Designing Domain Specific Languages for Verification: First Steps

Phillip James and Markus Roggenbach
Swansea University, UK.

Abstract

This paper introduces a first approach at developing a design methodology for creating domain specific languages focused towards modelling and verification. The work presented is ongoing. The overall aim of the work is to show how capturing domain specific knowledge, and then tailoring proof goals around this domain specific knowledge, can improve automatic verification results, whilst also providing a graphical domain specific language.

1 Introduction

For many years, the application of verification processes such as model checking and interactive theorem proving to varying industrial case studies has been successfully illustrated, e. g. see [22, 19, 9, 10]. Even though these approaches have been successful, the use of formal methods within industry is often still limited. Without experts in the field of formal verification, the verification process is often complicated and hence simply takes too long for the everyday domain engineer to learn effectively. Domain specific languages [7] aim to abstract away such technical details from the domain engineer, allowing them to create programs or specifications without having to be an expert programmer or specifier.

Along with these problems, several research projects within the railway domain have shown that automatic verification can fail when domain models do not contain enough "domain knowledge" [6, 10, 11]. For example, in [10, 11], model checking approaches were applied to verify railway interlockings. The results of the verification were only partially successful, as many of the counter examples produced by the model checking process were later ruled to be impossible by domain experts. The problems encountered were due to underspecified programs created by the domain engineers.

This work, in co-operation with Invensys Rail, aims to show that following a particular design methodology for creating domain specific languages allows the creation of a graphical domain specific language that not only makes the task of automatic verification possible, but also less complex. Such gains are possible via carefully designing a domain specific language to ensure it captures domain knowledge relevant to the class of properties which one would like to verify.

2 The railway domain and DSLs

To illustrate our approach we use the railway domain. Here we review existing work in the area of verification within the railway domain and the area of domain specific language design.

2.1 Modelling and verification in the railway domain

A prominent example of where formal methods have been applied is the railway domain. Approaches that have been taken include algebraic specification, e.g. [4], process algebraic modelling and verification, e.g. [22, 18], and model oriented specification, where, for example the B method has been used in order to verify part of the Paris Metro railway [5] in terms of both safety and

liveness properties. These approaches show the successful application of formal methods to the railway domain, but fail to comment on the applicability of such processes by domain engineers.

This work is inspired by the work of Bjørner [4]. To this end, we follow the natural language specification of the railway domain given by Bjørner [4]. Bjørner has also given a formal version of this natural language specification using the RSL specification language [20]. In contrast, we focus on using CASL, the Common Algebraic Specification Language [16] as it provides us with more features than RSL, including established tool support in the form of the Heterogeneous Toolset (Hets) [15]. The Hets environment not only provides both interactive and automatic theorem proving support, but also allows translation between different logics through institution maps [14]. Such a translation is shown to be useful in Section 3.

2.2 Domain specific language design

The creation of domain specific languages is often aided by the use of a development framework. There are several examples of such tools including ASF+SDF [21] a meta-environment based on a combination of the algebraic specification formalism ASF and the syntax defining language SDF. ASF+SDF allows creation of domain specific languages and tools such as parsers, compilers and static analysers for the created domain specific language. Extending ASF+SDF, there is Rascal [12], which is currently under development at CWI. Finally, MetaEdit+ [2] is an industrial tool allowing the creation of visual domain specific languages. Interestingly, MetaEdit+ has been used to create a domain specific modelling for railway layouts, see [2].

With respect to our approach for creating domain specific languages, we make use of the *Graphical Modelling Framework*, GMF [8]. GMF is an Eclipse plugin that provides the infrastructure to create, from a UML like model, a Java based graphical editor. This editor can then easily be extended with Java code allowing it to output CASL specifications. The simplicity of this creation process fits well with our design methodology outlined in Section 3.

3 Towards a design methodology

In this section, we outline a first proposal for a design methodology for creating domain specific languages for verification. Figure 1 illustrates the proposed design and verification process.

Capturing knowledge: The first area that is illustrated in the left of Figure 1 is the capture of domain knowledge. A natural language specification can be formalised using the OWL Ontology language [3]. OWL has been designed to formalise knowledge about a given domain and thus provides a range of constructs to allow the capture of domain knowledge. It allows specification of *concepts* within a given domain via *classes*, specification of *attributes* of the concepts via *data properties* and specification of *relations* between concepts via *object properties*. It also allows axioms to be stated over such properties. These constructs are very similar to those within UML or any object orientated language. OWL has a well defined formal semantics [17] meaning that every OWL specification has a precise and unambiguous meaning. As we wish to use only automatic tool support, we make use of a decidable fragment of full OWL known as OWL-DL [3]

Creating the DSL: Given an OWL specification, an automatic translation to a GMF (UML like) meta-model is possible.[1] This meta-model along with a set of graphical elements can be used within the GMF process to create a graphical editing tool for the domain. The production

[1] We are currently implementing this XML based translation.

