Applying a Business Policy Meta-Model in a Domain Specific Modeling Language

Tod Sedbrook, Monfort College of Business, University of Northern Colorado, Greeley, CO USA
Tod.Sedbrook@unco.edu

Abstract

Developing a Domain Specific Modeling Language (DSML) for expressing business policies requires representing, organizing and maintaining the knowledge of business domain experts such as executives or business process managers. This study defines a DSML meta-model to express complex semantic patterns of business policies and demonstrates a way to transform resulting policy models into executable code specifications. We validate the design by exploring business applications within the context of the Resource, Event, Agent - Enterprise Ontology (REA-EO). The application of the DSML policy framework is illustrated through a succession of models to capture, validate and apply business polices.

Introduction

Model Driven Engineering (MDE) and Domain Specific Modeling Languages (DSML) promises a ten-fold productivity increase by assisting requirements analysis and supporting code synthesis (Umble, Haft et al. 2003; Sprinkle, Mernik et al. 2009). The vision of MDE is to create a transparent path from high level requirements to executable code. Business modeling is a key MDE process that drives the definition of business policies where designers and testers assure policies are captured and verified within subsequent design and testing models.
Business managers collaborate and modelers progressively annotate designs assisted through DSML meta-modeling tools to engineer policies for business systems. Resulting DSML models can then detect inconsistencies, document exceptions and verify the resulting software implementations (Sprinkle, Mernik et al. 2009; Frantz 2011).

Work is underway to define a domain specific language (DSL) that conforms to the REA–EO operational level ontology. But, there is a need to extend DSLs beyond the operational level to the REA-EO policy level (Sonnenberg, Huemer et al. 2011). In particular, there is a need to explore capabilities for representing REA-EO policy extensions within a modeling framework that helps mangers as they capture, apply and evolve business policies.

This study defines a DSML meta-model that generalizes the linguistic definitions of knowledge-level extensions as specified in the REA-EO (Geerts and McCarthy 2006). The DSML meta-model is applied to capture domain specific business policies. The resulting high-level domain policies are then automatically translated into executable code. The study develops a high-level DSML meta-model, applies the resulting language to specify domain specific models of business policies and demonstrates how transforming model representations produce executable code for use in policy validation frameworks. We apply a design science methodology to build and evaluate a DMSL (Hevner, March et al. 2004) and investigate its construction and validation within a case study.

Defining a Meta-Model for a Policy Language

Creating a DSML meta-model is an iterative process that evolves through repeated stages of analysis, design and implementation (Mernik, Heering et al. 2005; Cho 2011). We first investigated general characteristics of existing policy languages across a wide variety of domains. In addition, we reviewed philosophical theories of reasoning and knowledge representation and identified four ontological relationships necessary for a policy language:

1) **Intrinsic and Mutual Properties** – relates elements to values
2) **Classification Relations-** relates two or more properties of elements to form higher level types
3) **Exemplar Relations** – relates types to a prototype or specification
4) **Aggregations** – relates collections of exemplars

We tested these ontological relationships by evaluating their correspondence to the policy terminology and representational patterns from five existing concrete policy languages across the domains of the REA-EO (knowledge-level), business rule vocabularies, privacy, security and UML/OCL modeling.

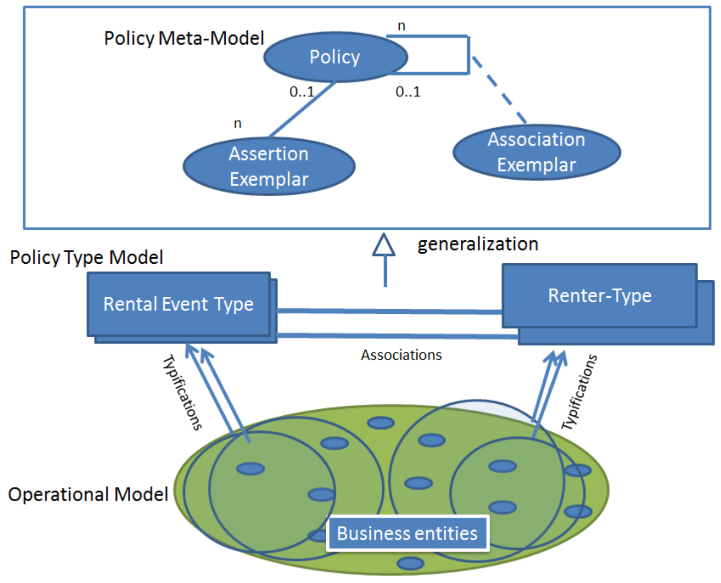
The four ontological relationships were mapped to a meta-model. Figure One presents the meta-model and the following defines meta-model elements of *Policy*, *Assertion Exemplar* and *Association Exemplar*. A *Policy* is an aggregation of *Assertion Exemplars* and *Policies* are coupled through the A*ssociation Exemplar* association class. An *Assertion Exemplar* represents a set of intrinsic properties derived from a policy-level type where each property is mapped to a desired or intended value. For example, the "Renter type" defines the intrinsic property "Age" and the *Assertion Exemplar* maps "Age" to the value "over 25 years old". An A*ssociation Exemplar* represents a set of the desired or intended value of a mutual property meaningful only in the context of two or more *Policies*. That is, the value assigned to an *Association Exemplar's* property represents a desired relationship between *Policies*. For example, "Rental" and "Pick Up" *Policies* are related by the mutual property "Valid Date" with the desired value of "true".

