Bridging Proprietary Modelling and Open-Source Model Management Tools: The Case of PTC Integrity Modeller and Epsilon

Athanasios Zolotas, Horacio Hoyos Rodriguez
Dimitrios S. Kolovos, Richard F. Paige
University of York, United Kingdom
Email:{thanos.zolotas, horacio.hoyos, dimitris.kolovos, richard.paige}@york.ac.uk

Stuart Hutchesson
Rolls Royce, United Kingdom
Email: stuart.hutchesson@rolls-royce.com

***Abstract*—While the majority of research on Model-Based Software Engineering revolves around open-source modelling frameworks such as EMF, the use of commercial and closed-source modelling tools such as RSA, Rhapsody, MagicDraw and PTC Integrity Modeller 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.**

I. Introduction

Large enterprises often use proprietary and closed-source software and system modelling tools, such as MagicDraw [7], Rhapsody [5] and PTC Integrity Modeller [12] 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., EMF [13]). 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 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[1]), which is used extensively at Rolls-Royce to support MBSE activities, and the open-source Epsilon family of model management languages (eclipse.org/epsilon), which is driven by MBSE research primarily conducted at York and Birmingham. In particular, we discuss the design and implementation of an interoperability layer through which Epsilon model management programs (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.

The rest of the paper is structured as follows. Section II discusses the current practice of MBSE at Rolls-Royce and motivates our work. Section III then describes the design and implementation of the IM-Epsilon interoperability layer (*driver*), and in Section IV, the driver is evaluated by executing validation rules on models of real systems provided by Rolls-Royce. Section VI, concludes the paper and presents directions for future work.

II. 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 [1] implementation. A SPARK profile is used to extend the UML, allowing 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 [3] to produce functional and physical models of

[1]We will refer to PTC Integrity Modeller as "PTC IM" or just "IM" in the rest of the paper for brevity.

the control system and perform early validation of the design choices.

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 meta-model used by the modelling tool (IM), 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, large XMI files need to be fully re-exported;
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.

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.

III. 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 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.

A. 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) [8], 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 Figure 1.

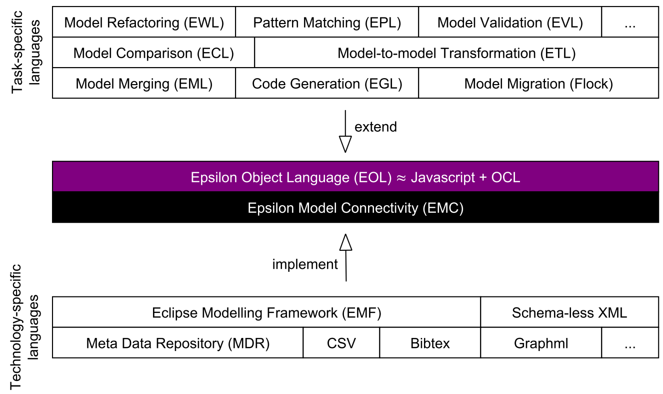

Fig. 1. Architecture of Epsilon[2]

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 Figure 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.

B. PTC Integrity Modeler

PTC Integrity Modeller (formerly known as Atego Artisan Studio) allows the definition of UML and SysML models and diagrams. Among other functionality, IM offers facilities for synchronization with other modelling tools (e.g., Simulink [14], Doors [6]) and automatic code synchronization for many programming languages (e.g., C, Ada, Java).

In IM, models are stored in a centralised object database called Enabler [4], 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 diagrams 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

[2] From https://www.eclipse.org/epsilon/doc/

each type of elements (e.g., elements of type *Class* have a boolean attribute called *IsAbstract*). In addition, each property is characterized by four boolean flags: *isReadOnly*, *isAssociation*, *isMultiple* and *isPublic*. 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).