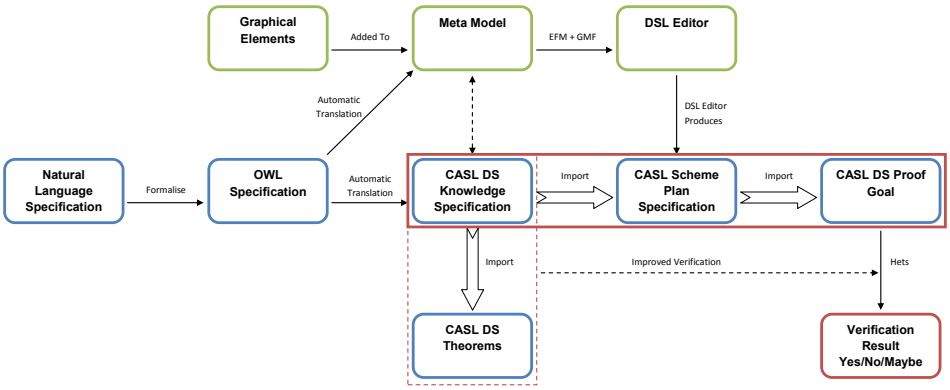

Figure 1: A first design methodology for creation of DSLs focusing on verification.

of this graphical editing tool is outlined in the top branch of Figure 1. Here we provide an overview of the steps involved in the GMF process and for more details we refer the reader to [8]. The first step within the GMF creation process is to select the concepts of the domain which should become graphical constructs within the language. These graphical constructs can be split into two main classes essentially representing nodes and edges within the final graphical editor. The next step is to associate with each chosen construct for the language, a graphical image to represent it. Finally, the attributes (or properties for a given concept) which should be attached to each graphical element can be selected. Once these steps have been completed, the GMF tool will automatically produce a Java based graphical editor. This editor consists of a drawing canvas and a palette. Graphical elements from the palette can be dragged and positioned onto the drawing canvas. Along with these features, the Java code base for the editor is readily extensible and we use this fact to extend the editor to produce CASL specifications. Namely, we add a small amount of code for each construct that simple produces a CASL specification for that construct when it is added to the drawing canvas. Obtaining such a CASL specification for each construct is discuss below.

Semantics: To provide a semantics for the graphical editing tool we propose the use of CASL [16]. The main motivation for the use of CASL is thanks to the tool support that is available in the form of the Hets environment [15]. Hets not only provides syntax checking and static analysis of CASL specifications, but also an interface to various interactive and automated theorem provers. The central path of Figure 1 illustrates the addition of CASL to the graphical editing tool as a semantic base. Within Hets, an automatic semantic preserving translation from OWL into CASL has been implemented [13]. The motivation for using OWL and translating to CASL, rather than directly using CASL, is that OWL provides constructs suited towards capturing domain knowledge in a UML style which can be easily adopted by most domain engineers. Using the resulting CASL domain knowledge specification, the graphical editing tool can be extended to produce CASL specifications for domain models created using the editor.

Verification: Finally, verification of the CASL specifications produced by the graphical editing tool is possible using the Hets framework. At this point, Figure 1 highlights the advantage of adding domain knowledge to the verification process. That is, there are two possibilities for verification. The first – illustrated by the solid lined box – is simply verifying the given problem without any domain specific theorems on the domain knowledge level. The second – illustrated by the dotted lined box – includes domain specific theorems that have been proven on the

domain knowledge level. These theorems provide a potential gain for automated verification: (1) They have the potential to remove false counter examples like those experienced in [10]; (2) They allow general domain specific theorems to be added, these in turn improve the speed of the proof process for particular domain specific proof goals.

4 A first example: Domain knowledge helps

To illustrate the advantages that can be gained through adding domain knowledge to the verification process, we study a common installation within the railway domain. Figure 2 shows part of a standard "double lead" junction.