Code Transformations for Policy Validation

The definition of the policy meta-model provides a language to specify formalized policy models. The next step is to transform the explicit policy models into a programming language that can be plugged into standard validation frameworks. To investigate the policy meta-model, prototypes were developed to extend Microsoft Visual Studio's DSL tools. Visual Studio (2010) supports defining a policy meta-language to produce a stand-alone graphical designer suitable for creating concrete policy models. The resulting stand-alone graphical designer (the output of applying the meta-model) provides business experts with the ability to manipulate graphical shapes to define specific configurations of concrete policies. Concrete policies are then translated into F# code which is plugged into the .Net validation framework.

We choose the F# functional language as .Net language for translating policies models for the following reasons. First, the translation of domain policies to F# retains the policy's semantic meaning by supporting English-like representations which can be independently examined. Second, the interoperable F# representations provide pluggable extensions to augment Visual Studio's existing validation APIs. Third, F# shares common attributes with lisp, Haskell, OCaml, ML and other functional programming languages that have demonstrated a long history of creating concise, readable and logically verifiable code. (Cabot and Teniente 2007; Syme, Granicz et al. 2010).

Figure One. A Policy Meta-Model generalizes the REA-EO Policy framework to define a language for creating Policies.



Model Driven Policy Management

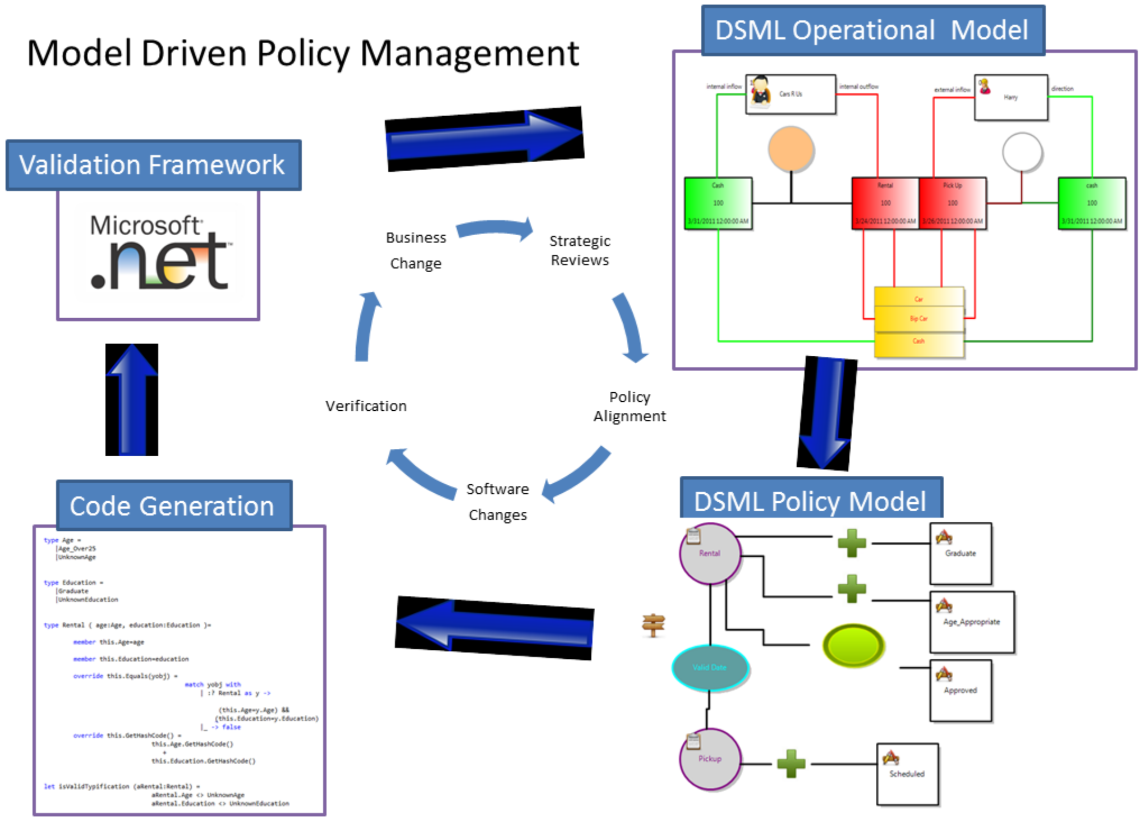
Figure Two illustrates model driven policy management where an REA-EO operational model is used to express business exchanges (Sonnenberg, Huemer et al. 2011). The operational model provides a focal point to identify and refine business policies. These policies result in policy models which are then transformed into validation code and applied within the .Net framework.

