Software & Systems Modeling (2020) 19:17–38
https://doi.org/10.1007/s10270-019-00732-1

SPECIAL SECTION PAPER

Bridging proprietary modelling and open-source model management tools: the case of PTC Integrity Modeller and Epsilon

Athanasios Zolotas[1] · Horacio Hoyos Rodriguez[1] · Stuart Hutchesson[2] · Beatriz Sanchez Pina[1] · Alan Grigg[2] · Mole Li[2] · Dimitrios S. Kolovos[1] · Richard F. Paige[1,3]

Received: 25 July 2018 / Revised: 20 February 2019 / Accepted: 12 April 2019 / Published online: 11 May 2019

Abstract

While the majority of research on Model-Based Software Engineering revolves around open-source modelling frameworks such as the Eclipse Modelling Framework, the use of commercial and closed-source modelling tools such as RSA, Rhapsody, MagicDraw and Enterprise Architect appears to be the norm in industry at present. This technical gap can prohibit industrial users from reaping the benefits of state-of-the-art research-based tools in their practice. In this paper, we discuss an attempt to bridge a proprietary UML modelling tool (PTC Integrity Modeller), which is used for model-based development of safety-critical systems at Rolls-Royce, with an open-source family of languages for automated model management (Epsilon). We present the architecture of our solution, the challenges we encountered in developing it, and a performance comparison against the tool's built-in scripting interface. In addition, we use the bridge in a real-world industrial case study that involves the coordination with other bridges between proprietary tools and Epsilon.

Keywords Model-driven engineering · Model management · Open-source

1 Introduction

Large enterprises often use proprietary and closed-source software and system modelling tools, such as MagicDraw [23], Rhapsody [13] and Enterprise Architect [28] as these come with extensive documentation and are backed by commercial vendors offering guaranteed maintenance and support. By contrast, the majority of research in Model-Based Software Engineering (MBSE) is conducted using open-source modelling tools and frameworks (e.g. Eclipse Modelling Framework (EMF) [29]). This technological gap means that research outcomes are more often than not largely inaccessible to enterprise users. This is clearly detrimental to both enterprise users, who are often unable to readily exploit recent advances in MBSE research, and to researchers, who would benefit from the feedback of enterprise users on the use of research outcomes in industrial-scale applications. In addition, enterprise users are restricted to use only built-in model analysis and management facilities provided by the modelling tool. Companies may see the need to eventual transition from proprietary tools to open-source modelling and model management tools, in order to reduce costs or use state-of-the-art MBSE technologies.

Communicated by Mr. Vinay Kulkarni.

✉ Athanasios Zolotas
thanos.zolotas@york.ac.uk

Horacio Hoyos Rodriguez
horacio.hoyos.rodriguez@ieee.org

Stuart Hutchesson
stuart.hutchesson@rolls-royce.com

Beatriz Sanchez Pina
basp500@york.ac.uk

Alan Grigg
alan.grigg@rolls-royce.com

Mole Li
mole.li@rolls-royce.com

Dimitrios S. Kolovos
dimitris.kolovos@york.ac.uk

Richard F. Paige
richard.paige@york.ac.uk; paigeri@mcmaster.ca

[1] Department of Computer Science, University of York, York, UK

[2] Rolls-Royce, Control Systems, Derby, UK

[3] Department of Computer Science, McMaster University, Hamilton, Canada

The risks associated with this transition may be very high, especially if the *legacy* model has been used to develop safety critical software that has undergone safety assessment as part of certification. In addition, the challenges of such an attempt include the potential need of bridging different technologies which might have an impact in the time required to execute the model management tasks. Any proposed solution should perform as fast as the built-in model management scripting interface (or as close as possible to that).

In this paper, we present the results of collaboration between researchers at the University of York and practitioners at Rolls-Royce, on bridging the gap between a proprietary UML modelling tool, PTC Integrity Modeller (IM), which is used extensively at Rolls-Royce to support MBSE activities, and the open-source Epsilon family of model management languages (http://www.eclipse.org/epsilon/), which is driven by MBSE research primarily conducted at the University of York and Ashton University. In particular, we discuss the design and implementation of an interoperability layer through which Epsilon model management programs (e.g. validation constraints, model-to-model and model-to-text transformations, etc.) can query and modify IM models without needing to transform them to an intermediate representation (e.g. XMI) first. We also report on the findings of experiments which evaluate the performance and maintainability of equivalent model validation rules defined using IM's built-in scripting language (Visual Basic) and Epsilon's EVL language.

This paper is an extended version of the work presented in [34]. Compared to [34], in this paper, we also:

1. Apply our solution to a real-world industrial case study that involves the use of other bridges available in Epsilon. This demonstrates the full potential of the implementation of bridges between open-source and proprietary tools.
2. Include additional related work.
3. Discuss in detail the handling of UML stereotypes.
4. Propose and implement a solution that supports execution in both 32-bit and 64-bit environments instead of the 32-bit-only version presented previously.

The rest of the paper is structured as follows. Section 2 discusses the current practice of MBSE at Rolls-Royce and motivates our work. Section 3 then describes the design and implementation of the IM-Epsilon interoperability layer (*driver*). In Sect. 4, examples of using the bridge are presented. In Sect. 5, the driver is evaluated by executing validation rules on models of real systems provided by Rolls-Royce. In addition, a new case study that involves the coordination of two bridges built for MathWorks Simulink and Microsoft Excel with the PTC IM bridge is presented in the same section. Section 6 presents our observations and the lessons learnt working on this project. Section 7 presents related work, and Sect. 8 concludes the paper and presents directions for future work.

2 Background and motivation

Rolls-Royce has successfully used a combination of UML class and structure models to define the software architecture for Full-Authority Digital Engine Control (FADEC) systems for over 15 years. This approach uses class models to describe the software structure, and employs model-to-text transformation to generate a SPARK [2] implementation. A SPARK profile is used to allow the structure of the SPARK program to be fully described at the lowest modelled level of abstraction.

The UML modelling environment is used to define the architectural framework and the design details for the hosted components. Design artefacts are produced from the UML models through automatic report generation. These are used as configured design artefacts to support the software system approval (certification) process.

The company has more recently started to employ Model-Based Systems Engineering approaches to design and analyse the FADEC system at a higher level of abstraction. This makes use of SysML [11] to produce functional and physical models of the control system and perform early validation of the design choices. Rolls-Royce Control Systems is using MBSE/SysML to capture system and software-level requirements, in accordance with a Control Systems modelling standard that defines the subset of SysML notation to be used and the mandatory information content of the model. This gives significant benefits in terms of both quality and levelling of system requirements but is, however, currently limited to the use of textual requirements within the structural framework of the SysML model.

Automated validation scripts are executed against both the systems and software-level models to ensure consistency, correctness (where possible) and compliance to modelling standards. Currently, the development of these validation scripts is a specialist activity as it requires a relatively deep knowledge of the underlying metamodel used by the modelling tool (IM), plus Visual Basic programming skills to interact with the tool's scripting interface. This approach is also highly coupled with the particular modelling tool, so the validation checks are not easily portable across modelling environments. To leverage higher-level model management (e.g. model validation, M2M and M2T transformation) languages that provide support for different environments, the only available option is to use IM's model exporting facilities which can serialise models in the form of XMI documents. This option has two notable shortcomings:

1. It imposes a significant overhead as even when small changes are made to models within the tool; the whole

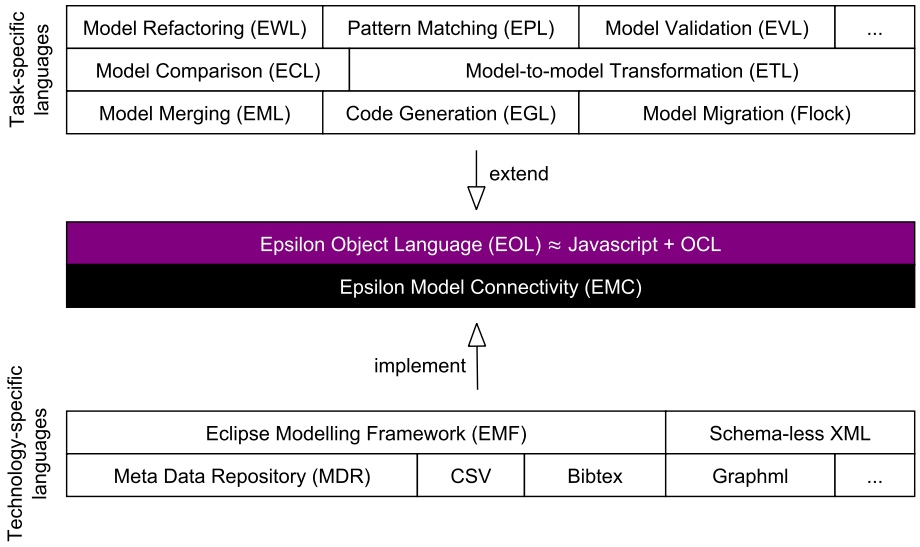


Fig. 1 Architecture of Epsilon (https://www.eclipse.org/epsilon/doc/)

model should be exported again to include the changed elements.
2. Some of the information in the native model representation (particularly diagram layout information) cannot be exported to XMI, which in practice makes programmatic modification and re-importing of the XMI prohibitive.
3. Taking into account the fact that PTC IM can be deployed on a server and the models can be edited by multiple users, this can lead to situations where engineers work with stale versions of the model which may be undesirable especially in the development of safety-critical applications.

To overcome these challenges, particularly with a view to enabling heterogeneous modelling, analysis and code generation in the future, in this work, we developed a direct bridge between IM and the Epsilon family of task-specific model management languages, which provides Epsilon programs with direct and full (read/write) access of in-memory IM models. This solution tackles all the aforementioned shortcomings of the XMI serialisation approach.

3 Bridging Epsilon with PTC Integrity Modeller

In this section, we briefly introduce Epsilon and the Epsilon Model Connectivity (EMC) layer atop which the IM driver[1] has been developed. We also provide a brief overview of IM before and then discuss the architecture and implementation of the driver along with appropriate examples.

3.1 Epsilon

Epsilon is a mature open-source family of interoperable task-specific languages that can be used to manage models of diverse metamodels and technologies. At the core of Epsilon is the Epsilon Object Language (EOL) [19], an OCL-based imperative language that provides support for querying and modifying models conforming to diverse modelling technologies. Although EOL can be used as a general-purpose model management language, its primary aim is to be embedded as an expression language in hybrid task-specific languages. Indeed, a number of task-specific languages have been implemented atop EOL, including languages for model-to-model (ETL) and model-to-text (EGL) transformation (ETL), model comparison (ECL), merging (EML), validation (EVL), refactoring (EWL) and pattern matching (EPL) as illustrated in Fig. 1.

One of the notable features of Epsilon is that its languages are not bound to any particular metamodelling architecture. To treat models of different technologies in a uniform manner and to shield the languages of the platform (and the developers of model management programs) from the intricacies of underlying technologies, Epsilon provides the Epsilon Model Connectivity (EMC) layer (illustrated at the lower part of Fig. 1).

The core abstractions provided by EMC are the *IModel*, *IPropertyGetter* and *IPropertySetter* interfaces, which provide methods for creating, retrieving (by ID or by type) and deleting model elements, and for retrieving and setting the values of their properties, respectively. These interfaces are discussed in more detail in the section that follows while presenting the implementation of the IM driver for Epsilon.

[1] The driver can be installed to an Epsilon distribution from this Eclipse update site: http://zolotas.net/EpsilonPTC/updates/v1.0.1/.