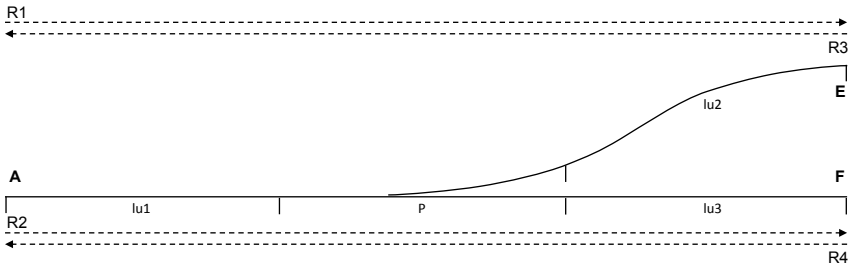

Figure 2: A typical railway junction.

Trains can travel from location A to locations E or F, or from locations E or F to location A. The path along which a train will travel is determined by the position of point P. Logically, the junction is segmented into routes. Here there are four possible routes. Route $R1$ can be set, i.e. trains can travel from A to E when the point is in "reverse" position and there are no trains occupying the point and track segments *lu1* and *lu2*. Route $R2$ can be set, i.e. trains can travel from A to F when the point is set in "normal" position and there are no trains occupying the point and track segments *lu1* and *lu3*. In a similar manner, routes $R3$ and $R4$ can be set to allow trains to travel from E to A and F to A respectively.

```
spec Junction [op p: Switch; op lu1: Linear; ... ] =
    %% axioms for connecting components such as points and tracks
    ...
    forall t: Time . point_EnabledReverseAndLu2At n if p stateAt n = unocc /\
         p positionAt n = reverse /\
         (exists t :Time . n < t /\ lu2 stateAt t = unocc);
    ...
    forall n: Time . route1_enabledAt n  if lu1 stateAt n = unocc /\
         (exists t : Time . n < t /\ point_EnabledReverseAndLu2At t);
    ...
then %implies
    ...
    forall n : Time . exists t: Time . n < t /\ route1_enabledAt t   %(Thm1)%
    forall n : Time . exists t: Time . n < t /\ route2_enabledAt t   %(Thm2)%
    forall n : Time . exists t: Time . n < t /\ route3_enabledAt t   %(Thm3)%
    forall n : Time . exists t: Time . n < t /\ route4_enabledAt t   %(Thm4)%
end
```

Figure 3: A parametrised specification of a junction in CASL.

As such a junction is a common installation within the railway domain, it would naturally form a concept or class within an OWL specification for the railway domain, e.g see [1]. Within CASL, it makes sense to capture a junction with parametrisation. Part of the parametrised CASL specification for the junction is given in Figure 3.

The junction specification illustrates the use of domain specific theorems. These theorems capture domain knowledge about the junction. *Thm1* expresses that there always exists a time in the future where route $R1$ is enabled, and similarly *Thm2*, *Thm3* and *Thm4* expresses this for routes $R2$, $R3$ and $R4$ respectively. Here, due to space constraints, we omit the behaviour of trains and points and assume they behave as expected. These theorems are provable using the Hets toolset in a few seconds.[2] Via instantiation of the junction specification, we can now specify the example train station in Figure 4. This station consists of six junctions in total.

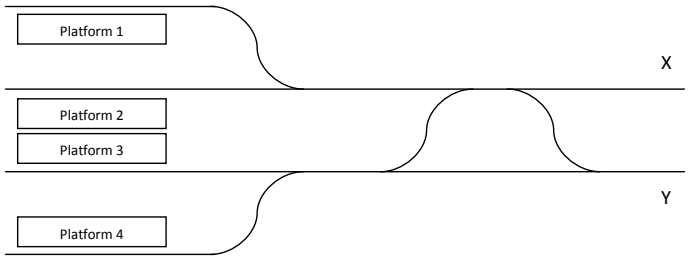

Figure 4: A track plan for an example station.

Over this new track plan for a station, we would like to reason about the enabling of routes allowing trains to enter or leave the station. For example we may wish to know that there always exists a time in the future when a train can leave *Platform 1* and travel to X. This condition is dependent on the setting of several routes across junctions. This property can be expressed as:

$$\forall\ n : Time \bullet \exists\ t1, t2, t3 : Time \bullet n < t1 \wedge t1 < t2 \wedge t2 < t3 \wedge$$
$$route3\_1\_enabledAt\ t1 \wedge route4\_3\_enabledAt\ t2 \wedge route2\_5\_enabledAt\ t3$$

Referring to Figure 1, if we try to verify such a condition without adding domain specific theorems, i.e. *Thm1* through to *Thm4*, to the verification process, then verification with Hets is not possible.[3] When adding the domain specific theorems into the process, the verification is possible within ten seconds. This illustrates that exploiting domain specific knowledge of particular domain constructs can aid the verification process considerably.

