Extending a Policy Language in a Structured way using Model Driven Techniques

Zohra Ahsan Khowaja and Stephan Reiff-Marganiec

Department of Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK
{zak4, srm13}@le.ac.uk

Abstract. Policy languages have been used for a variety of applications in software systems – usually each application has received its own language. An example of a policy language that has been designed with domain specialization in mind is APPEL, however so far no structured way for domain specialization has been designed. In this paper we use model driven design techniques, in particular parameterization, to present a framework for providing said structured way for adopting/extending APPEL to a specific domain. We exemplify the approach with a case study.

1 Introduction and Motivation

Policies are descriptive and provide information which can be used to modify the behaviour of a system without the need of recompilation and deployment [1]. They are usually written in a policy description language (PDL) which allows end users to specify their requirements, preferences and constraints [2] and are used in software application areas such as e-commerce, network management and access control where specific languages are developed for each area. APPEL (the Adaptable Programmable Policy Environment Language)[3] is a generic PDL which can be used in a variety of application domains once a domain extension has been defined.

Concerns about policy languages in addition to being domain specific, include simplicity and analysability. Using models will aid with analysing policies, e.g. for policy conflict [4], as some issues can be addressed at more abstract model levels and will then automatically hold for all specializations.

Typically new languages (or language variants) are developed in a pragmatic way when a new domain is being addressed, which clearly is a shortcoming. Furthermore specializations are often quite refined in order to enable end users to write meaningful policies – we shall see in the example that they are not just for the domain, but actually much closer to an application level.

In this paper we us a model driven development approach to develop a framework for extensions of policy languages in a structured way – hence the main contribution is applying existing model driven techniques in a what we believe a new domain. We will use the APPEL language as an example, and use the process of parametrization twice, first to specialize APPEL with domain concepts in the Workflow and SoA domain and then for bringing the concepts of a particular application into the language.

2 Background

APPEL [3] is a general language for expressing policies in variety of application domains. It is conceived with a clear separation between core language and specialization for concrete domains. APPEL is designed for end users; its style is close to natural language permitting ordinary user to formulate policies.

APPEL has been developed for call control but has since been used for a number of other domains, including sensor networks and service oriented architecture (SoA)[5]. Each time the specialization has been ad-hoc.

STPOWLA (Service Target Policy Workflow Approach) [5] addresses the integration of business processes, policies and SoA at a high level of abstraction. It captures essential requirements at a business level in the form of workflows and the variability in terms of policies that are expressed in a language close to the business goals. The domain specialisation is for workflows, but clearly those and the policies themselves need to also talk about application specific concepts: for example banking applications differ from traffic management.

The syntax of the ad-hoc extension to APPEL used in STPOWLA introduces special tigger events such as *task_entry* notifying of starting a task in the workflow and specific actions, e.g. $req(\_, \_, \_)$ which takes three arguments: the type of Service (i.e. the basic functionality), a list of service parameters and a service level requirement both of which are application specific. There are further additional triggers and actions that can be used in policies; these are not application but rather workflow domain specific (for details see [6]).

Model Driven Development (MDA) provides a conceptual framework for using models and applying transformations thus allowing to understand complex ideas while providing reuse of common approaches [7]. Models can be used as concrete artefacts [8] and be subjected to operations such as construction, projection and transformation. In particular, meta models allow the definition of models by specifying abstract and concrete syntax as well as semantics of a problem domain. Parameterization allows for a metamodel to be enriched by means of another metamodel, or in our case the core language model can be specialised for domains that are modelled seperately.