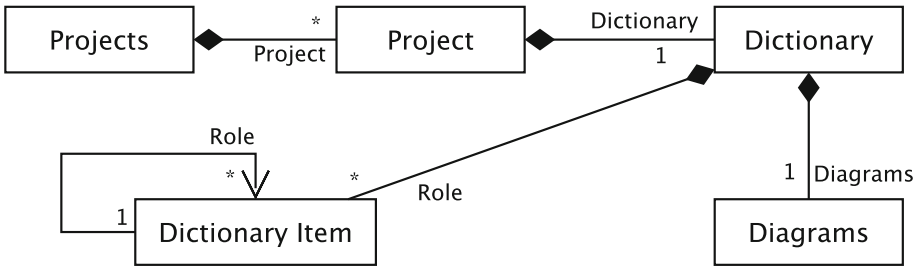


Fig. 2 Metamodel hierarchy in IM repository (taken from [26])

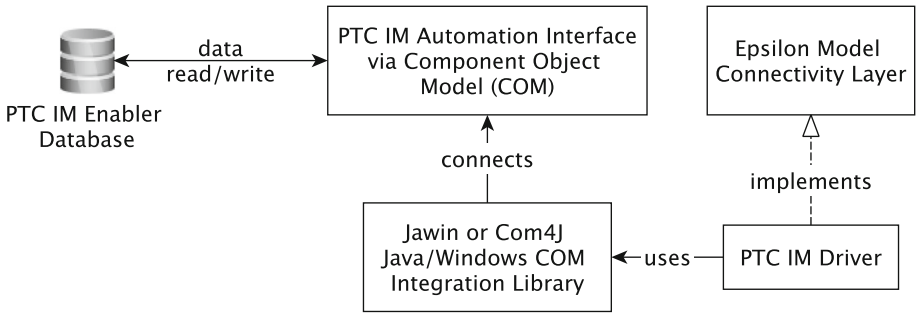


Fig. 3 High-level architecture of the solution

3.2 PTC integrity modeller

PTC Integrity Modeller (formerly known as Atego Artisan Studio) allows the definition of UML and SysML models and diagrams. Among other functionalities, IM offers facilities for synchronisation with other modelling tools (e.g. Simulink [32], Doors [14]) and automatic code synchronisation for many programming languages (e.g. C, Ada, Java).

In IM, models are stored in a centralised object database called Enabler [12], developed by Fujitsu. The model repository consists of three layers: the repository services, the integration services and the user access layer. Models, model elements, relationships, attributes and their values are stored in Enabler's datastore kernel files. The datastore also provides a cache that stores recently accessed elements to improve performance.

Figure 2 shows the organisation of an IM model repository. The *Projects* item holds all the projects in the repository. Each project consists of one *Dictionary* where all model elements (*Dictionary Item*) and a diagram object (*Diagrams*), which holds all the diagrams in the model, are stored. Each model element has a set of attributes and associations (collectively referred to as *properties*) that are common between all types. For example, each element has a unique *id*, a *name* and a *type* attribute. There are also properties which are specific for each type of elements. For example, *IsAbstract* has a value for each element of type *Class* which value is 'true' for a class that is abstract (e.g. *AbstractPersonClass1*). In addition, each property is characterised by four boolean flags: *isReadOnly*, *isAssociation*, *isMultiple* and *isPublic* (e.g. the property *IsAbstract* of the *AbstractPersonClass1* has a value 'false' for the *isAssociation* property). These flags allow the tool to identify which operations are permitted on each property (e.g. if a property is read-only, then setting its value is not allowed).

Engineers are able to access and manipulate model elements programmatically through a scripting API in Visual Basic (VB). Listing 1.1 shows an example VB script that prints the names of all the elements of type *Activity* in the *HSUV* model, which is one of the examples that ship with the tool.

```
1 Dim projects = CreateObject("OMTE.Projects")
2 Dim project = projects.Item("Reference", "\\Enabler\Desktop\Examples\Filling
      Station\0")
3 Dim dictionary = project.Item("Dictionary", "Dictionary")
4 Dim activities = dictionary.Items("Activity")
5 Do While activities.MoreItems
6   a = activities.NextItem
7   Console.WriteLine(a.Property("Name"))
8 Loop
```

Listing 1.1 Example of a Visual Basic program that queries an IM model

3.3 The IM-Epsilon bridge

This section presents the details on providing the IM-Epsilon bridge as an Epsilon EMC Driver. An overview of the high-level architecture of the bridge developed between IM and Epsilon is presented in Fig. 3. IM exposes models through the Automation Interface (the one that is also used for the VB scripting functionality) that provides model query and modification operations. Access to the Automation Interface is given through a Windows COM layer. As the Epsilon Model Connectivity layer requires the extension of three Java interfaces, our integration (labelled *PTC IM Driver* in Fig. 3) needs to use an intermediate layer that realises Java/COM communication. Our initial implementation was based on the open-source Jawin [31] library. This was replaced in the cur-

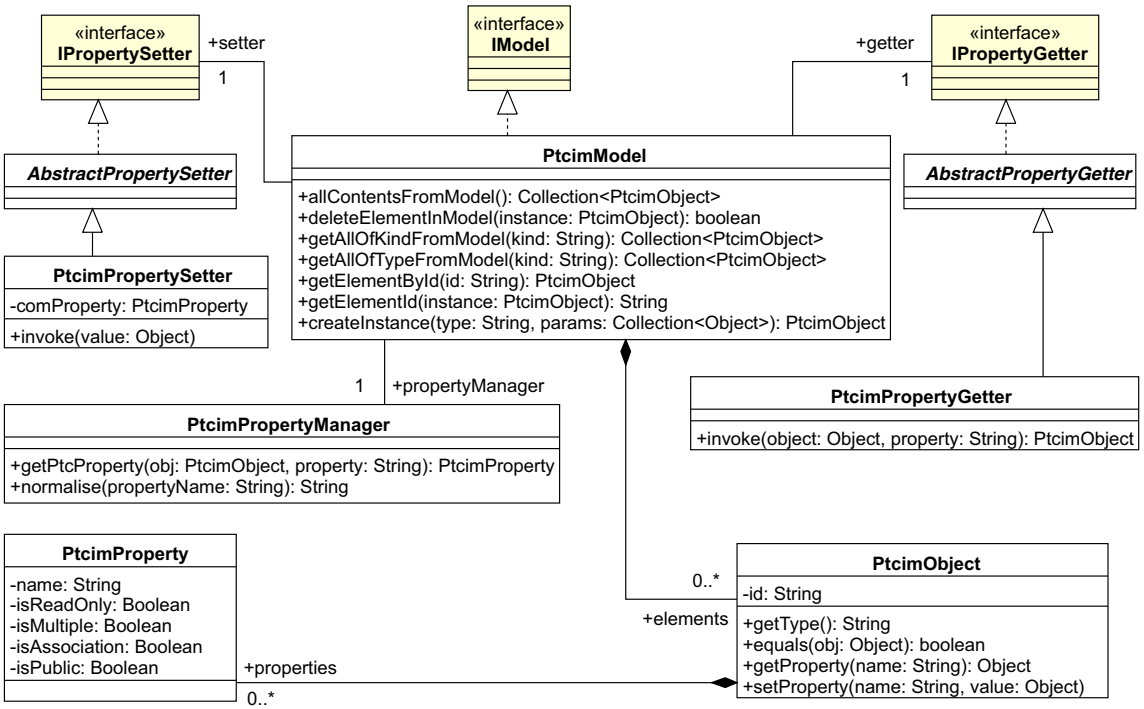


Fig. 4 Class diagram of the IM-Epsilon driver

rent implementation by the com4j [16] library - we discuss the rationale behind this decision in Sect. 3.4.

By using the IM-Epsilon bridge, developers are able to query the IM models, access and modify all model element properties exposed through the COM interface using any of the model management languages that are part of the Epsilon suite. Examples of properties include the *name*, *type* and *last change date* attributes, or the *child object*, *scoping iItem* and *stereotype* associations. A comprehensive list of supported types, attributes and references (i.e. IM's metamodel) can be found in the IM documentation [25].

Figure 4 shows a class diagram of the driver. Every Epsilon driver consists of three main classes that implement the *IModel, IPropertyGetter* and *IPropertySetter* interfaces. For the PTC IM driver presented in this work, these are the *PtcimModel, PtcimPropertyGetter* and *PtcimPropertySetter* classes (see Fig. 4), respectively. The *PtcimModel* class provides (among other) implementations of functions that create new elements, delete existing elements, return all the elements in a model, return all elements of a specific type, etc. The most important methods in the *PtcimModel* class are presented below. For each of them, a mapping to the equivalent IM's COM interface method is given.

- *getAllOfTypeFromModel(type : String) : PtcimObject[]*: This method returns all the elements of the given type.
- *getAllOfKindFromModel(kind : String) : PtcimObject[]*: Returns all the elements of the type including all the elements whose type extends the given type. However, IM does not have a notion of meta-type hierarchy; thus, this method delegates its functionality to *getAllOfTypeFromModel(…)*.
- *allContentsFromModel() : PtcimObject[]*: Returns all the elements in the model.
- *getElementById(id : String) : PtcimObject*: This method returns the element that has a specific id.
- *createInstance(type : String) : PtcimObject*: Creates new elements of a specified type. This is realised by calling the *Add(type)* method in IM COM.
- *deleteElementInModel(element : PtcimObject)*: Deleted elements from the model. By invoking this method, IM also automatically removes all the elements that are connected to this element via associations that are flagged with the *Propagate Delete* value set to true.

A *PtcimModel* consists of a number of *PtcimObject*s which are proxies for the elements of the model and which provide the following methods.

- *getId() : String*: This method returns the unique id of the element.
- *getType() : String*: Returns the type of the element.
- *getProperty(name : String) : Object*: This method retrieves the value of a property. If the property is an attribute, this is achieved by invoking the *Property(arg)* method, else the *Items(property)* or *Item(property)* are invoked

depending on whether the association is multi-valued or single-valued.
– *setProperty(name : String, value : Object)*: This method sets the value of a model element property by invoking the Add(value) method of the COM API if the property is an association or the PropertySet(value) method in case the property is an attribute.

Each model element has a number of properties which are represented as instances of the *PtcimProperty* class. As discussed above, each property in IM has four boolean flags that characterise it (e.g. *isAssociation*, etc.). These flags are retrieved by a method in the *PtcimPropertyManager* class which is described below.

– *getPtcProperty(obj, property)*: This method invokes the *Property('All Property Descriptors')* method of the IM automation interface. The later returns a string containing the four boolean values, separated by the new line character (\n), which are used to create a new*PtcimProperty*.

A getter and a setter are instantiated for each *PtcimModel* and are attached to it. They include methods for getting and setting the value(s) of model element properties, which delegate to the *getProperty(…)* and *setProperty(…)* methods of *PtcimObject* discussed above.

Finally, all property names are normalised using the *normalise(propertyName : String)* method of the *PtcimPropertyManager* class (see Fig. 4). This method turns all characters to lower case and strips all white space. As a result, the user can refer to attributes and associations using different aliases without the need of paying attention to capitalisation and word separation. For example, developers can access the *Child Object* association using any of the following aliases: *childObject*, *childobject*, *ChildObject*, *child object*, etc.

3.4 Java/Windows COM integration

IM exposes all the functionality available to the native scripting facility (using VB), through the Windows Component Object Model (Windows COM). As Epsilon is written in Java, the bridge should also be built using Java. Although COM allows direct interaction with many programming languages, calling COM objects using Java cannot be achieved natively. For that reason, a number of Java/Windows COM interoperability libraries were developed, including Jawin [31], com4j [16] and Jacob [1].