We first created a REA-EO DSML for a case study of car rental exchanges. The purpose of this DSML operational model was to experiment with scenarios and help managers identify required policies to respond to customer requests, ensure consistency of pricing and routing, and direct rental operations. The resulting collection of polices were then modeled through a DSML driven by the policy meta-model which formally defined policies. The resulting policies models were translated into F# code and plugged into the .Net validation framework. Policies expressed as code were then applied to validate further refinements to the evolving operational model. As system implementation proceeded, the policy code base was ultimately embedded into production code to validate user input and assure system transactions produced valid states.

Conclusion

We identified key challenges involved in modeling business policies within a DSML. Business models that include business policies must match the dynamics of business environments where business strategies, markets and technologies are constantly changing. The DSML prototypes allowed modelers to flexibly manage policies at a higher level and demonstrated that functional programming languages, such as F#, offer a promising approach to support maintainability, scalability and reliability of business policies. The DSML prototypes instilled confidence by addressing a coherent and integrated view of rental exchanges policies investigated in a case study. The evaluation provided insight into managing policy elicitation, policy formalization, code generation and validation and offers encouragement for further investigations of DSML policy languages.

Figure Two. The DSML Policy model was evaluated in the context of Model Driven Policy Management.



```
type Age =
    |Age_Over25
    |UnknownAge


type Education =
    |Graduate
    |UnknownEducation


type Rental ( age:Age, education:Education )=

        member this.Age=age

        member this.Education=education

        override this.Equals(yobj) =
                                        match yobj with
                                             | :? Rental as y ->

                                                   (this.Age=y.Age) &&
                                                  (this.Education=y.Education)
                                             |_ -> false
        override this.GetHashCode() =
                                this.Age.GetHashCode()
                                    +
                                this.Education.GetHashCode()


let isValidTypification (aRental:Rental) =
                                aRental.Age <> UnknownAge
                                aRental.Education <> UnknownEducation
```

References

Cabot, J. and E. Teniente (2007). "Transformation techniques for OCL constraints." Science of Computer Programming 68(3): 152-168.

Cho, H. (2011). A demonstration-based approach for designing domain-specific modeling languages. Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. Portland, Oregon, USA, ACM: 51-54.

Frantz, R. Z. (2011). "A domain-specific language to desing enterprise application integration solutions." International journal of cooperative information systems : IJCIS 20(2): 143-176.

Geerts, G. L. and W. E. McCarthy (2006). "Policy-Level Specifications in REA Enterprise Information Systems " Journal of Information Systems(Fall 2006).

Hevner, A. R., S. T. March, et al. (2004). "Design Science in Information Systems Research." MIS Quarterly 28(1): 75-105.

Mernik, M., J. Heering, et al. (2005). "When and how to develop domain-specific languages." ACM Comput. Surv. 37(4): 316-344.

Sonnenberg, C., C. Huemer, et al. (2011). The REA-DSL: A Domain Specific Modeling Language for Business Models. Advanced Information Systems Engineering. H. Mouratidis and C. Rolland, Springer Berlin / Heidelberg. 6741: 252-266.

Sprinkle, J., M. Mernik, et al. (2009). "Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?" Software, IEEE 26(4): 15-18.

Syme, D., A. Granicz, et al. (2010). Expert F# 2.0 (Expert's Voice in F#). New York,N.Y., Springer-Verlag.

Umble, E. J., R. R. Haft, et al. (2003). "Enterprise resource planning: Implementation procedures and critical success factors." European Journal of Operational Research 146(2): 241-257.