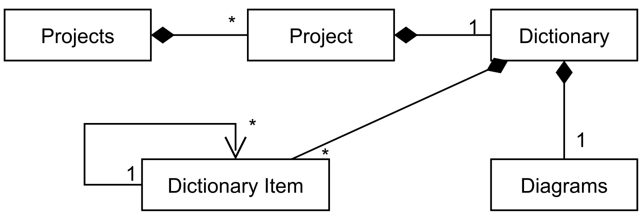

Fig. 2. Metamodel hierarchy in IM repository

Engineers are able to access and manipulate model elements programmatically through a scripting API in Visual Basic. Listing 1 shows an example VB script that prints the names of all the elements of type *Class* in the *HSUV* model which is one of the examples that ship with the tool.

```
Dim projects = CreateObject("OMTE.Projects")
Dim project = projects.Item("Reference", "\\
    Enabler\Desktop\Examples\HSUV\0")
Dim dictionary = project.Item("Dictionary", "
    Dictionary")
Dim classes = dictionary.Items("Class")
Do While classes.MoreItems
  c = classes.NextItem
  Console.WriteLine(c.Property("Name"))
Loop
```

Listing 1. Example of a Visual Basic program that queries an IM model

Figure 3 shows the high level architecture of the developed bridge between IM and Epsilon. IM models are exposed through a Windows COM layer that provides model query and modification operations. Our integration (labelled *IM Driver* in Figure 3) implements the interfaces of the Epsilon Model Connectivity Layer and uses the open-source Jawin [11] library to realise Java/COM communication.

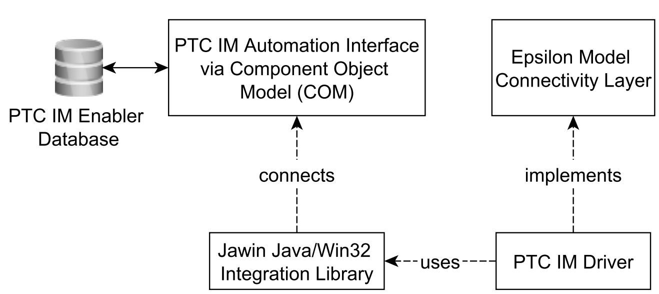

Fig. 3. High level architecture of the solution

C. The Epsilon IM Bridge

Using the Epsilon driver, users are able to query the IM models and access and modify all model element properties exposed through the COM interface. Examples of properties include the *name*, *isAbstract* and *id* attributes, and the *Child Object*, *Owned Constraint* and *Super Class* associations. A comprehensive list of supported types, attributes and references (i.e. IM's metamodel) can be found in the IM documentation [12].

Figure 4 shows a class diagram of the driver. As stated in Section III-A, every Epsilon driver consists of three main classes that implement the *IModel, IPropertyGetter* and *IPropertySetter* interfaces. In the driver presented in this paper, these are the *PtcimModel, PtcimPropertyGetter* and *PtcimPropertySetter* classes (see Figure 4). The *PtcimModel* class provides (among other) implementations of functions that return all elements in a model, retrieve all elements of a specific type, return an element by its id, create new elements and remove them from the model. The following list explains the core methods in the *PtcimModel* class and maps them to the equivalent methods in IM's COM interface.

- *getAllOfTypeFromModel(type : String) : PtcimObject[]*: This method returns all the elements in the model that are instances of the specified type (e.g., *Package*, *Class*). This is achieved by invoking the IM method named *Items(type)* which accepts a parameter specifying the type of interest and returns the unique ids of all the elements of the given type.
- *allContentsFromModel() : PtcimObject[]*: This method returns all the elements in the model. It leverages the same method as above (*Items("")*) but this time an empty string is passed as the type argument.
- *getElementById(id : String) : PtcimObject*: As hinted above, elements in IM have unique ids. This method returns the element that has a specific id by invoking the *ItemById(id)* method in IM.
- *createInstance(type : String) : PtcimObject*: One of the core capabilities of every Epsilon driver is creating new elements of a specified type. In this driver this is realized by calling the *Add(type)* method in IM which creates an element in the model.
- *deleteElementInModel(element : PtcimObject)*: This method can be used to remove elements from the model. This is achieved by invoking the *Remove(id)* method in IM. 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.
- *getAllOfKindFromModel(kind) : PtcimObject[]*: IM does not have a notion of meta-type hierarchy thus, this method delegates its functionality to *getAllOfTypeFromModel(...)*.