In our previous work [34], we used Jawin as the Java/COM bridge. IM is a 32-bit Windows application and Jawin, which (only) supports 32-bit execution environments, was a good candidate. As described in Sect. 3.1, Epsilon supports the execution of model management scripts on models that are based on a variety of modelling technologies (e.g. EMF, CSV, Spreadsheets, etc.). It is often the case that more than one of the supported modelling technologies are used in the same Epsilon script, as shown in the case study presented in Sect. 5.2, where SysML models are transformed to Simulink models and the latter are simulated using values taken from an Excel spreadsheets. Some of the supported modelling technologies (e.g. Simulink) only support 64-bit execution environments, thus combining 64-bit-only bridges with the 32-bit-only PTC IM bridge was not possible.

Table 1 Time (in seconds) each library needed to execute a set of simple commands 10,000 to IM through COM

Library	32-bit (first/second repeat)	64-bit (first/second repeat)
Com4j	296/304 s	311/322 s
Jacob	304/312 s	315/322 s
Jawin	285/297 s	–

In order to overcome this problem, we investigated the possibility of using a 32/64-bit Java/COM bridge instead. Com4j and Jacob are two libraries that support both 32 and 64-bit execution environments. However, a direct connection of 64-bit environments with PTC IM's 32-bit-only COM interface is not possible. Either a 64-bit DLL should be provided instead (but this requires PTC to develop it) or a workaround of using a surrogate 64-bit COM interface can be used instead. The later will be visible to the 64-bit Java environments and will delegate to the 32-bit provided by PTC IM. This can be done by asking through the OLE/COM Object Viewer the default dllhost to act as the needed surrogate for the 32-bit IM interface. We opted for this second solution.

In order to decide which library we should use for re-implementing the driver, we ran a small-scale experiment. We invoked a set of eleven simple commands to IM through the COM interface using each of the libraries, 10,000 times. We executed the experiment twice for each of the 32 and 64-bit variations. Table 1 summarises the execution times for Jacob and com4J, while the 32-bit times for Jawin are also presented for reference.

As one can see from the results, there are no notable differences between the libraries. We decided to use com4j as it was slightly faster in 32-bit environments than Jacob.

3.5 Caching

In order to be able to offer comparable performance to the built-in scripting interface, the driver provides two different caches. The first one caches the boolean flags for each property and the second the actual value of each property. Both are implemented as instances of the *WeakHashMap* data structure which allow the key to be garbage-collected when there is no reference to it outside the map, making them useful for the implementation of caches. Three new classes are cre-

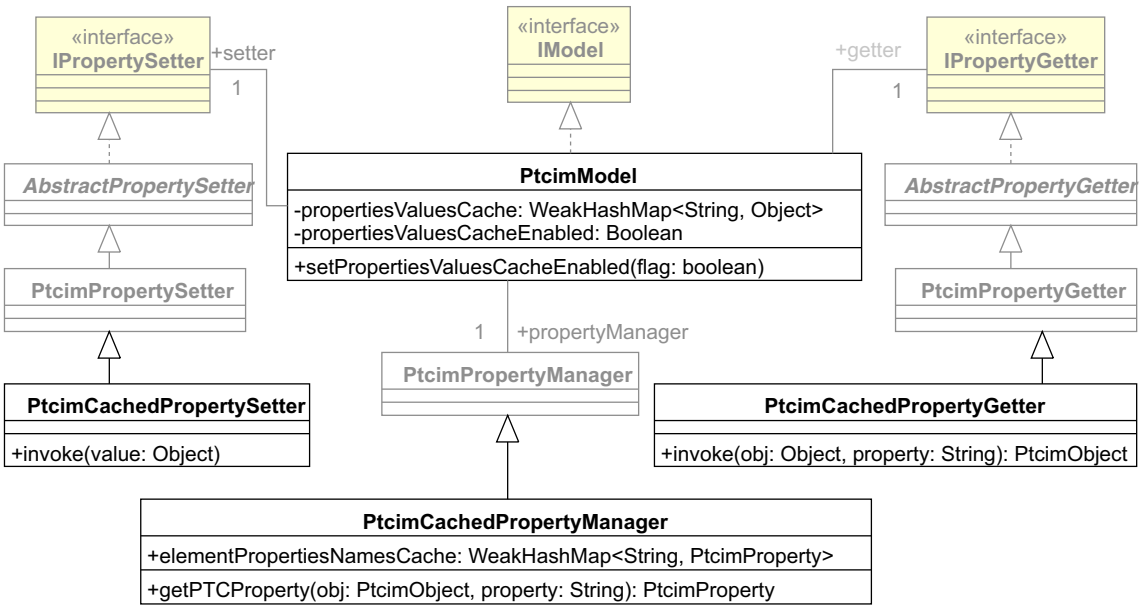
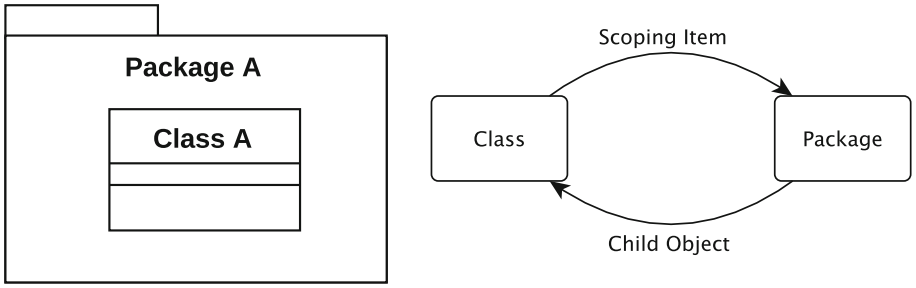


Fig. 5 Class diagram of the caches in IM-Epsilon driver

ated to implement caching. Their relationships with the other classes described before are shown in Fig. 5 and are explained below.

- *PtcimCachedPropertyManager*: The first of the caches (i.e. *elementPropertiesNamesCache*) is hosted in the *PtcimCachedPropertyManager* which extends the *PtcimPropertyManager* class. Elements of the same type have common properties; thus, they share the same boolean flags. This cache maps the fully qualified name of each property to the property's boolean flags following a *type.propertyName* → *PtcimProperty* pattern. For example, all elements of type *Activity* have a property called *isReentrant*. The first time an element of type *Activity* is accessed, an entry in the map is created with *Activity.isReentrant* as key. The four boolean values are queried when creating the PtcimProperty object using the overridden *getPtcProperty(…)* method. If the key (e.g. Activity.isReentrant) exists in the cache, the boolean values are returned. If a property of a type has not been visited before (thus the key is not in the cache), this method delegates to the *super getPtcProperty(…)* method which queries the boolean flags through the COM interface and stores them in the cache.
- *PtcimCachedPropertyGetter*: The second cache (i.e. *propertiesValuesCache*), hosted in the *PtcimModel* class, is used in the *PtcimCachedPropertyGetter* which extends the *PtcimPropertyGetter* class. This cache stores the actual values of the properties of each element. The key used in this cache is constructed by concatenating the unique id of the element and the name of the property that is accessed. For example, the value of the name attribute of an element with id *5eg494* is mapped using the key *5eg494.name* to its value. *PtcimCachedPropertyGetter* overrides the *invoke(…)* method of *PtcimPropertyGetter*. Every time the value of a property needs to be retrieved, the *invoke* method queries the cache first. If a property has not been accessed before (hence the key is not in the cache), the *invoke* method delegates to its superclass implementation to query the value through the COM interface and then stores it in the cache.
- *PtcimCachedPropertySetter*: Caching can lead to inconsistencies when values of properties are changed for the reasons explained below. Thus, value caching is optional. When value caching is enabled, a *PtcimCachedPropertySetter* is created instead of the default *PtcimPropertySetter*. The former overrides the *invoke(…)* method of its superclass. This method adds or updates the mapping *id.property* → *value* to the values cache and then calls its superclass method that updates the property's value in IM.

As mentioned above, value caching can lead to inconsistencies when values of properties are set as a result of *opposite references* in a model. Consider the following example depicted in Fig. 6, the user retrieves the package in which a class is contained via the *Scoping Item* relationship (see Fig. 6b). The package will be stored in the values cache. Next, the user retrieves the contents of that package by navigating the *Child Object* relationship and removes the aforementioned class from its contents (effectively removing the class from the package). The cache will be updated (thus the *Child Object* relationship of the package will not include the class). However, if now the user navigates again the *Scoping Item* relationship of the class, the returned value will be the same package (while it should now be null). This



(a) UML Package and Class. **(b)** Package and Class relationship in IM.

Fig. 6 Example of a value caching failing scenario

is because IM does not expose a special relationship between the two properties (in Ecore terminology these would be *opposite references*) and as such the driver fails to update the cache on both ends consistently. As such, value caching is only safe to use when an IM model is accessed in read-only mode.

Moreover, even in read-only mode, the property values cache—like all caches—has a memory overhead which may not be justifiable (i.e. if the majority of property accesses occur only once). As such, value caching is optional and needs to be enabled/disabled by the developer according to the nature of the model management program.

3.6 Working with stereotypes

Stereotypes in SysML (and UML) are important because they provide the means to extend the meaning of a model element. A stereotype can be used to specialise the type of an element, for example to clarify its purpose or to change its semantics. Additionally, a stereotype can also define additional properties for an element, referred to as tagged definitions. This allows a model of the system to capture additional information that can be used throughout the modelling life cycle.

Profiles are managed differently by different modelling frameworks. Ideally, if profile A can be applied to elements of type Class and it defines an 'isSerializable' (boolean) property, all classes to which the profile is applied should have an additional 'isSerializable' attribute that can be set by the users. Implementation wise, making the profile properties available as attributes of the class is very difficult. The reason is that since profiles are defined by the user, it is impossible to foresee them and attach them to the Class implementation. Depending on the implementation, adding additional attributes dynamically can be difficult or impossible.

A common pattern to solve the aforementioned issue is to model applied profiles as a feature/attribute of Classes. Each of the applied profiles is an instance of the profile definition, and this instance holds the values of the profile attributes for that particular Class. Figure 7 presents this in practice. The *Bean* stereotype is applied to Class elements (blue elements in the diagram). The *Person* and *Account* classes (in violet) have the Bean stereotype applied. The *appliedStereotype* reference contains Bean stereotype instances.

The use of this pattern means that attributes provided via stereotypes cannot be accessed directly when navigating the SysML model. For example, to get the value of the *isSerializable*, the following property access (see Listing 1.2) is necessary:

```
1 Class.all().first().appliedStereotypes().selectOne(s | s.name = "Bean
      ").isSerializable;
```

Listing 1.2 Stereotype attribute access.

On the contrary, IM handles the definition of stereotypes in a dynamic way. That is, once a stereotype is applied to an element, the stereotype properties are directly accessible in the class. That is, Listing 1.2 can be rewritten as shown in Listing 1.3.

```
1 Class.all().first().isSerializable;
```

Listing 1.3 Stereotype attribute access.

Our bridge provides full support for the dynamic stereotype property access as implemented by IM. Consider the following example in which pieces of MATLAB code need to be attached to action nodes in UML activity diagrams. In PTC IM, end users might decide to implement this by creating a new stereotype (e.g. *MATLAB Action*) that has a property (e.g. *MATLAB code*), of type text, where the MATLAB code will be stored. Listing 1.4 shows how the *MATLAB code* property defined in the *MATLAB Action* stereotype applied to an UML action node can be retrieved using EOL.

```
1 var allActionNodes = IM!ActionNode.all();
2 var allActionNodesWithMatlabActionStereotype = allActionNodes.select(n|n.
      appliedStereotypes.name.includes("MATLAB Action"));
3 for (node in allActionNodesWithMatlabActionStereotype) {
4   (node.name + " MATLAB code: " + node.MATLABCode).println();
5 }
```

Listing 1.4 Example of EOL code that prints the content of a custom property created through a stereotype.