3 Metamodel for APPEL Policy Language

The APPEL policy language has been designed with extensibility and domain customization in mind. So far extension have been rather ad-hoc. This section formalizes APPEL by providing a metamodel as a basis for structured extensions. The core language defines the basic constructs needed in all policies such as triggers, conditions and actions and their relationships. The details are left for later definition as required for a particular domain. We use the core concepts of APPEL to design a metamodel for APPEL, shown in Fig. 1. A *policy* contains a *policy rule group*, that contains one or more *policy rule*s. Policy rules can be combined with operators (g:guarded, u:unguarded, par:parallel, seq: sequential). Each policy rule consists of *triggers*, *conditions* and *actions*, where triggers and conditions are optional. A policy rule is applicable if its trigger occurred and its

condition is satisfied (if no trigger or condition is specified they are considered to be trivially true). Triggers, conditions and actions occur in both simple and complex forms where the latter are obtained using specific operators. Additionally policies are assigned a *location*, the exact semantics of this depends on the domain (for workflows a task forms a suitable location).

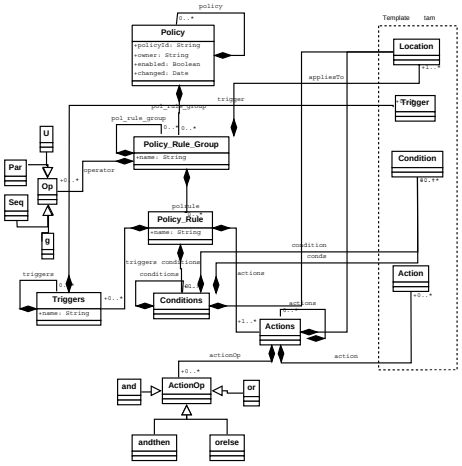

Fig. 1. APPEL Metamodel

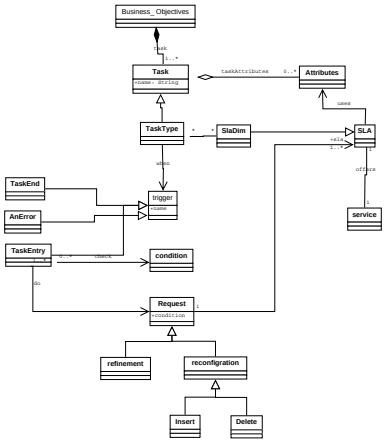

Fig. 2. Domain Model: Service Oriented Business Process

4 PDL Specialisation for Domains and Applications

A domain is an area giving specific meaning to a set of concepts and terminology. A policy language needs vocabulary to reflect this, typically actions and triggers are specific to the domain. When APPEL has been extended to deal with workflows, new concepts have been made available, e.g. "task_entry" as a trigger. The domain specialization to workflows needs a domain model for workflows to be composed with APPEL. However, the case of workflows brings a very interesting item to be: What is an action at domain level? Leeting it simply be the execution of a task is not of much use to the practioner who requires concepts from a specific application domain and hence specialization beyond "workflows" is needed to express e.g "the booking of a hotel" or "payment for a car repair".

Clearly this argument is not restricted to actions, but applies to other elements as well. We refer to this second level of specialization as application, but it is again just a model composition. Based on this argument, we will now consider model composition by parameterization and apply this twice to specialize a policy language: first to make the language domain specific, then to add the specific concepts for an application domain.

The APPEL core language leaves the details of triggers, conditions and actions undefined, leaving a need to specialize these with domain concepts. This is achieved by defining the domain concept as a metamodel which is then used as a parameter to the APPEL metamodel, to produce a target model using a

standard composition approach. In general, parameterization results in some elements in a model being refine by concepts from another model. The process of parameterization [9] can be defined in terms of the APPEL policy language as: $tm = am[tam \xleftarrow{\phi} dm, F_{dm}]$.

The metamodel of the APPEL policy language am is represented in Fig. 1. We consider the elements *location*, *trigger*, *condition* and *action* as template tam – they are the places where the APPEL core language is specialised to a new domain. The domain model dm is represented in Fig. 2 . The operator ϕ replaces elements of tam by dm as follows[1]:

$\phi : \{< Location, Task >, < tam.Trigger, dm.Trigger >,$
$< tam.Condition, dm.Condition >, < Action, Request >\}$

By applying ϕ elements in the am are replaced resulting in an APPEL model specialized for the Business Workflows domain tm.