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

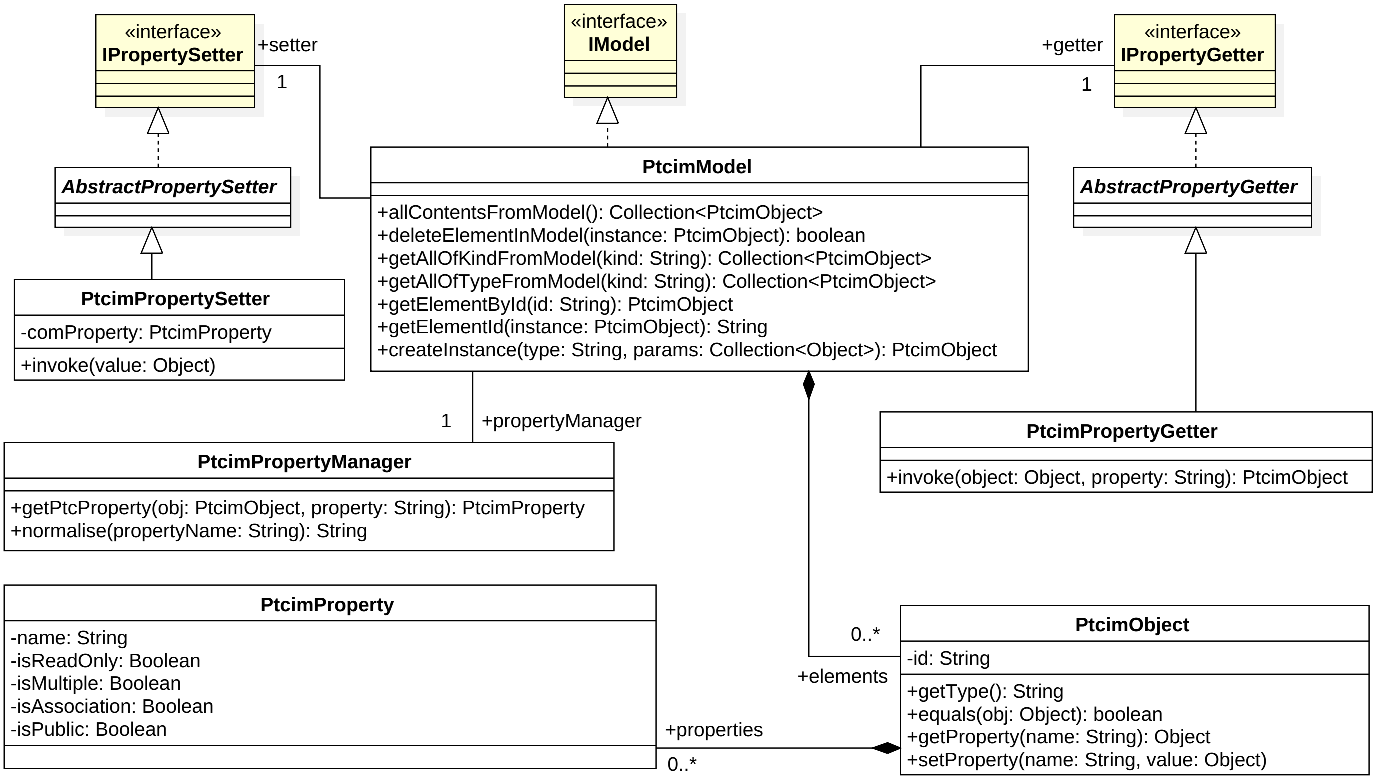

Fig. 4. Class diagram of the IM Epsilon driver

- *getType() : String*: This method returns the type of the element by invoking the *Property("Type")* method of the IM automation interface.
- *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. This method is re-used by *PtcimPropertyGetter* which is explained later.
- *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 of an attribute.
- *equals(obj : Object) : Boolean*: Java's default equality method is overridden as there are cases where the same IM element might be accessed via multiple paths that result in different proxy *PtcimObject*s. For example, a *Class* element can be retrieved through the *Owned Contents* relationship of the package that contains it or via the *Scoping Item* association of one of its attributes. In this scenario, two different proxy objects are created that refer to the same IM element. As such, equality in the driver is checked based on the *id*s of elements.