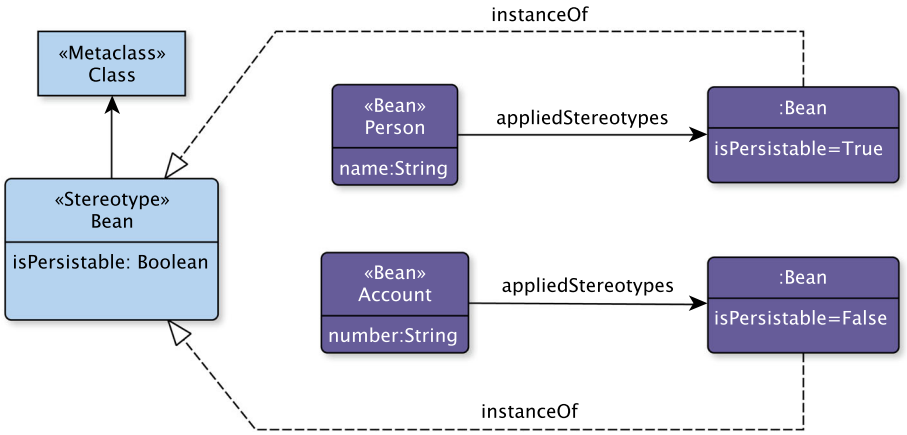


Fig. 7 SysML profiles modelled as class attributes

4 Demonstration

Epsilon scripts can be run either from pure Java applications or within the Eclipse IDE.[2] In this section, we show how to set up run configurations for Epsilon programs to be executed on IM models from within the Eclipse IDE. Information on how to run Epsilon scripts from Java applications can be found in the Epsilon website.[3] Following, we give two examples of Epsilon scripts that access information in an IM model. The examples demonstrate that IM model element properties can be accessed directly by property name via *dot* navigation. Further, the model-to-model transformation script demonstrates that by using other EMC drivers, it is possible to use multiple combinations of proprietary and open-source tools. Finally, we also discuss how UML stereotypes defined in an IM model can be accessed via the Epsilon-IM bridge. Model management scripts can be run through a configuration dialogue which is part of the driver's user interface and allows developers to select and configure IM models to be used in Epsilon programs. The dialogue allows developers to set

- the name through which the Epsilon program can refer to the model (in case the program operates on more than one models concurrently)
- the server that hosts the repository of interest
- the repository that holds the model of interest
- the name of the model in the repository
- whether property value caching should be enabled during execution
- the element to be treated as the root of the model (to limit the scope of a program to a sub-tree of the model)

[2] https://www.eclipse.org, last accessed May 9, 2019.

[3] https://www.eclipse.org/epsilon/examples/index.php?example=org.eclipse.epsilon.examples.standalone, last accessed May 9, 2019.

4.1 Running model management programs

Listings 1.5 and 1.6 show a validation constraint (in EVL [20]) and a fragment of a model-to-model transformation (in ETL [18]) that can be executed against IM models. The constraint of Listing 1.5 checks that the names of all elements in the IM model which are of type *Class* start with an upper-case letter. In line 1, the *context* keyword is used to define the elements to which the constraint applies. In line 2, we declare that this is a soft constraint (*critique*), and in line 3 of the script, the condition to be satisfied is provided following the *check* keyword. If the condition is not satisfied for a particular class, a context-aware warning message is produced in line 4.

```
1 context Class {
2   critique NameShouldStartWithUpperCase {
3       check : self.name.substring(0,1) = self.name.substring(0,1).
              toUpperCase()
4       message : "The name of class " + self.name + " (" + self.Id + "
              ) should start with an upper-case letter"
5   }
6 }
```

Listing 1.5 Example of an EVL critique which checks if the name of a class starts with upper-case letter

It is important to note that the bridge implementation supports not only reading from models created in PTC IM, but also writing to them. The user can create new elements and set or change the values of their properties. This is achieved by using the CreateInstance() and SetProperty() methods of the bridge which delegate to the appropriate PTC IM COM methods as described in Sect. 3.3.

One of the distinguishing features of Epsilon is that it is metamodelling technology agnostic and thus its languages can manage different types of models. Listing 1.6 demonstrates a fragment of a model-to-model transformation that produces a Papyrus [21] UML model from an IM model. The *Package2Package* rule in line 1 transforms all packages in the IM model to packages in the Eclipse UML model. In particular, it copies across the name of the IM package (line

```
1  rule Package2Package
2    transform s : IM!Package
3    to t : UML!Package {
4
5    t.name = s.name;
6    t.nestedPackage ::= s.scopedPackage;
7    t.ownedType ::= s.packageItem.
8      select(pilpi.isTypeOf(IM!Class));
9  }
10
11 rule Class2Class
12   transform s : IM!Class
13   to t : UML!Class {
14
15   t.name = s.name;
16 }
```

Listing 1.6 Fragment of an ETL M2M transformation that produces Eclipse/Papyrus UML models from IM models.

5), it recursively transforms the IM package's sub-packages (line 6), and then, it populates the *owned types* of the UML package with the transformed equivalents of classes under the IM package (lines 7 and 8). The *Class2Class* rule in line 12 transforms IM classes to Eclipse UML classes and copies names across.

5 Evaluation

Having presented the architecture and implementation of the Epsilon-IM driver and how to use it within the Eclipse IDE, in this section we present an evaluation of the IM-Epsilon bridge from two perspectives. The first is an experimental setup to evaluate the performance of the driver against the native Visual Basic support in executing model management tasks on the models stored in PTC IM. The second, which is a new evaluation added to this extended version of the original work presented in [34], is an applicability setup to evaluate the ability to use the driver in a real-world case scenario. The experimental setup is based on a set of validation constraints that capture some of the Rolls-Royce internal modelling standards. The real-world case study describes how an existing manual process can be automated by using Epsilon languages and a combination of EMC drivers.

5.1 Performance experiment

The objective of the experiment is to compare the performance, measured as execution time, of a validation script written in Epsilon's EVL (and run from Eclipse) against equivalent constraints expressed in Visual Basic (and run from within IM). The complete EVL and Visual Basic implementations are listed in the appendix of the paper.

Table 2 The evaluated constraints

Id	Description
#1	Classes' names should start with upper-case letter
#2	Attributes' names should start with lower-case letter
#3	Classes should not have more than seven operations
#4	Operations should not have more than seven parameters
#5	Classes must not have multiple inheritance
#6	The upper multiplicity of aggregation ends must be 1
#7a	The lower bound of an association start must not be greater than its upper bound
#7b	The lower bound of an association start must not be greater than its upper bound
#8a	Numeric upper bounds of association starts must be positive integers
#8b	Numeric upper bounds of associations ends must be positive integers

5.1.1 Experiment setup

Our experiments involved the execution of ten constraints that look for common errors and violations of naming conventions in IM models. Table 2 summarises these constraints.

We executed the constraints on three real models of Rolls-Royce engine controllers constructed using IM and ranging from 13,823 to 116,251 model elements, and on 16 smaller example models that ship with IM. Column *# Elements* of Table 3 summarises the sizes of all 19 models used for our experiments.

Five configurations were used in total: (1) Visual Basic, (2) EVL and the Epsilon-IM driver without caching, (3) with both caches enabled and finally (4, 5) two experiments with *only* one of the two caches enabled each time. The constraints were executed three times on each model, and the execution time was logged for each iteration. The Epsilon cache was empty before each run. To avoid any overheads due to *warm-up* effects on the Java and database level, the first run of each experiment was ignored.

5.1.2 Results

Table 3 summarises the execution times[4] of evaluating the constraints on all models for all five configurations. The models marked with an asterisk are the real-world models constructed by Rolls-Royce. Two line graphs (see Figs. 8 and 9) present the execution times of Visual Basic and EVL (with both caches turned on).

As illustrated in Table 3, the native Visual Basic implementation is faster than all four EVL configurations. In

[4] **Execution environment.** Operating System: Windows 10 Pro 64-bit, CPU: Intel Core i7-6560 @ 2.2GHz, RAM: 16GB @ 1066MHz, Disc: Toshiba XG3 SSD (512GB).

Table 3 Execution time for different models for all five configurations

Model Name	#Elements	Average execution time (in seconds)				
		VB	Epsilon (both caches)	Epsilon (flags cache)	Epsilon (values cache)	Epsilon (no cache)
Template - Small Project	**21**	0.024	0.066	0.072	0.064	0.068
Template - Incremental Process	**32**	0.037	0.082	0.082	0.088	0.088
Heart Monitor C	**109**	0.015	0.196	0.163	0.224	0.284
BallCpp	**123**	0.024	0.296	0.296	0.400	0.520
Heart Monitor Java	**159**	0.022	0.212	0.218	0.274	0.306
Template - Component-based Products	**227**	0.328	0.390	0.380	0.402	0.392
Traffic Lights	**297**	0.067	0.446	0.442	0.814	0.948
Distributed Ball Game MDA Example	**395**	0.074	0.476	0.469	1.035	1.133
VB Another Block (Tetris) Example	**675**	0.295	2.046	2.050	4.780	5.021
C# Another Block (Tetris) Example	**695**	0.301	2.098	2.119	4.607	5.144
Waste System	**815**	0.152	1.273	1.304	2.856	3.296
Traffic Lights - SySim	**1323**	0.267	1.517	1.586	4.206	5.984
Speed Controller	**1405**	0.442	2.143	2.264	5.946	8.191
Filling Station	**1519**	1.010	3.432	3.556	7.636	8.363
HSUV	**2186**	1.304	5.210	5.504	12.693	16.602
Search and Rescue	**5956**	0.965	3.886	4.083	11.418	15.450
Large Civil Aero-Engine 1 Small Model*	**13,823**	7.974	42.797	46.167	141.310	216.010
Large Civil Aero-Engine 2 Control SW*	**90,221**	65.091	410.509	489.496	851.138	1450.564
Large Civil Aero-Engine 3 Control SW*	**116,251**	79.721	713.034	708.492	1474.994	2243.216

particular, EVL (with both caches enabled) is up to almost 10× slower than Visual Basic for the biggest model we have experimented with (116K model elements). This is to be expected given that EVL execution has the overhead of crossing the (expensive) Java-COM bridge every time it needs to fetch new information from the model. Indeed, by profiling the EVL execution we observed that the majority of the execution time (more than 90%) is consumed in the method of the Jawin interface that invokes the COM layer of IM.

The driver configuration that uses no caching is up to five times slower than the configuration that uses both caches. Looking at the respective columns of Table 3, this is largely due to the use of the first (property flags) cache as the constraints do not make heavy reuse of the same property values in order to benefit substantially from the second (property values) cache. This justifies the design decision to make property value caching optional, as its cost (memory overhead) can sometimes outweigh its benefits (performance).

Figures 8 and 9 present the results of Table 3 for the Visual Basic experiment and the Epsilon (both caches enabled configuration).

