structure that contains only the core values set in a rule language, with semantically relevant data contained therein. It excludes all the notation details like keywords, symbols, sizes, white space or positions, comments and color attributes of graphical notations. The abstract syntax may be considered as more structurally defined by the grammar and Meta model, representing the structure of the domain. The BNF may be regarded as the standard form for expressing the grammar of rule language, and some type describes how to recognize the physical set of rules. Analysis and downstream processing of rule language are the main usages of Abstract syntax. Users interact with a stream of characters, and a parser compiles the abstract syntax by using a grammar and mapping rules.

For example, we do process activities in our case domain (Global Digital Content): Extraction and Machine Translation (MT). The list of the process model, event and condition are following:

List of Process

```
<Process-ModelList>::=<gic:Extraction>|
<gic:MachineTranslation>
```

List of Event

```
<EventList> ::={
gic:Text-->SourceTextInput,
gic:Text-->SourceTextEnd,
gic:Text -->SourceTextSegmentation,
gic:Text-->SourceParsing,
gic:Text-->MTSourceStart,
gic:Text-->MTTargetEnd,
gic:Text-->TargetTextQARating,
gic:Text-->TargetTextPostEditing,
}
```

List of Conditions

```
<ConditionList>::=<gic:Extraction.Condition>|
            <gic:MachineTranslation.Condition>
<gic:Extraction.Condition>::=IF(<gic:Text.Length::=<L)|
    IF(<Source.Language::==Language_List>)
    IF(<Target.Language::==Language_List>)
```

```
    IF(<SingleLangugeDetection((gic:Text)::== True
                                          |False>)
    IF(<MultiLanguageText(gic:Text)::== True|False>)
<gic:MachineTranslation.Condition>::= IF (<gic:Translation
(Source.Lang, TargetLang,gic:Text)  ::= True|False>)|
    IF (<gic:Translation.Memory ::= <TM)(Mem Underflow)|
    IF (<gic:Translation.Memory ::= >TM)(Mem Overflow)|
    IF(<gic:Translation(gic:TxtSource,Source.Lang)>
gic:Translation(gic:TxtTarget,Target.Lang)>)
```

where L is length of text and TM is the specific memory size.

5.4 Concrete Syntax

Rule languages use textual concrete syntax, which implies that a stream of characters expresses the program syntax. The modelling languages traditionally have used graphical notations and primarily in modelling languages. Though textual domain-specific languages (and mostly failed graphic based general-purpose languages) have been in use for a long time only recently, the textual syntax has found a prominent use for domain-specific modelling. Textual, concrete syntax form have been traditionally used to store programs, and this character stream is transformed using scanners and parsers into an abstract syntax tree for further processing by the Programming languages. In the modelling languages, editors have found a major usage, as it directly manipulates the abstract syntax and uses projection to render the concrete syntax in the form of diagrams.

The concrete syntax of DSLs is expected to be textual by default. If good tool support is available, the textual support has been found to be adequate for comprehensive and complex software systems. The programmers write lesser code in DSL as compared to a GPL for expressing the same functionality—because the available abstractions are quite similar to the domain. An additional language module suitable for the domain is defined easily by the programmers.

6 Rule Language Definition

Now we go back to the full rule definition. The DSRL grammar [2] is defined as follows. We start with a generic skeleton and then map the globic domain model (gic).

```
<DSRL Rules> ::= <EventsList>
                 <RulesList>
```

```
                <ProcessModelList>
<EventLists> ::= <Event> | <Event> <EventLists>
<Event> ::= EVENT <EventName> IF <Expression> |
            EVENT <EventName> is INTERN or EXTERN
<RulesList> ::= <Rule> | <Rule> <RuleList>
<Rule> ::= ON<EventName>
           IF<Condition>DO<ActionList>
<ActionList> ::= <ActionName> |
                 <ActionName>,<ActionList>
<ProcessModelList> ::= <ProcessModel> |
                       <ProcessModel>,<ProcessModelList>
<ProcessModel> ::= ProcessModel <ProcessModelName>
               ::= <ProcessModelName>
                   [TRANSITION_(SEQUENCIAL(DISCARDDELAY))],
                   [TRANSITION_PARALLEL(DISCARD|DELAY)]
                   [INPUTS(<InputList>)]
                   [OUTPUTS(<OutputList>)]
```

TRANSITION_SEQUENCIAL and TRANSITION_PARALLEL denoted as transitions of a process model.