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., *isReadOnly*, 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*.

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

All property names are normalised using the *normalise(propertyName : String)* method of the *PtcimPropertyManager* class (see Figure 4) which strips all white space and turns all characters to lower case. As a result, the user can refer to the *Child Object* association using any of the following aliases: *childObject*, *childobject*, *Child Object*, etc.

D. *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. *WeakHashMap*s allow the key to be garbage-collected when

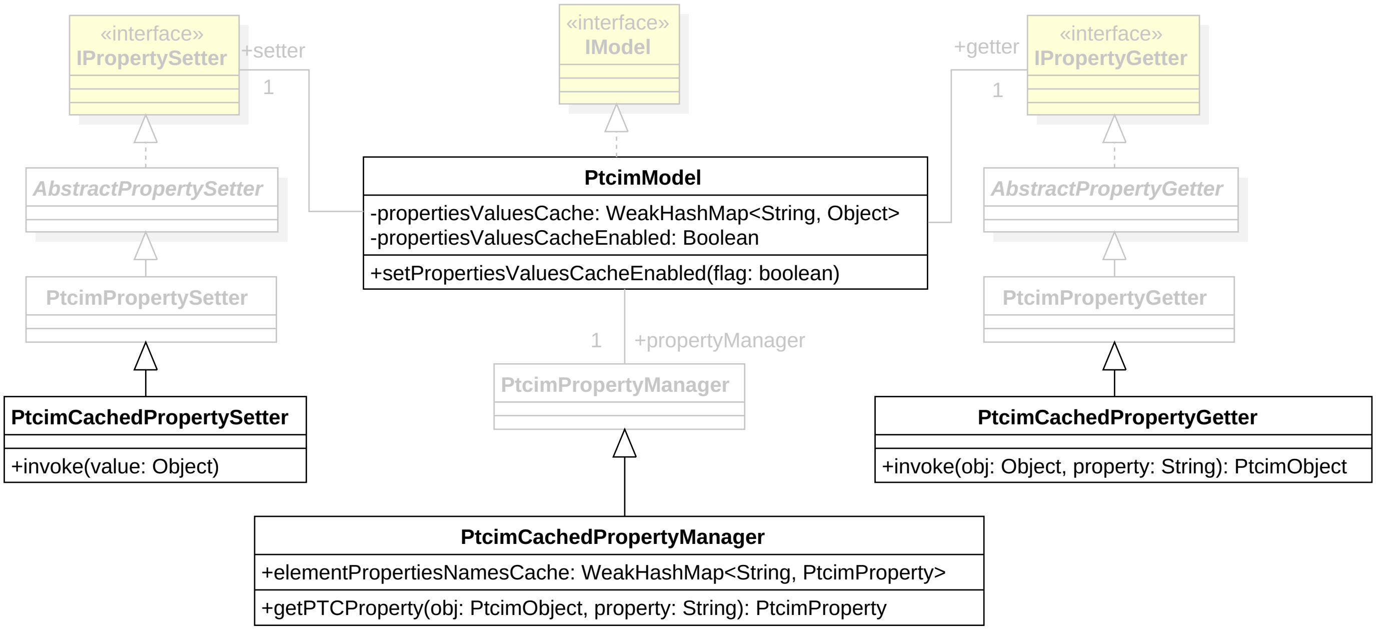

Fig. 5. Class diagram of the caches in IM Epsilon driver

there is no reference to it outside the map, making them useful for the implementation of caches. Figure 5 shows the additional classes needed to implement caching and their relationships with the other classes described above. Three new classes are created for this purpose which are explained below.