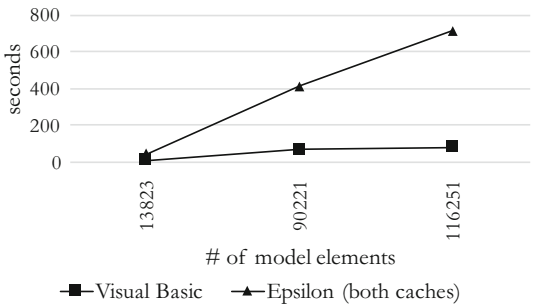


Fig. 8 Execution time of the constraints using VB and Epsilon (both caches enabled) with Rolls-Royce real models

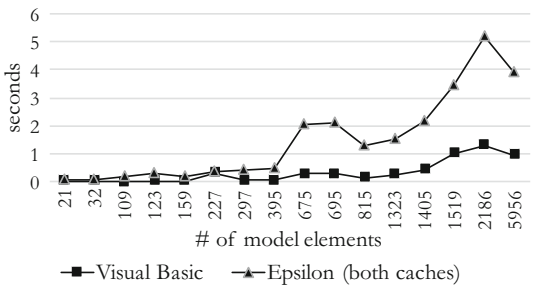


Fig. 9 Execution time of the constraints using VB and Epsilon (both caches enabled) with the IM example models

5.1.3 Threats to validity

For all models, the constraints were violated 12,901 times in total in the case of the Visual Basic and 12,887 for the

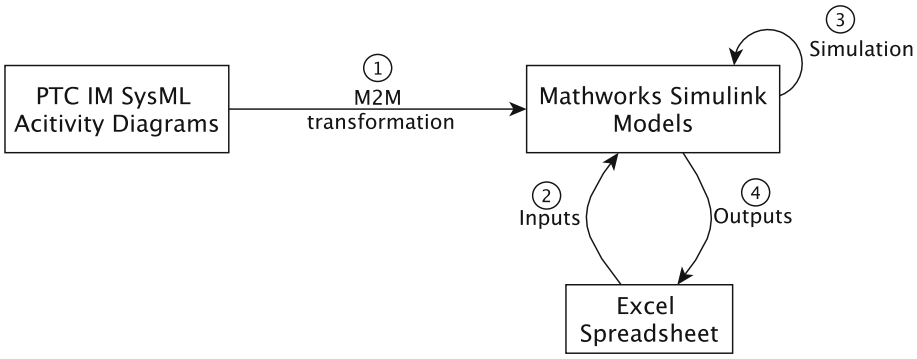


Fig. 10 Overview of the IM to Simulink case study

Epsilon script. By examining the error report, we identified that 12,887 errors and warnings were **identical**, while the 14 extra constraint violations in the Visual Basic implementation were on model elements whose name started with a special character (i.e. <) or a space. The Epsilon script treated the upper-case of this special character as the same of the lower-case, which was not the case in Visual Basic. These 14 additional violations do not significantly impact the logged execution times as the properties and the values of the elements were actually accessed to check the constraint conditions in both cases. However, this highlights the risks in migrating certified applications implemented in one technology to another, as corner cases might be overlooked.

The experiments were run three times on each model. The first execution was ignored to avoid any overhead due to the Enabler database and Epsilon/Java cache warm-up. We ran a small-scale experiment on the example models provided by the tool where we evaluated all five solutions by running the constraints for *ten* iterations and we identified that the execution time was consistent after the second (first, if one does not take into account the warming-up run) execution. As a result, we do not have reasons to believe that the same would not be the case for the three remaining larger models constructed by Rolls-Royce.

5.2 UML activity diagrams to Simulink

One important aspect of modelling is the ability to validate and verify the models. In the previous experiment, we used a model validation DSL, EVL, against the IM models to validate if they have been built according to the project specifications and comply to internal and external standards. However, for verification, it is often the case that separate activities like testing, simulation, among others, must be carried out in order to determine if the system is error-free, produces the correct outputs given a set of known inputs, and such. Currently, IM models must be manually recreated in Simulink in order to simulate them. In this section, we present how Epsilon transformations can be used to automate this process. In particular, we use three proprietary EMC Drivers for IM, Simulink and Excel [10]. A model-to-model transformation written in ETL performs the IM to Simulink transformation, while a script written in EOL performs the simulation and validates the results. This gives the ability to perform early validation of the functional requirements captured in the SysML model (IM) at any given level of system definition by executing the corresponding Simulink model against defined test cases. The proposed solution was applied to a real-world model developed in Rolls-Royce.[5]

Figure 10 presents an overview of the case study. Initially, activity diagrams in PTC IM are transformed to Simulink models (step 1 in Fig. 10). For each of the input pins of the activity diagram, an instantiation value is taken from the Excel document (step 2). The simulation is run (step 3) based on these inputs, and the results produced in the output pins are then stored in the Excel spreadsheet and compared with the expected results (written by the engineers) also stored in the same file (step 4). If all the expected values are the same with those returned, the simulation is marked as 'PASSED' in the spreadsheet.[6] More simulations are run until all the given input values are used. Engineers are able to compare the returned result of the simulation with the expected results. A more detailed explanation of the transformation follows.

Listing 1.7 shows fragments of the M2M transformation (step 1 in Fig. 10). We explain the transformation through a running example. In this, the activity diagram shown in Fig. 11 is transformed to a Simulink model presented in Figs. 12 and 13.

In lines 1–5 of the transformation, the activity diagram (i.e. 'Protection Activity' in Fig. 11) is transformed to a Simulink subsystem (i.e. 'Protection Activity' element in Fig. 12),

[5] The original names of the elements and variables appearing in the models used in this work and provided by Rolls-Royce have been replaced with descriptive ones.

[6] In principle, the simulations could be run automatically taking inputs from Simulink *DataSource* elements. However, in the process followed, we use the Excel spreadsheet to take the input values as in the same file we need to store the actual output, compare the results against the expected output and also mark if the test has passed or failed. Having everything stored in the same file makes debugging easier for the engineers.

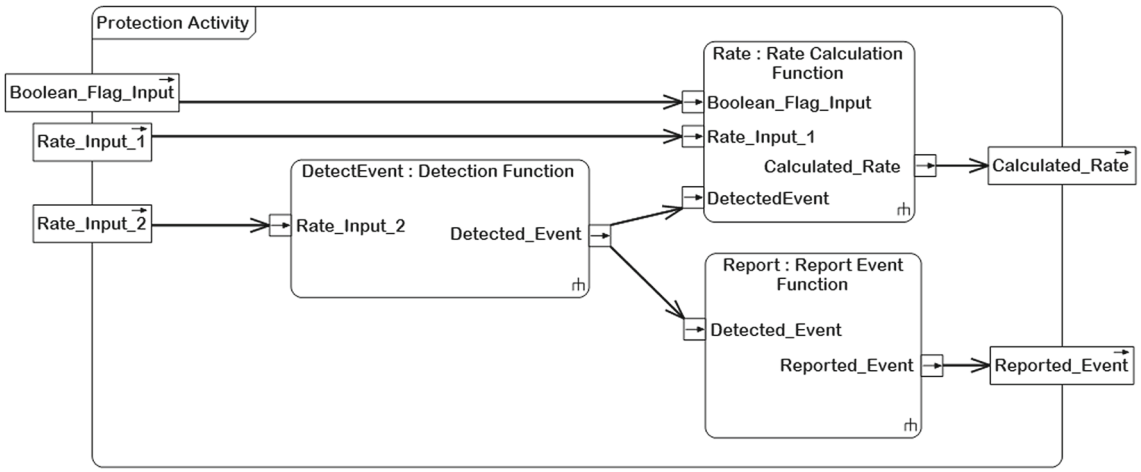


Fig. 11 Example activity diagram in IM

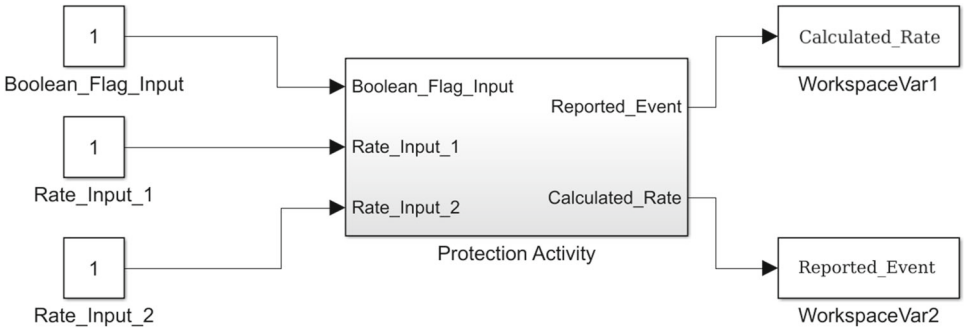


Fig. 12 Simulink generated model for activity diagram of Fig. 11

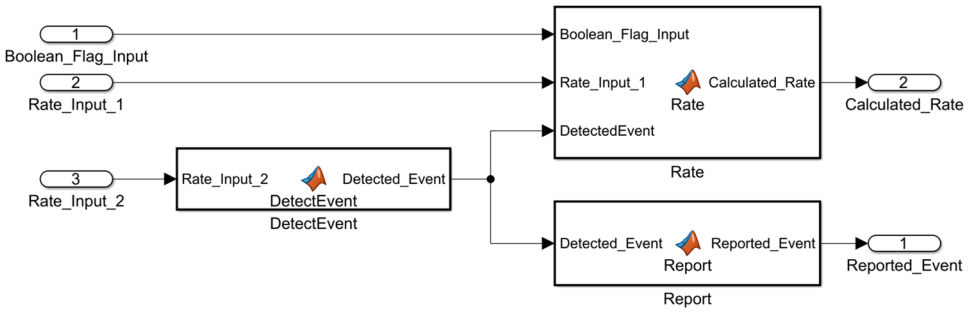


Fig. 13 The contents of the 'Protection Activity' subsystem of Fig. 12

while the name is also copied (line 4). In the transformation rule presented in lines 7–14, a new MATLAB function (e.g. 'DetectEvent' block in Fig. 13) is created for each UML Action Node[7] (e.g. 'Detect Event: Detection Function' in Fig. 11) appearing in the activity diagram. The parent of the Action Node can be found by querying the *Scoping Item* relationship on it. This will return an activity diagram. By using the equivalent notation (::=) in Epsilon, we can get the equivalent MATLAB object that was created for this IM object (i.e. the Subsystem created in the previous rule). The transformation sets the parent of the newly created Function block to this returned element. MATLAB code is stored in a stereotype for each of the action nodes. For example, the 'Detect Event: Detection Function' action node has 'Detected_Event = Rate_Input_2 > 5' MATLAB code stored. This code is retrieved and stored in the attribute *script* of each MATLAB Function block in line 12 (the content of the getScript() operation is omitted for reasons of brevity).

The rule in lines 16–20 create input ports for each UML Input (e.g. 'Rate_Input_1' in Fig. 11) pin. The parent of each pin is identified using again the equivalent (::=) functionality as explained before. In the same manner, output ports and links between the ports are created. The transformation rules for these are not shown here for reasons of brevity. New Simulink elements are created in a default position on the canvas. As a result, the elements overlap, while Simulink

[7] In SysML and PTC IM these are called SysML Call Behaviour Action nodes.

Fig. 14 Simulation scenarios stored in an Excel Spreadsheet

	A	B	C	D	E	F	G	H
1	INPUTS			EXPECTED OUTPUTS		ACTUAL OUTPUTS		RESULTS
2	Boolean_Flag_Input	Rate_Input_1	Rate_Input_2	Calculated_Rate	Reported_Event	Calculated_Rate	Reported_Event	TestResult
3	true	2.0	5.0	0.0	true	0.0	true	PASSED
4	false	5.3	4.0	4.2	false	5.3	false	FAILED
5	true	2.3	1.0	0.0	true	0.0	true	PASSED
6	false	1.5	9.0	0.0	true	0.0	true	PASSED
7	true	5.7	2.9	5.7	false	5.7	false	PASSED

does not offer an auto-layout function. For this reason, in line 13, we assign a position to the newly created element taking its coordinates from the position of the equivalent element in the PTC IM model. The *getPosition()* operation that we defined takes the position of the element in PTC IM (coordinates of the top-left corner) and the width and height of the element and calculates the top-left and bottom-right coordinates that are needed by Simulink. This demonstrates how our bridge can exploit the graphical information provided by PTC IM COM interface. Finally, links are created to connect the output and input pins between activities in lines 32–36. (Low-level implementation details are omitted as these are beyond the scope of this paper.)

```
1  rule Activity2Subsystem
2    transform s : UML!'UML AD Activity'
3    to t : Simulink!'simulink/Ports & Subsystems/Subsystem' {
4    t.name = s.'DictionaryItemName';
5  }
6
7  rule CallBehaviourAction2Function
8    transform s : UML!'UML AD Action Node'
9    to t : Simulink!'simulink/User-Defined Functions/MATLAB Function' {
10   t.parent ::= s.'ScopingItem';
11   t.name = s.'DictionaryItemName';
12   t.script = s.getScript(t);
13   t.position = s.getPosition();
14 }
15
16 rule InputPin2Inport
17   transform s : UML!'UML AD Pin'
18   to t : Simulink!'simulink/Ports & Subsystems/In1' {
19   ...
20 }
21
22 operation Any getPosition() {
23   var left = self.position.split(",").get(0).asInteger()/5;
24   var top = self.position.split(",").get(1).asInteger()/5;
25   var right = left.asInteger() + self.width.asInteger()/4;
26   var bottom = top.asInteger() + self.height.asInteger()/10;
27   return ("[" + left + "," + top + "," + right + "," + bottom + "]"
         );
28 }
```

Listing 1.7 Fragment of an ETL M2M transformation that produces Simulink models from IM models.