The description of DSRL contains lists of events, condition, an action of the rules and process model states. An event can be an internal or external (for rules generated as an action, it may be the INTERNAL or EXTERNAL term) or generated when the expression should have been satisfied by the condition or becomes true. An event name activates with ON syntax, which is a Boolean expression to determine the conditions that apply and the list of actions that should be per-formed when event and condition are matched or true (preceded by DO syntax). The process models contain the state name. A certain policy is decided in the process model when sequential, and parallel actions are performed or sent in that state. An action is an executable program or set of computation decussation. The action provides methods or function invocation, creating, modifying, updating, communicating or destroying an object. DISCARD allows discarding the instructions, and DELAY allows delaying the instructions, but one.

For example, the gic:Extraction is used in an event on Text Input by user as a source data.

```
EVENT IF TextInput_ON
EVENT gic:TextInput::BOOL IF TextInput_Get
EVENT gic:TextInput::BOOL IF TextInput_ON
ON presence
IF (gic:SourceLang:EN) DO
(
     ON presence
     IF (gic:TextLength <X) DO
gic:Translate(Text)
```

```
ELSE

 Notification to user (Text LENGTH LESS  THAN X)
)
ELSE
Notification to user(Source language is invalid)
```

7 Implementation of Principle Architecture of DSRL Generator

The principle architecture of a domain-specific rule (DSR) is the automated model to text generator on accessible domain models, extracting information from them, and translating it into output in a specific target syntax. This process follows the concept of Model-Driven Architecture which depends on the metamodel. The modeling language with its concepts, the source syntax, semantics and its rules are required by the domain-specific framework and target environment. We present the process architecture of a domain model translation and the target rule environment in Fig. 4.

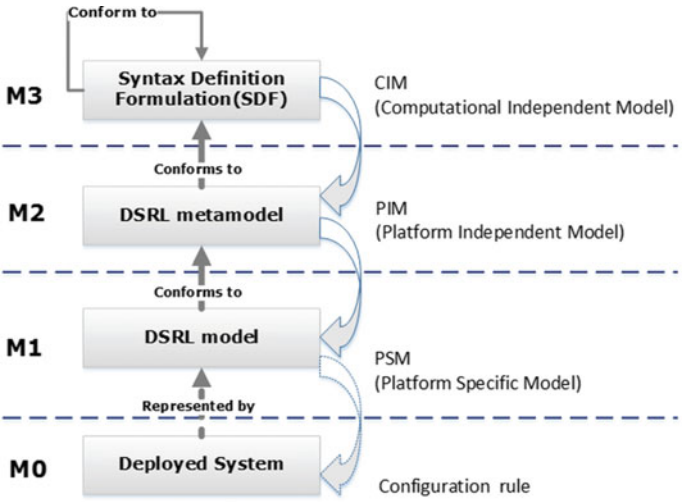

Fig. 4 MDA organisation view of models approach and artifacts of DSRL generator

7.1 Architecture

The architecture of the DSRL generator follows the MDA as four-level model organization, presented by Bzivin [32] as illustrated in Fig. 4. At the top level, the M3 is the Syntax Definition Formalism (SDF) metametamodel which is the grammar of the SDF. This level is also known as Computational Independent Model (CIM) or metametamodel as defined (and thus conforms to) itself [33]. A self-representation of the BNF notation takes some lines. This notation allows a defining infinity of well-formed grammars. A given grammar allows description of the infinity in syntactically correct DSR configuration.

At the M2 level, we describe the DSRL metamodel, i.e., the grammar of DSRL with ECA as defined in SDF and this level is called Platform Independent Model (PIM). The metamodel conforms to the metametamodel at level M3.

At the M1 level, we describe DSRL models for configuration applications. It is known as Platform Specific Model (PSM) consisting of entity and definitions. The model conforms to the metamodel at level M2. The bottom level is called M0, we define the configuration of BPM customization consisting of DSR and XML rules, which represent the models at the M1 level.