5 Summary and future work

In this paper, we have briefly introduced a first attempt at a design methodology for creating domain specific languages focused towards verification. We have also illustrated how the capture and exploitation of domain specific knowledge obtained via this design methodology can provide gains within the automatic verification process. As future work, we wish to explore further examples of how domain specific knowledge can be advantageous. The result will be a classification of types of knowledge and the benefits they can bring to the verification process. We also wish to explore providing useful feedback to domain engineers when a prove attempt is not successful. That is, we wish to explore the production of counter-examples on the level of the graphical editing tool.

[2] Verification times are only rough guidelines and not exact scientific benchmarks.
[3] Within fifteen minutes.

Acknowledgements: We would like to thank our industrial partner Invensys Rail for their useful co-operation throughout this work. A special thanks also goes to Erwin R. Catesbeiana (Jr.) for his reflections and comments on our design methodology.

References

[1] Invensys Rail Data Model – Version 1A, 2010.

[2] MetaEdit+, Webpage, last accessed April 2011. `http://www.metacase.com/`.

[3] G. Antoniou and F. Harmelen. Web ontology language: Owl. *Handbook on ontologies*, 2009.

[4] D. Bjørner. *Domain Engineering – Technology Management, Research and Engineering*. JAIST Press, 2009.

[5] J. Boulanger and M. Gallardo. Validation and verification of METEOR safety software. In J. Allen, R. J. Hill, C. A. Brebbia, G. Sciutto, and S. Sone, editors, *Computers in Railways VII*, volume 7. WIT Press, 2000.

[6] W. Fokkink and P. Hollingshead. Verification of interlockings: from control tables to ladder logic diagrams. In J. Groote, S. Luttik, and J. V. Wamel, editors, *FMICS'98, Formal Methods for Industrial Critical Systems*. CWI, 1998.

[7] M. Fowler and R. Parsons. *Domain Specific Languages*. Addison-Wesley, 2010.

[8] R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. Addison-Wesley Professional, 2009.

[9] G. Holland, T. Kahsai, M. Roggenbach, and B.-H. Schlingloff. Towards formal testing of jet engine rolls-royce BR725. In L. Czaja and M. Szczuka, editors, *Proc. 18th Int. Conf on Concurrency, Specification and Programming, Krakow, Poland*, 2009.

[10] P. James. SAT-based Model Checking and its applications to Train Control Software. Master's thesis, Swansea University, 2010.

[11] P. James and M. Roggenbach. SAT-based Model Checking of Train Control Systems. Technical report, CALCO-jnr'09, University of Udine, n.5-2010, 2009.

[12] P. Klint, T. Van Der Storm, and J. Vinju. EASY Meta-programming with Rascal. *Generative and Transformational Techniques in Software Engineering III*, 2011.

[13] O. Kutz, D. Lücke, T. Mossakowski, and I. Normann. The OWL in the CASL - Designing Ontologies Across Logics. In C. Dolbear, A. Ruttenberg, and U. Sattler, editors, *OWLED*. CEUR-WS.org, 2008.

[14] T. Mossakowski. Relating CASL with other specification languages: the institution level. *Theoretical Computer Science*, 286(2), 2002.

[15] T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set, Hets. *Tools and Algorithms for the Construction and Analysis of Systems*, 4424, 2007.

[16] P. D. Mosses, editor. *CASL Reference Manual*, volume 2960. Springer, 2004.

[17] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. Technical report, W3C, 2004.

[18] J. Peleska, D. Große, A. E. Haxthausen, and R. Drechsler. Automated verification for train control systems. In E. Schnieder and G. Tarnai, editors, *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems*. Technical University of Braunschweig, 2004.

[19] A. Simpson. A formal specification of an automatic train protection system. In G. Goos, J. Hartmanis, and J. V. Leeuwen, editors, *FME '94: Proceedings of the Second International Symposium of Formal Methods Europe on Industrial Benefit of Formal Methods*. Springer, 1994.

[20] The RAISE Language Group. *The RAISE specification language*. Prentice Hall, 1993.

[21] M. Van Den Brand, A. Van Deursen, J. Heering, H. De Jong, M. De Jonge, T. Kuipers, P. Klint, L. Moonen, P. Olivier, and J. Scheerder. The ASF+SDF Meta-environment: A Component-Based Language Development Environment. *LNCS*, 2027, 2001.

[22] K. Winter. Model checking railway interlocking systems. *Australian Computer Science Communications*, 24(1), 2002.