At this point, the generated Simulink model is executable. However, one needs to create sample inputs that will resemble test input data and output sinks where the results of the simulation will be stored. This is done in a post-transformation script shown in Listing 1.8 (lines 2–13). We create one 'Constant' input block for each of the input ports in the top-level subsystem and one 'ToWorkspace' output sink for each of the output ports. The values for the inputs in each simulation are taken from the Excel spreadsheet (see Fig. 14—columns A–C). Each row represents one simulation, and as such, we parse the file line by line using the loop in lines 16–33 (see Listing 1.8). For each line, using the Epsilon's Excel spreadsheet bridge, we collect the values for each input and we set them in the constant inputs created in the Simulink model (lines 17–20). We then execute the simulation (line 21). Then, in lines 22–32, the simulation results, stored in the 'To Workspace' variables, are moved to the appropriate Excel columns (F and G in Fig. 14) and are compared with the expected ones (columns D and E). If they match, the script prints the word 'PASSED' in the results column (H in Fig. 14), otherwise it prints 'FAILED'.

6 Observations and lessons learnt

This section summarises the main observations and lessons learnt through our attempt to bridge Epsilon with IM.

Performance Despite using caching aggressively, the performance of the Epsilon IM driver is still substantially inferior (up to 10×) to that of IM's native Visual Basic. While this may not be an issue for smaller models and simple model management activities, it can become disruptive as models and model management programs grow in size and complexity. This observation is consistent with our experiences from attempting to bridge out to other modelling tools such as MetaEdit+[8] and Simulink[9] in a *live* manner through their APIs.

The performance difference was mostly occurred in the communication between our bridge, built-in Java and the COM. In our experiments, over 90% of the execution time was spent in the commands exchange between the two technologies and not internally (i.e. in the handling of objects and the execution of the model management commands by Epsilon). Thus, bridging the Java/COM gap is expensive – but not prohibitive—in terms of performance. This sheds light on the value of open/standard model persistence formats (e.g. XMI) for which performing support can be implemented

[8] https://github.com/epsilonlabs/emc-metaedit.

[9] https://github.com/epsilonlabs/emc-simulink.

```
post {
  // Create constants for inputs
  for (i in Sequence{1..inports.size()}) {
    var constant : new Simulink!'simulink/Sources/Constant';
    ...
  }

  // Create workspace variables to store the output
  var myOuts : new Sequence;
  for (i in Sequence{1..outports.size()}) {
    var out : new Simulink!'simulink/Sinks/To Workspace';
    ...
  }

  "Starting Simulations".println();
  for (v in XL!Values.all) {
    for (inport in inports) {
      var simulinkConstant = Simulink!Constant.all.selectOne(c|c.Name.
          equals(inport.'DictionaryItemName'));
      simulinkConstant.Value =
          getValueOfPropertyWhichNameIsOnlyKnownAtRuntime(v, inport
          .'DictionaryItemName');
    }
    Simulink.simulate();
    var testPassed = "PASSED";
    for (anOutport in outports) {
      var returnValue = Simulink.getWorkspaceVariable(anOutport.'
          DictionaryItemName')[0].println();
      var expectedValue =
          getValueOfPropertyWhichNameIsOnlyKnownAtRuntime(v,
          anOutport.'DictionaryItemName').println();
      var colName = anOutport.'DictionaryItemName';
      v.setValueOfPropertyWhichNameIsOnlyKnownAtRuntime(v, colName,
          returnValue);
      if (returnValue.asString() <> expectedValue.asString()) {
        testPassed = "FAILED";
      }
      v.TestResult = testPassed;
    }
  }
  "Finished Simulations".println();
}
```

Listing 1.8 Fragment of an ETL M2M transformation that executes the simulations on the produced Simulink models.

across different platforms. Yet, not many proprietary products expose the totality of the modelling concepts [6] in these formats.

Interoperability The development of the Epsilon-IM driver has opened a wide range of possibilities for further model-based activities in Rolls-Royce, which were not considered previously, including bespoke Epsilon-based transformations between IM and EMF-based models, between IM and Simulink models (as shown in Sect. 5.2) and synchronisation facilities between IM models and Ada source code. (The latter can be parsed into XML using the AdaCore GNAT toolkit.[10])

Incrementality While the constraints in VB execute faster than those in EVL, their execution time is far from negligible (almost 80 s for the largest model in our experiments), which means that re-evaluating them upon every model change to discover problems as they are being introduced is not a realistic option. To provide near-instant feedback, in the future, constraints need to be executed incrementally as demonstrated in [8]. While this is not easy to achieve using a general-purpose language like VB, it is straightforward to implement using a task-specific language such as EVL or OCL, whose engines provide support for recording property access events [8,24]. IM provides a built-in facility for recording fine-grained model element changes, which is another essential component for achieving performant incremental re-execution of model management programs [24].

Usability Domain-specific languages offer an advantage compared to general-purpose languages in terms of *conciseness* and *expressive power* [33]. Model management programs are noticeably more concise—and therefore arguably easier to write and maintain—when expressed in the task-specific languages of Epsilon compared to Visual Basic. Thus, bridging a modelling tool, like PTC IM, with a model management suite, helped towards the direction of increased conciseness compared to the built-in Visual Basic solution. For example, the same constraints used as part of the performance evaluation (see Sect. 5 and Listings 1.9 and 1.10) are expressed in 119 lines of code using EVL compared to the 200 that the VB approach requires. In addition, although knowledge of VB is a much less rare skill compared to the knowledge of a DSL, like EVL, the latter due to their relatively small size and the fact that they target a specific domain are easier to learn and use. In fact, after a few days of personal study, our collaborators were able to use various Epsilon languages with the bridge we implemented, and they were able to develop complicated scripts after a one-day workshop. In contrast, the bridge offers no gains in terms of the knowledge one should have on the underlying metamodel that PTC IM uses in order to be able to write model management programs. The bridge uses the *same* metamodel as the native solution, thus, regardless of the language one picks (i.e. a DSL or the built-in general-purpose language) they need to understand the modelling tool's structure of the stored models.

Other Despite their independent implementations and their different model representations, IM provided a relatively similar API to that provided by EMF. A notable difference is that the IM API provides built-in support for retrieving all instances of a type, whereas when using the standard EMF interfaces, this needs to be achieved by iterating through all the contents of a model. On the other hand, IM does not support explicit inheritance among meta-types and does not

[10] https://docs.adacore.com/gnat_ugn-docs/html/gnat_ugn/gnat_ugn/gnat_utility_programs.html#the-ada-to-xml-converter-gnat2xml.

provide metadata that allow external tools to identify pairs of *opposite* properties.

From all the above, we believe that a bridging solution could be useful in scenarios where the performance is not an issue (e.g. scripts are run overnight, or only once, or their execution takes a minimal amount of time). In addition, if conciseness and expressiveness is of value and performance is an acceptable trade-off, then bridging solutions offer some benefits. However, we believe that the most important aspect of such a solution is the fact that it allows the seamless integration and exchange of information between multiple modelling technologies (e.g. PTC IM, Simulink, Rhapsody, etc.) as shown in the experiment presented in Sect. 5.2.

7 Related work

In order to manage models from a variety of proprietary modelling tools, we might need a common-ground modelling framework. [6] proposed the definition of a pivot language based on UML profiles and fUML in the open-source Papyrus project to bridge models from an arbitrary number of proprietary tools such as Bridgepoint and Rational Software Architect. The authors discuss how the approach removes the need for import and export facilities tailored for each involved tool [6] although bidirectional mappings between the pivot language and each proprietary tool are still required. In a similar approach, all the Epsilon family of languages (e.g. ETL [18], EVL [20]) can manipulate models or arbitrary modelling technologies, either proprietary (e.g. IM, Simulink) or open source (e.g. EMF), as long as there is an implementation of the Epsilon Model Connectivity (EMC) interface for the technology. Another example is MDEForge [3], which is an extensible software-as-a-service modelling platform that can be used to foster models and execute model-to-model transformations [7] although currently only EMF models and the execution of ATL[30] transformations are supported.

An alternative to manipulate models managed by other modelling tools is to use the import and export facilities of the tools regarding common exchange modelling formats such as XMI. Unfortunately, sometimes tools that provide these import/export facilities do not fully comply with these formats (e.g. PTCIM[11] and GenMyModel[12]). Sometimes, it is open-source projects such as the MATLAB Simulink Integration Framework (MASSIF)[13] that provide these import/export facilities that the proprietary tools do not offer. Work by [15] shows how Simulink models are exported into EMF using MASSIF facilities so they can be consumed by an open-source analysis tool.

When import/export facilities into exchange modelling formats are missing from a modelling tool, developers rely on any available API that could be used to bridge the tool with others. As in our case study, transformations from SysML to Simulink models have motivated several research works such as [5,22,27]. Those three papers have used model-to-text transformation that produce MATLAB scripts which are used to create and populate Simulink models from SysML. In addition, [27] proposes a way back from Simulink into SysML through a MATLAB script that parses the Simulink model and produces an XML model description file that can be parsed by the SysML tool. Further differences in their approaches are that [22] generates several MATLAB scripts that populate different parts of the Simulink model, while [5] proposes a UML profile to annotate the SysML diagrams before the MATLAB code generation. In contrast, our case study used a direct model-to-model transformation that involved three heterogeneous models: IM, Simulink and Excel.

The Open Services for Lifecycle Collaboration (OSLC)[14] is an initiative that aims at simplifying the software tool integration problem among proprietary tools. Built atop the W3C Resource Description Framework (RDF) [17], Linked Data [4], and the REST architecture, OSLC provides a set of specifications targeted at different aspects of application and product lifecycle management. OSLC is becoming widely popular among proprietary tool vendors (e.g. IBM Rational DOORS [14]) who are exposing a range of services following these specifications, and it has also been adopted by open-source tools (e.g. [9]). The comprehensiveness of the model information exposed by these services is at the discretion of the service provider. PTC Integrity Modeller, as of version 8.4 (2017/08), claims to be OSLC-compliant with requirement and architecture management provider services.

8 Conclusions and future work

In this paper, we presented a solution that bridges a proprietary modelling tool, used for modelling safety-critical systems in Rolls-Royce, with the Epsilon open-source model management suite. The Epsilon-IM driver enables programs written in languages of the Epsilon platform to read and write IM models in the context of a wide range of model management activities such as model validation and model-to-model and model-to-text transformation, in conjunction with artefacts captured using different technologies such as Simulink, EMF and Excel spreadsheets.

[11] https://bit.ly/2WtOOOJ.

[12] https://bit.ly/2Uvsy59.

[13] https://github.com/viatra/massif.

[14] http://open-services.net/.

Our evaluation has demonstrated that the price to pay for this flexibility and interoperability is increased execution time, compared to the native Visual Basic scripting facilities provided by IM.