The domain model represented in Fig. 2 represents the concepts of service oriented business processes. These concepts are abstract at this stage, for example $TaskType$ cannot be defined at this level and *attributes* also come from specific application domains. The model tm can be enriched by filling these further gaps with concepts from the specific application.

In this paper we consider an *On Road Assistant* scenario from the SENSORIA project as an application model, but the technique is general for any application domain. In brief the scenario is as follows: The diagnostic system of a car engine reports a severe failure in the engine, for example a low oil level. This triggers sending of a message with the diagnostic data and the vehicles GPS data to a service centre. Based on availability and the drivers preferences, the service discovery system identifies and selects the appropriate services for garages, tow trucks and rental cars in the area. An appointment is booked with the garage, a towing service is identified and notified to attend to the problem and to tow to the selected garage. The example introduces many domain concepts and these are modeled as in Fig. 4.

In order to specialize the target model tm with the application we have to repeat the process of parameterization to bring the concepts of the application into the language. The final model fm is shown in Fig. 3.

The following policy is a typical example of a policy – it is a policy that refers to the example workflow (Figure 5) where it attaches to the *OrderGarage* task and expresses that if the car fault happens in the driver's home town the driver will select the garage, otherwise one is chosen automatically:

Policy *P*1 **appliesTo** *OrderGarage*
when taskEntry **if** location=myTown **do** req (*main*, [ ], [*Automation* = *interactive*])
seq
when taskEntry **do** req (*main*, [ ], [*Automation* = *Automatic*])

The policy contains some core concepts (e.g. when, if and do), some workflow specific (aka domain) concepts (taskEntry and req) but also some application specific terms (OrderGarage, the automation attribute and also the fact that location inside conditions applies to geographic locations). The language gener-

[1] Some of the elements names occur in both the tam and dm model so we reference by prefixing the model using common notation of $< model.element >$

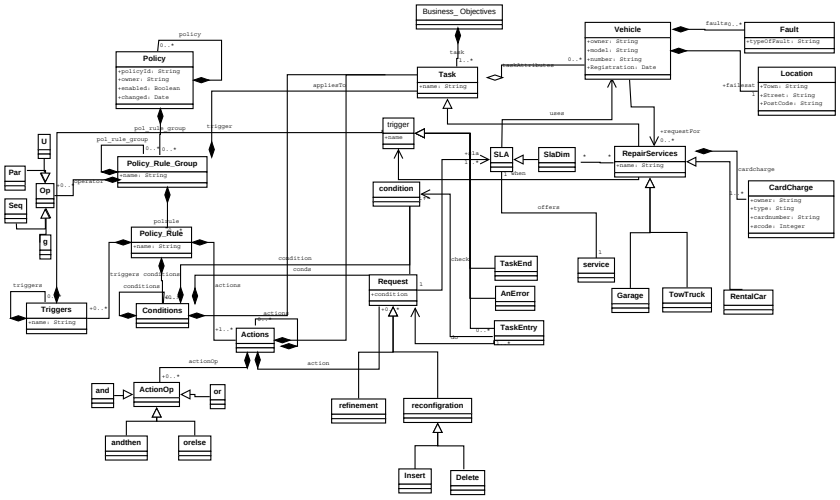

Fig. 3. APPEL with Specific Domain and Application

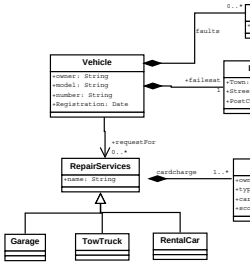

Fig. 4. Application Model: On Road Assistant scenario

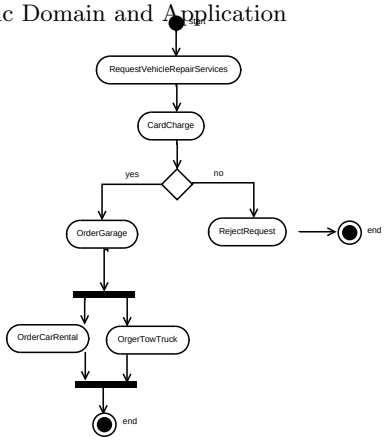

Fig. 5. Workflow: Car Repair Scenario

ated through the two levels of parameterization introduced earlier allows for the specification of exactly these policies.