7.2 Mappings Domain Model and Domain-Specific Rule Language

A mapping is description of mapping rule definitions, generation, configuration and execution of order specification. Each mapping rule specifies what target model fragment is created for the given DSR. The mapping rule body contains one or more class of the domain model occurrences (Sect. 4) with all attribute and operational value set. Expressions for attribute, functional and operational setting are based on a specific source metamodel, Fig. 5 shows the corresponding domain model of gic:extraction sub type of digital content process used as a source metamodel to describe the DSRL conceptualization as illustrated in Fig. 2 (Sect. 3). Although the given source metamodel is completely translated into graphical model to text rule by using the grammar or language definition of source and target metamodel as given in Sects. 4 and 5, it is sufficient to show all basic mapping constructs.

7.3 Domain Model Translation into DSR

A rule generation is made automatic such that domain models are accessed in a way to extract information and translate it into output in a specific syntax based on feature model, as selected by the domain user. This process model is guided by the metamodel, the modeling language with its high level of concepts, syntactical,

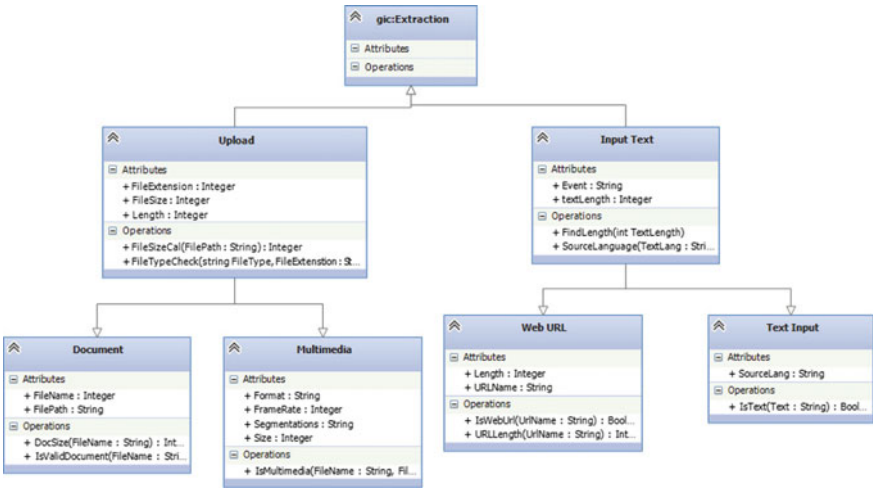

Fig. 5 Source metamodel of gic:Extraction as example DSRL used for mapping

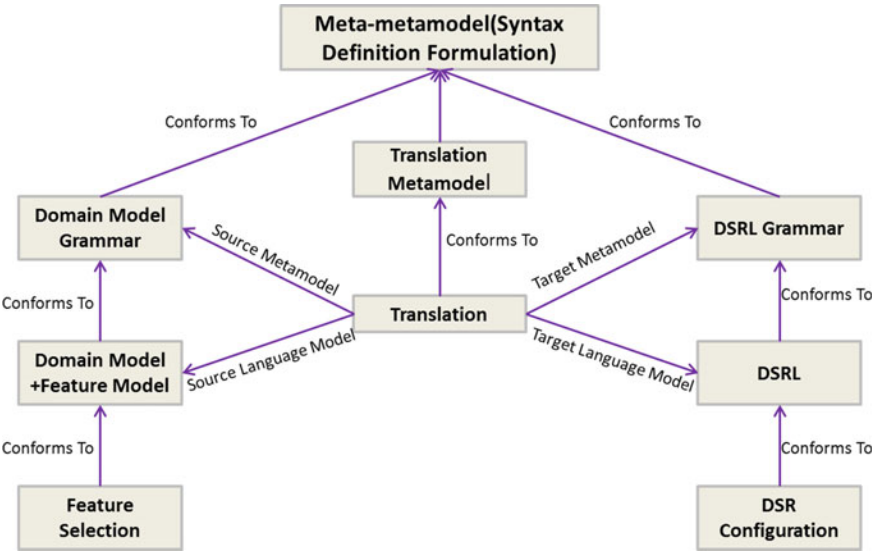

Fig. 6 Domain model to DSR translations

and semantics rules. The input required by the user (selection of the feature model) needs a domain to process the domain model, target environment as rule generation and configuration. We present the process of domain model translation and target rule environment in Fig. 6.

In an example of a rule generation from a domain model, we propose an approach: the domain model would be translated into a domain-specific rule language through a model transformation or translation, and then the DSRL meta-metamodel would be synthesized into DSR text by means of a rule generator. It is advantageous to have syntactical and semantic domain translation achieved by a graphical model to text model translation. It is a dedicated technology, because rule generators deal with the abstract and concrete syntaxes of the target language (in Sect. 3) directly. The entire process separates two distinct tasks (translation and synthesis) that are performed using appropriate tools.