We are currently working on a robust and extensible implementation of incremental model management infrastructure for Epsilon (a proof of concept has already been implemented for EGL [24]), which will enable Epsilon to strengthen its position as the preferred option for interacting with IM models in Rolls-Royce not only from a conciseness and openness but also from a performance point of view.

Acknowledgements This work was partially supported by Innovate UK and the UK aerospace industry through the SECT-AIR project, the Mexican National Council for Science and Technology (CONACyT) under Grant No.: 602430/472773 and the Engineering and Physical Sciences Research Council (EPSRC) through the National Productivity Investment Fund in partnership with Rolls-Royce under Grant No.: EP/R512230/1.

Appendix

Listing 1.9 presents the EVL implementation of the evaluation constraints of Section 5, and Listing 1.10 presents the equivalent implementations in Visual Basic.

```
context Class {
    critique NameShouldStartWithUpperCase {
        check : self.name.substring(0,1) =
            self.name.substring(0,1).
            toUpperCase()
        message : "The name of class " +
            self.name + " (" + self.Id + ")
            should start with an upper-case
            letter. [#1]"
    }
}

context Attribute {
    critique NameShouldNotStartWithUpperCase
         {
        check : self.name.substring(0,1) =
            self.name.substring(0,1).
            toLowerCase()
        message : "The name of attribute " +
             self.name + " (" + self.Id + ")
             should not start with an upper-
            case letter. [#2]"
    }
}

context Class {
    critique
        OperationsShouldeBeLessThanSeven {
        check : self.`operation`.size <= 7
        message : "Class " + self.name + "
            (" + self.Id + ") has more than
            7 operations. [#3]"
    }
}

context Operation {
    critique
        OperationsShouldHaveLessThanSeven
        Parameters {
        check : self.parameter.size <= 7
        message : "Operation " + self.name +
             " (" + self.Id + ") has more
            than 7 parameters. [#4]"
    }
}

context Class {
    constraint
        MultipleInheritanceIsNotAllowed {
        check : self.superclass.size - self.
            superclass.select(i|i.
            isInterface.equals("TRUE")).size
            () < 1
        message : "Class " + self.name + "
            (" + self.Id + ") has multiple
            inheritance. [#5]"
    }
}

context Association {
    constraint
        AggregateStartMultiplicityShould
        BeAlwaysOne {
        check {
            if (self.aggregate.equals("Start
                ") and (not self.
                EndMultiplicityUML.equals
                ("1"))) {
                return false;
            }
            return true;
        }
        message : "Aggregation " + self.name
             + " (" + self.Id + ") has
            multiplicity different than 1.
            [#6]"
    }
}

context Association {
    constraint
        LowerBoundShouldBeSmallerThanUpper
        BoundStart {
        check {
            var startMultiplicity = self.
                startMultiplicityUML;
            if (startMultiplicity.matches
                ("(-)?[0-9]+\\.{2}(-)
                ?[0-9]+")) {
                var lowerBound =
                    startMultiplicity.split
                    ("\\.{2}").get(0);
                var upperBound =
                    startMultiplicity.split
                    ("\\.{2}").get(1);
                if (lowerBound.asInteger() >
                     upperBound.asInteger())
                     {
                    return false;
                }
            }
            return true;
        }
        message : "Lower bound is bigger
            than upper bound in the start of
             association " + self.name + "
            (" + self.Id + "). [#7a]"
    }
}


context Association {
    constraint
        LowerBoundShouldBeSmallerThanUpper
        BoundEnd {
        check {
```

```
            var endMultiplicity = self.
                endMultiplicityUML;
            if (endMultiplicity.matches
                ("(-)?[0-9]+\\.{2}(-)
                ?[0-9]+")) {
                var lowerBound =
                    endMultiplicity.split
                    ("\\.{2}").get(0);
                var upperBound =
                    endMultiplicity.split
                    ("\\.{2}").get(1);
                if (lowerBound.asInteger()
                    > upperBound.asInteger
                    ()) {
                    return false;
                }
            }
            return true;
        }
        message : "Lower bound is bigger
            than upper bound in the end of
            association " + self.name + "
            (" + self.Id + "). [#7b]"
    }
}

context Association {
    constraint
        UpperBoundShouldBePositiveStart {
        check {
            var startMultiplicity = self.
                startMultiplicityUML;
            if (startMultiplicity.matches
                ("(-)?[0-9]+\\.{2}(-)
                ?[0-9]+")) {
                var upperBound =
                    startMultiplicity.split
                    ("\\.{2}").get(1);
                if (upperBound.asInteger()
                    <= 0) {
                    return false;
                }
            }
            return true;
        }
        message : "Upper bound in the start
             of association " + self.name +
             " (" + self.Id + ") should be
            a positive integer. [#8a]"
    }
}

context Association {
    constraint
        UpperBoundShouldBePositiveEnd {
        check {
            var endMultiplicity = self.
                endMultiplicityUML;
            if (endMultiplicity.matches
                ("(-)?[0-9]+\\.{2}(-)
                ?[0-9]+")) {
                var upperBound =
                    endMultiplicity.split
                    ("\\.{2}").get(1);
                if (upperBound.asInteger()
                    <= 0) {
                    return false;
                }
            }
            return true;
        }
        message : "Upper bound in the end
            of association " + self.name +
            " (" + self.Id + ") should be a
             positive integer. [#8b]"
    }
}
```

Listing 1.9 Evaluation constraints implemented in EVL

```
Private Function CheckConstraint1(dictionary
     As Object)
    Dim errorBuilder As New StringBuilder
    Dim c
    Dim Number As Integer
    Dim classes = dictionary.Items("Class")
    Do While classes.MoreItems
        c = classes.NextItem
        Dim cName = c.Property("Name")
        If ((Not Integer.TryParse(cName.
            Substring(0, 1), Number)) And (
            Not Char.IsUpper(cName, 0)))
            Then
            errorBuilder.AppendLine("[VB],
                Class " + cName + " (" + c.
                Property("Id") + ") does not
                 start with uppercase.,[#1]"
                )
            numberOfTotalErrors += 1
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint2(dictionary
     As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim Number As Integer
    Dim attributes = dictionary.Items("
        Attribute")
    Do While attributes.MoreItems
        a = attributes.NextItem
        Dim aName = a.Property("Name")
        If ((Not Integer.TryParse(aName.
            Substring(0, 1), Number)) And (
            Char.IsUpper(aName, 0))) Then
            errorBuilder.AppendLine("[VB],
                Attribute " + aName + " (" +
                 a.Property("Id") + ")
                should not start with
                uppercase.,[#2]")
            numberOfTotalErrors += 1
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint3(dictionary
     As Object)
    Dim errorBuilder As New StringBuilder
    Dim c
    Dim classes = dictionary.Items("Class")
    Do While classes.MoreItems
        c = classes.NextItem
        Dim cName = c.Property("Name")
        If (c.ItemCount("Operation") > 7)
            Then
            errorBuilder.AppendLine("[VB],
                Class " + cName + " (" + c.
                Property("Id") + ") has more
                 than 7 operations.,[#3]")
            numberOfTotalErrors += 1
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint4(dictionary
     As Object)
    Dim errorBuilder As New StringBuilder
    Dim o
    Dim operations = dictionary.Items("
        Operation")
    Do While operations.MoreItems
        o = operations.NextItem
        Dim oName = o.Property("Name")
        If (o.ItemCount("Parameter") > 7)
            Then
            errorBuilder.AppendLine("[VB],
                Operation " + oName + " (" +
                 o.Property("Id") + ") has
```

```
                    more than 7 parameters.,[#4]
                    ")
57              numberOfTotalErrors += 1
58          End If
59      Loop
60      Return errorBuilder.ToString
61  End Function
62
63  Private Function CheckConstraint5(dictionary
        As Object)
64      Dim errorBuilder As New StringBuilder
65      Dim c
66      Dim classes = dictionary.Items("Class")
67      Do While classes.MoreItems
68          c = classes.NextItem
69          Dim cName = c.Property("Name")
70          Dim superClasses = c.Items("
                SuperClass")
71          'Dim numOfSuperClasses = c.ItemCount
                ("SuperClass")
72          Dim numOfNonInterfaces = 0
73          Dim s
74          Do While superClasses.MoreItems
75              s = superClasses.NextItem
76              If (s.Property("IsInterface") =
                    "FALSE") Then
77                  numOfNonInterfaces += 1
78              End If
79          Loop
80          If (numOfNonInterfaces > 1) Then


81              errorBuilder.AppendLine("[VB],
                    Class " + cName + " (" + c.
                    Property("Id") + ") has
                    multiple inheritance.,[#6]")
82              numberOfTotalErrors += 1
83          End If
84      Loop
85          Return errorBuilder.ToString
86  End Function
87
88  Private Function CheckConstraint6(dictionary
        As Object)
89      Dim errorBuilder As New StringBuilder
90      Dim a
91      Dim associations = dictionary.Items("
            Association")
92      Do While associations.MoreItems
93          a = associations.NextItem
94          Dim aName = a.Property("Name")
95          If (a.Property("Aggregate") = "Start
                ") Then
96              If (a.Property("
                    EndMultiplicityUML") <> "1")
                     Then
97                  errorBuilder.AppendLine("[VB
                        ],Aggregation " + aName
                        + " (" + a.Property("Id"
                        ) + ") has multiplicity
                        different than 1.,[#7]")
98                  numberOfTotalErrors += 1
99              End If
100         End If
101     Loop
102     Return errorBuilder.ToString
103 End Function
104
105 Private Function CheckConstraint7a(
        dictionary As Object)
106     Dim errorBuilder As New StringBuilder
107     Dim a
108     Dim associations = dictionary.Items("
            Association")
109     Do While associations.MoreItems
110         a = associations.NextItem
111         Dim aName = a.Property("Name")
112         Dim startMultiplicity = a.Property("
                StartMultiplicityUML")
113         If (Regex.IsMatch(startMultiplicity,
                 "(-)?[0-9]+\.{2}(-)?[0-9]+"))
                Then
114             Dim splitMultiplicity =
                    startMultiplicity.Split(New
                    String() {".."},
                    StringSplitOptions.None)
115             Dim lowerBound =
                    splitMultiplicity(0)
116             Dim upperBound =
                    splitMultiplicity(1)
117             If (lowerBound > upperBound)
                    Then
118                 errorBuilder.AppendLine("[VB
                        ],Lower bound is bigger
                        than upper bound in the
                        start of association " +
                         aName + " (" + a.
                        Property("Id") + ").,[#8
                        a]")
119                 numberOfTotalErrors += 1
120             End If
121         End If
122     Loop
123     Return errorBuilder.ToString
124 End Function
125
126 Private Function CheckConstraint7b(
        dictionary As Object)
127     Dim errorBuilder As New StringBuilder
128     Dim a
129     Dim associations = dictionary.Items("
            Association")
130     Do While associations.MoreItems
131         a = associations.NextItem
132         Dim aName = a.Property("Name")
133         Dim endMultiplicity = a.Property("
                EndMultiplicityUML")
134         If (Regex.IsMatch(endMultiplicity, "
                (-)?[0-9]+\.{2}(-)?[0-9]+"))
                Then
135             Dim splitMultiplicity =
                    endMultiplicity.Split(New
                    String() {".."},
                    StringSplitOptions.None)
136             Dim lowerBound =
                    splitMultiplicity(0)
137             Dim upperBound =
                    splitMultiplicity(1)
138             If (lowerBound > upperBound)
                    Then
139                 errorBuilder.AppendLine("[VB
                        ],Lower bound is bigger
                        than upper bound in the
                        end of association " +
                        aName + " (" + a.
                        Property("Id") + ").,[#8
                        b]")
140                 numberOfTotalErrors += 1
141             End If
142         End If
143     Loop
144     Return errorBuilder.ToString
145 End Function
146
147 Private Function CheckConstraint8a(
        dictionary As Object)
148     Dim errorBuilder As New StringBuilder
149     Dim a
150     Dim associations = dictionary.Items("
            Association")
151     Do While associations.MoreItems
152         a = associations.NextItem
153         Dim aName = a.Property("Name")
154         Dim startMultiplicity = a.Property("
                StartMultiplicityUML")
155         If (Regex.IsMatch(startMultiplicity,
                 "(-)?[0-9]+\.{2}(-)?[0-9]+"))
                Then
156             Dim splitMultiplicity =
                    startMultiplicity.Split(New
                    String() {".."},
                    StringSplitOptions.None)
157             Dim upperBound =
                    splitMultiplicity(1)
```

```
158                 If (upperBound <= 0) Then
159                     errorBuilder.AppendLine("[VB
                            ],Upper bound in the
                            start of association " +
                             aName + " (" + a.
                            Property("Id") + ") must
                             be a positive integer
                            .,[#9a]")
160                     numberOfTotalErrors += 1
161                 End If
162             End If
163         Loop
164         Return errorBuilder.ToString
165 End Function
166
167 Private Function CheckConstraint8b(
        dictionary As Object)
168         Dim errorBuilder As New StringBuilder
169         Dim a
170         Dim associations = dictionary.Items("
                Association")
171         Do While associations.MoreItems
172             a = associations.NextItem
173             Dim aName = a.Property("Name")
174             Dim endMultiplicity = a.Property("
                    EndMultiplicityUML")
175             If (Regex.IsMatch(endMultiplicity, "
                    (-)?[0-9]+\.{2}(-)?[0-9]+"))
                    Then
176                 Dim splitMultiplicity =
                        endMultiplicity.Split(New
                        String() {".."},
                        StringSplitOptions.None)
177                 Dim upperBound =
                        splitMultiplicity(1)
178                 If (upperBound <= 0) Then
179                     errorBuilder.AppendLine("[VB
                            ],Upper bound in the end
                             of association " +
                            aName + " (" + a.
                            Property("Id") + ") must
                             be a positive integer
                            .,[#9b]")
180                     numberOfTotalErrors += 1
181                 End If
182             End If
183         Loop
184         Return errorBuilder.ToString
185 End Function
```