5 Related Work

Model composition techniques are use in many approaches. The metamodel merge method represented in [10] composes two modeling languages, the constructs of the two languages share a set of real world entities, those concepts are used as join points to stitch the two languages together into a unified whole. This method is of course very close to what we have used in the paper.

The conceptual framework and methodology presented in [9] allows the creation of DSMLs (domain specific modelling languages) for prototyping and verification. This represents the concepts and framework of metamodel extension by

parameterization. Our approach is inspired by the mechanism, but we have to apply the parameterization process several times to get the required metamodel.

[11] introduces a generic policy model where specific PDLs are created as simple extensions of this model. They represents the core concepts of the languages but do not propose an approach or methodology for extending the languages. Our approach is applied to a specific policy language (albeit it will be transferable to other such languages) and focuses on providing a structured approach for tailoring this to specific domains and applications.

6 Conclusion and Future Work

We have presented a framework for using parameterization to specialize general purpose policy languages in a structured way. This has been employed to specialize a specific language (APPEL) to the domain of workflows and then further for specific applications in this domain (we presented a car repair scenario as a sample). What is possibly unique about policy languages is that they are usually created in rather ad-hoc ways for specific application domains. What we think is also quite interesting is the two levels of domain specialization required.

In future work we will consider how the models can be used to provide tools for policy editing in standard ways. More interestingly, we believe that certain aspects of policy conflict can be identified at the model level and dealt with in the abstract. This aspect will deserve some future consideration.

References

1. Lupu, E., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Transactions on software engineering **25**(6) (1999) 852–869
2. Gorton, S., Reiff-Marganiec, S.: Policy-driven Business Management over Web Services. Integrated Network Management, 2007 (2007)
3. Reiff-Marganiec, S., Turner, K.J., Blair, L.: APPEL: The ACCENT policy environment/language. Technical Report CSM-164, University of Stirling (2005)
4. Montangero, C., Reiff-Marganiec, S., Semini, L.: Logic-based conflict detection for distributed policies. Fundam. Inform. **89**(4) (2008) 511–538
5. Gorton, S., Montangero, C., Reiff-Marganiec, S., Semini, L.: StPowla: SOA, Policies and Workflows. Lecture Notes In Computer Science (2009) 351–362
6. Bocchi, L., Gorton, S., Reiff-Marganiec, S.: From StPowla processes to SRML models. Formal Aspects of Computing (2009)
7. Brown, A.: An Introduction to Model Driven Architecture. Website (2004) `http://www.ibm.com/developerworks/rational/library/3100.html`.
8. Muller, A., Caron, O., Carre, B., Vanwormhoudt, G.: On some properties of parameterized model application. Lecture Notes in Computer Science (2005) 130–144
9. Pedro, L.V., Amaral, V., Buchs, D.: Foundations for a domain specific modeling language prototyping environment: A compositional approach. In: Proc. 8th OOPSLA ACM-SIGPLAN Workshop on Domain-Specific Modeling (DSM). (2008)
10. Emerson, M., Sztipanovits, J.: Techniques for metamodel composition. In: 6th OOPSLA Workshop on Domain-Specific Modeling (DSM06). (2006) 123
11. Verlaenen, K., De Win, B., Joosen, W.: Towards simplified specification of policies in different domains. In: Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on. (2007) 20–29