For translation, the models need to be expressed in a modeling language (e.g., UML for design models, and programming languages for source models). A metamode expresses the modeling languages syntax and semantics by themselves. For example, the syntax of the Domain-metamodel has feature notations expressed using class diagrams, whereas its semantics is described by well-defined rules (expressed as OCL constraints) and a mixture of natural languages [34]. Based on the language in which the source and target models of a translations (grammar of model change) or transformation are expressed, a distinction is made between endogenous and exogenous transformations. The endogenous transformations are expressed between models in the same languages (when the grammar and structure are same).

The exogenous transformations are conversions made between models and expressed using different languages (the grammar and syntax are different), which is also known as translation. Essentially the same distinction was proposed in [35], but ported to a model transformation setting. We use the exogenous transformation that taxonomy and graphical model to text rule, whereas the term translation is used for an exogenous transformation.

8 Generated Rule Analysis and Evaluation

In rule generation evaluation, we validate the type of generated output concerning the correctness, completeness, output effectiveness and efficiency. Our primary goal is to have a proof of fully functional and operational correctness, and completeness of the rule for its feature requirement selected by the domain user. Our rule evaluation consists of following:

- Validation of rule generation concerning under- and over-generation.
  - Under generation—We define under generation as missing instance (for example events, actions etc.) at the time of generation or after generation.
  - Over generation—This is identified as an added information regarding syntax and semantics (functional and operational information).
- Evaluation of syntactical and semantical correctness of generated rule fulfills our goal. The above results imply that if one knows, for example, how to formulate partial correctness of a given deterministic algorithm in predicate mathematics,

the formulation of many other properties of the algorithm in predicate mathematics could have been straightforward. As a matter of fact, partial correctness has already been formulated in predicate mathematics and manual rule templates of many feature deterministic algorithms.

- Syntactical correctness means correct use of keys, functions, values, and grammatical rules.
- Semantical correctness is important for functional and operational point of view; here, we validate the correct order of generated rule and grammatical sequence.
- Grammatical correctness means the generated rules follow SDF grammar which is used in M3 level of MDA model as shown in Fig. 4.
- Comparing automated and hand-written rules.

- Evaluation of completeness of generated rule.
  - Completeness: Most rule languages are not designed as targets for rule generation, because of lack of functional and operational parameters in program fragments. Major challenging part in the rule sets has two constraints: dead-lock situation and live lock situation.
    Identification of rule deadlock.
    Identification of rule live lock.

It also needs to be identified if rules of the application require any additional set of rules to function as desired.

9 Discussion

In this paper, we have proposed a syntax definition for domain model language for rule generation and presented a DSR generation development for domain model through MDA approach using domain variability. We have presented a novel approach of handling knowledge transfer from domain concept (domain model) to conceptually configurable rule language, avoiding the inaccuracies and misunderstanding, model error, human error or semantic mismatches during translation of graphical abstract model to text rule. We have added adaptivity to the domain model. We provide a conceptual view of domain-specific rule generation and manage the domain model, and variability model using MDA. It helps in managing frequent changes of the business process along with variability schema of a set of structured variation mechanisms for the specification. The domain user can generate the DSRs and configure domain constraint in a dynamic environment. They can generate and configure DSRs without knowing any technical and programming skill. The novelty of our approach is a variability modelling usage as a systematic approach to transforming (generate) domain-specific rule from domain models.

We plan to extend this approach in combination with our existing work on business process model customization based on user requirement (feature model, domain

model and process models) so that a complete development life cycle for the customization and configuration of the business process model is supported. We explore further research that focuses on how to define the DSRL concerning abstract and concrete syntactical description with grammar formation across different domains, converting conceptual models into generic domain-specific rule language which are applicable in other domains. So far, it is a model for text translation but has the potential to serve as a system that learns from existing rules and domain models, driven by the feature model approach with automatic constraints configuration that is resultant to an automated DSRL generation.

Acknowledgements This research is supported by Science Foundation Ireland (SFI) as a part of the ADAPT Centre for Digital Content Technology at Dublin City University (Grant No: 13/RC/2106) and EAI COMPSE 2016, Penang, Malaysia.