- *PtcimCachedPropertyManager*: This class extends *PtcimPropertyManager* and hosts the first of the caches (i.e., *elementPropertiesNamesCache*). As the properties of elements of the same type are common and 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* $\rightarrow$ *PtcimProperty* pattern. For example, all elements of type *Class* have a property called *isAbstract*. The first time an element of type *Class* is accessed an entry in the map is created with *Class.isAbstract* as key. The four boolean values are queried when creating the PtcimProperty object using the overridden *getPtcProperty(...)* method. If the key (e.g., Class.isAbstract) exists in the cache the boolean values are returned. Of course, if a property of a type has not be 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*: This class extends *PtcimPropertyGetter* and uses the second cache (i.e., *propertiesValuesCache*) which is hosted in the *PtcimModel* 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 *5eg4-94* is mapped using the key *5eg4-94.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 be 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*: 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* $\rightarrow$ *value* to the values cache and then calls its superclass method that updates the property's value in IM.

It is important to highlight that value caching can lead to inconsistencies when *opposite references* in a model are modified. Consider the following example: the user retrieves the package in which a class is contained via the *Scoping Item* relationship. 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 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 (see Figure 6) according to the nature of the model management program.

E. Demonstration

Figure 6 shows a configuration dialog 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 dialog 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)

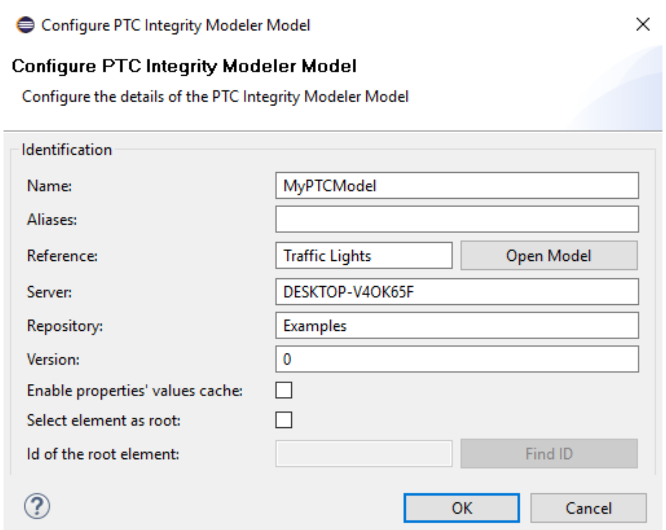

Fig. 6. The IM model configuration dialog in Epsilon

Listings 2 and 3 show a validation constraint (in EVL) and a fragment of a model-to-model transformation (in ETL) that can be executed against IM models. The constraint of Listing 2 checks that the names of all elements is the IM model which are of type *Class* start with an uppercase 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 2. Example of an EVL critique which checks if the name of a class starts with upper-case letter