Listing 1.10 Evaluation constraints implemented in Visual Basic

References

1. Adler, D.: The JACOB project: a JAva-COM bridge (2004). http://danadler.com/jacob/
2. Barnes, J.: High integrity Ada: the SPARK approach. Addison-Wesley Professional, Boston (1997)
3. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: MDEForge: an extensible web-based modeling platform. CEUR Work. Proc. **1242**(619583), 66–75 (2014)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. Int. J. Semant. Web Inf. Syst. **5**(3), 1–22 (2009)
5. Chabibi, B., Douche, A., Anwar, A., Nassar, M.: Integrating SysML with simulation environments (Simulink) by model transformation approach. In: Proceedings—25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2016, pp. 148–150 (2016)
6. Cucchiella, S., Cicchetti, A., Ciccozzi, F.: An open-source pivot language for proprietary tool-chaining. In: Proceedings—18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011, pp. 241–250 (2011)
7. Di Rocco, J., Di Ruscio, D., Pierantonio, A., Cuadrado, J.S., De Lara, J., Guerra, E.: Using ATL transformation services in the MDEForge collaborative modeling platform. Lect. Notes Comput. Sci. **9765**, 70–78 (2016)
8. Egyed, A.: Instant consistency checking for the UML. In: Proceedings of the 28th International Conference on Software Engineering, pp. 381–390. ICSE '06, ACM, New York, NY, USA (2006)
9. El-Khoury, J., Ekelin, C., Ekholm, C.: Supporting the linked data approach to maintain coherence across rich EMF models. Lecture Notes in Computer Science, vol. 7949, pp. 36–47. Springer, Berlin (2016)
10. Francis, M., Kolovos, D.S., Matragkas, N., Paige, R.F.: Adding spreadsheets to the MDE toolkit. In: International Conference on Model Driven Engineering Languages and Systems, pp. 35–51. Springer (2013)
11. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language. Morgan Kaufmann, Burlington (2014)
12. FUJITSU Enabling Software Technology GmbH: Enabler Administration, Release 7.0 Service Pack 1 (2006)
13. IBM: IBM—Rational Rhapsody family. Online (2017). http://www-03.ibm.com/software/products/en/ratirhapfami
14. IBM: Rational DOORS. Online (2017). http://www-03.ibm.com/software/products/en/ratidoor
15. Iyenghar, P., Wessels, S., Noyer, A., Pulvermueller, E., Westerkamp, C.: A novel approach towards model-driven reliability analysis of Simulink models. In: IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2016-Novem(d), pp. 1–6 (2016)
16. Kawaguchi, K.: com4j - Type-safe Java/COM bridge (2014). http://com4j.kohsuke.org/
17. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax (2006)
18. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon object language (EOL). In: Rensink, A., Warmer, J. (eds.) Model Driven Architecture – Foundations and Applications. ECMDA-FA 2006. Lecture Notes in Computer Science, vol. 4066, pp. 128–142. Springer, Berlin, Heidelberg (2006)
19. Kolovos, D.S., Paige, R.F., Polack, F.A.: The epsilon object language (EOL). In: Model Driven Architecture–Foundations and Applications, pp. 128–142. Springer, Berlin (2006)
20. Kolovos, D.S., Paige, R.F., Polack, F.A.: On the evolution of OCL for capturing structural constraints in modelling languages. In: Abrial, J.R., Glässer, U. (eds.) Rigorous Methods for Software Construction and Analysis. Lecture Notes in Computer Science, vol. 5115, pp. 204–218. Springer, Berlin, Heidelberg (2009)
21. Lanusse, A., Tanguy, Y., Espinoza, H., Mraidha, C., Gerard, S., Tessier, P., Schnekenburger, R., Dubois, H., Terrier, F.: Papyrus UML: An open source toolset for MDA. In: Proceedings of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009), pp. 1–4 (2009)
22. Natale, M.D., Chirico, F.: An MDA approach for the generation of communication adapters integrating SW and FW components from Simulink. Model Driv. Eng. Lang. Syst. **8767**, 353–369 (2014)
23. No Magic Inc.: MagicDraw. Online (2017). https://www.nomagic.com/products/magicdraw
24. Ogunyomi, B., Rose, L.M., Kolovos, D.S.: Property Access Traces for Source Incremental Model-to-Text Transformation, pp. 187–202. Springer, Cham (2015)
25. PTC: PTC Integrity Modeller. Online (2017). https://www.ptc.com/-/media/Files/PDFs/ALM/Integrity/PTC-Integrity-Modeler-Data-Sheet.pdf
26. PTC Inc.: PTC Integrity Modeler Automation Interface User's Guide Version 8.2 (2015)
27. Sindico, A., Di Natale, M., Panci, G.: Integrating SysML with Simulink using open-source model transformations. SIMULTECH

2011—Proceedings of 1st International Conference on Simulation and Modeling Methodologies, Technologies and Applications, pp. 45–56 (2011)
28. Sparx Systems Pty Ltd.: Enterprise Architect (2019). https://sparxsystems.com/products/ea/
29. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education, London (2008)
30. The Eclipse Foundation: The ATLAS Transformation Language Project. https://www.eclipse.org/atl/
31. The Jawin Project: Jawin—a Java/Win32 interoperability project. Online (2005). http://jawinproject.sourceforge.net/
32. The MathWorks Inc.: Simulation and model-based design. https://www.mathworks.com/products/simulink.html
33. Van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. ACM Sigplan Not. **35**(6), 26–36 (2000)
34. Zolotas, A., Rodriguez, H.H., Kolovos, D.S., Paige, R.F., Hutchesson, S.: Bridging proprietary modelling and open-source model management tools: the case of PTC integrity modeller and epsilon. In: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 237–247. IEEE (2017)

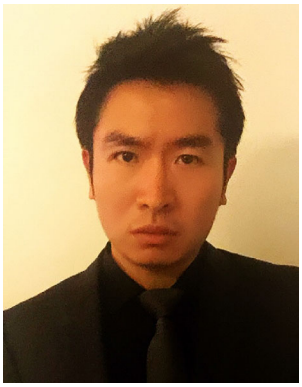
Athanasios Zolotas is a Research Fellow at the Computer Science Department of University of York, UK. Athanasios received his EngD in Large-Scale Complex IT Systems from the University of York in 2017. His research interests are in model-driven engineering, big data analytics, safety critical systems and requirements engineering while he is collaborating with leading companies in the aerospace and automotive domain such as Rolls-Royce and Vollskwagen.

Horacio Hoyos Rodriguez is a Research Associate at the University of York, where he currently researches on incremental execution of model management languages. His main interests are model management languages and model persistence technologies.

Dr. Stuart Hutchesson is Senior Software Specialist in the Control Systems division of Rolls-Royce. He has over 30 years' experience in the development of real-time embedded software for safety-critical systems, primarily FADEC systems for civil aerospace gas-turbine applications. Stuart's current interests include the use of model-based techniques for the specification, generation and verification of systems and software, and in the use of Product Line techniques to develop high-integrity applications. Stuart was a member of the working group that produced DO-331/ED-218 (the Model-Based Supplement to DO-178C/ED-12C). He is a Chartered Engineer and Fellow of both the BCS and IET.

Beatriz Sanchez Pina is a PhD candidate at the Department of Computer Science of the University of York where she received an MSc in Software Engineering in 2017. Her main research focus is on Model-Driven Engineering, Workflows and Traceability. She has been a contributor of the Eclipse Foundation Epsilon project since 2017, in particular, on its integration with the MATLAB Simulink and Stateflow toolboxes.

Alan Grigg graduated in Mathematics at Thames Polytechnic (now University of Greenwich) in 1985. After a spell in BAE Systems working on Integrated Modular Avionics (IMA) standardization programmes, he undertook a PhD at University of York to research timing analysis for distributed real-time systems which he completed in 2002 entitled 'Reservation-based Timing Analysis'. He then worked as part of the BAE Systems Hawk Advanced Jet Trainer software development team to pursue the first project deployment of IMA. He then spent 6 years in the BAE Systems Engineering Innovation Centre at Loughborough University working on collaborative industrial/academic research into novel system and software architectures for avionics. In 2012, he joined Rolls-Royce Control Systems to work on Model-Based Systems Engineering process improvements for Engine Control Systems.



Mole Li received the MSc degree from Loughborough University in 2013. Currently, he is finishing the Ph.D. degree at Loughborough University. His research is in the integration of Software Product Line Engineering with Model-based Systems Engineering. He is a control systems engineer at Rolls-Royce where he is currently working on Model Based Systems Engineering (MBSE) processes, Co-simulation and Model Based Safety Engineering research for Engine Control Systems. Mr. Li is a member of the IEEE, OMG and INCOSE.

Dimitris S. Kolovos is a Professor of Software Engineering in the Department of Computer Science at the University of York, where he researches and teaches automated and model-based software engineering. He is also an Eclipse Foundation committer, leading the development of the open-source Epsilon model-based software engineering platform, and an associate editor of the IET Software journal. Prof. Kolovos has co-authored more than 150 peer-reviewed papers and his research has been supported by the European Commission, UK's Engineering and Physical Sciences Research Council (EPSRC), InnovateUK and by companies such as Rolls-Royce and IBM.

Richard F. Paige is a Professor in the Department of Computing and Software at McMaster University, Hamilton, Canada, and holds a part-time appointment at the University of York, UK. His research interests are in modelling, model transformation, model validation, open-source software and safety-critical systems. He is on the editorial boards of the journals Software and Systems Modelling, Empirical Software Engineering and the Journal of Object Technology. He chairs the steering committee for the STAF conference series, and is on the steering committee for the MoDELS conference.