References

1. 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.
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. Tanrıöver, Ö. Ö, & Bilgen, S. (2011). A framework for reviewing domain specific conceptual models. Computer Standards. *Interfaces*, *33*(5), 448–464.
4. Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Communications of the ACM*, *7*(12), 735–736.
5. Deursen, A. V., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, *35*(6), 26–36.
6. Fowler, M. (2010). Domain-specific languages. Pearson Education.
7. Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, *37*(4), 316–344.
8. Hudak, P. (1997). *Domain-Specific Languages. Handbook of Programming Languages*, *3*, 39–60.
9. Ringert, J. O., et al. (2015). Code generator composition for model-driven engineering of robotics component connector systems. arXiv:1505.00904.
10. Edwards, G., Brun, Y., & Medvidovic, N. (2012). Automated analysis and code generation for domain-specific models. *2012 Joint Working IEEE/IFIP Conference on, Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. IEEE.
11. Prout, A., et al. (2012). Code generation for a family of executable modelling notations. *Software Systems Modeling*, *11*(2), 251–272.
12. Koch, N., et al. (2008). UML-based web engineering. *Web Engineering: Modelling and Implementing Web Applications* (pp. 157–191). Springer.
13. Ceri, S., Fraternali, P., & Bongio, A. (2000). Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, *33*(1), 137–157.
14. Groenewegen, D. M., et al. (2008). WebDSL: a domain-specific language for dynamic web applications. *Companion to the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*. ACM.
15. Ceri, S., Fraternali, P., & Matera, M. (2002). Conceptual modeling of data-intensive Web applications. *IEEE Internet Computing*, *6*(4), 20–30.

16. Moreno, N., et al. (2008). Addressing new concerns in model-driven web engineering approaches. *International Conference on Web Information Systems Engineering*. Springer.
17. Linaje, M., Preciado, J. C., & Sánchez-Figueroa, F. (2007). Engineering rich internet application user interfaces over legacy web models. *IEEE Internet Computing*, *11*(6), 53–59.
18. Distante, D., et al. (2007). Model-driven development of web applications with UWA, MVC and JavaServer faces. *International Conference on Web Engineering*. Springer.
19. White, S. A. (2004). *Introduction to BPMN. IBM Cooperation*, *2*.
20. Dumas, M., & Ter Hofstede, A. H. (2001). UML activity diagrams as a workflow specification language. *ł UML 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools* (pp. 76–90). Springer.
21. Davis, R. (2001). Business process modelling with ARIS: a practical guide. Springer Science Business Media.
22. van der Aalst, W. M. P., & ter Hofstede, A. H. M. (2005). YAWL: yet another workflow language. *Information Systems*, *30*(4), 245–275.
23. IBM. (December 2010). WebSphere©MQ Workow FlowMareket©Definition Language (FDL).
24. Zeng, L., et al. (2004). Qos-aware middleware for web services composition. *IEEE Transactions on, Software Engineering*, *30*(5), 311–327.
25. Boss, J. (January 2008). jBPM Process Definition Language (jPDL).
26. Maker, R., et al. (1992). *IDEF3-Process Description Capture Method Report*. Information Integration for Concurrent Engineering (IICE), Armstrong Laboratory, Wright-Patterson AFB: OH.
27. Mili, H., et al. (2010). Business process modeling languages: Sorting through the alphabet soup. *ACM Computing Surveys (CSUR)*, *43*(1), 4.
28. Recker, J., et al. (2009). Business process modeling-a comparative analysis. *Journal of the Association for Information Systems*, *10*(4), 1.
29. 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.
30. Musumbu, K., Diouf, M., & Maabout, S. (2010). Business rules generation methods by merging model driven architecture and web semantics. *2010 IEEE International Conference on Software Engineering and Service Sciences*. 2010. IEEE.
31. Pahl, C., Mani, N., & Wang, M. -X. (2013). A domain-specific model for data quality constraints in service process adaptations. *Advances in Service-Oriented and Cloud Computing* (pp. 303-317). Springer.
32. Bzivin, J. (2005). On the unification power of models. *Software Systems Modeling*, *4*(2), 171–188.
33. Visser, E. (1997). Syntax definition for language prototyping. Eelco Visser.
34. Group, O. M., Unified Modeling Language specification version 1.5. formal. 2003.
35. Visser, E. (2001). A survey of rewriting strategies in program transformation systems. *Electronic Notes in Theoretical Computer Science*, *57*, 109–143.