One of the distinguishing features of Epsilon is that it is metamodeling technology agnostic and thus its languages can manage different types of models. Listing 3 demonstrates a fragment of a model-to-model transformation that produces an Eclipse/Papyrus [9] 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 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.

```
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(pi|pi.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 3. Fragment of an ETL M2M transformation that produces Eclipse/Papyrus UML models from IM models.

IV. Evaluation

Having presented the architecture and implementation of the Epsilon-IM driver, in this section, we present a series of experiments conducted to evaluate its performance. We achieve this by comparing the performance of a set of validation constraints expressed in Epsilon's EVL (which exercise the new driver) against equivalent constraints expressed in IM's native Visual Basic. The complete EVL and Visual Basic implementations are listed in the appendix of the paper. We have chosen model validation as a representative of model management activities that can now be realised with Epsilon through the new driver; other activities such as model-to-model or model-to-text transformation could have been used instead.

TABLE I
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 be smaller than its upper bound
#7b	The lower bound of an association start must be smaller 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

A. Experiment Setup

Our experiments involved the execution of ten constraints that look for common errors and violations of naming conventions in IM models. Table I summarizes the constraints, which were implemented both in Visual Basic and in EVL.

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 II, summarizes 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 first run of each experiment was ignored to avoid any overheads due to *warm-up* effects.

B. Results

Table II summarizes the execution times[3] 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 Figures 7 and 8) present the execution times of Visual Basic and EVL (with both caches turned on).

As illustrated in Table II, the native Visual Basic implementation is faster than all four EVL configurations. In particular, EVL (with both caches enabled) is up to almost 10x 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 II, 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).

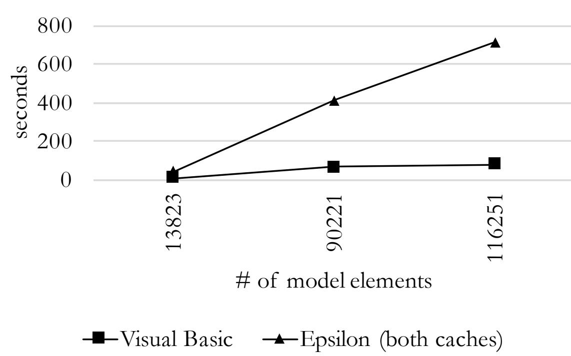

Fig. 7. Execution time of the constraints using VB and Epsilon (both caches enabled) with Rolls-Royce real models

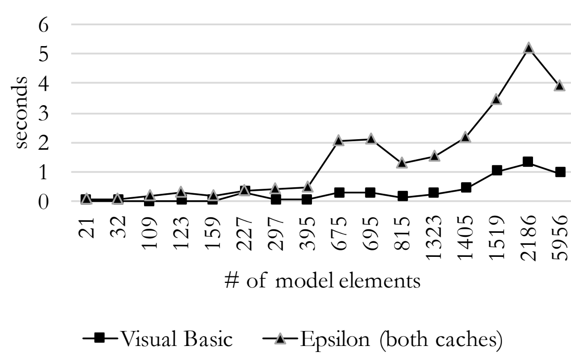

Fig. 8. Execution time of the constraints using VB and Epsilon (both caches enabled) with the IM example models

C. Threats to Validity

For all models, the constraints were violated 12,901 in total in the case of the Visual Basic and 12,887 for the Epsilon script. By examining the error report we identified that

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

TABLE II
EXECUTION TIME FOR DIFFERENT MODELS FOR ALL 5 CONFIGURATIONS

		Average execution time (in seconds)				
Model Name	**#Elements**	**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*	**13823**	7.974	42.797	46.167	141.310	216.010
Large Civil Aero-Engine 2 Control SW*	**90221**	65.091	410.509	489.496	851.138	1450.564
Large Civil Aero-Engine 3 Control SW*	**116251**	79.721	713.034	708.492	1474.994	2243.216

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.

The experiments were run three times on each model. The first execution was ignored to avoid any overhead due to the Enabler database warm-up. Additional iterations would be beneficial; we run 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.

V. Observations and Lessons Learnt

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

a) Performance: Despite using caching aggressively, the performance of the Epsilon IM driver is still substantially inferior (up to 10x) 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+[4] and Simulink[5] in a *live* manner through their APIs. This highlights the value of open/standard model persistence formats for which performant support can be implemented across different platforms, and demonstrates that an externally accessible API is not a good enough substitute (at least performance-wise) for an open model persistence format.

b) Incrementality: While the constraints in Visual Basic execute significantly faster than those in EVL, their execution time for large models is far from negligible (almost 80 seconds 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, constraints need to be executed incrementally as demonstrated in [2]. While this is not easy to achieve using a general-purpose language like Visual Basic, it is straightforward to implement using a task-specific language such as EVL or OCL, whose engines provide support for recording property access events [2], [10]. Our investigation has revealed that 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 [10].

c) 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 transformation and consistency checking facilities between IM and Simulink, transformations between IM and EMF-based models, and synchronisation facilities between IM models and Ada source code (the latter can be parsed into XML using the

[4]https://github.com/epsilonlabs/emc-metaedit
[5]https://github.com/epsilonlabs/emc-simulink

AdaCore GNAT toolkit[6]).

VI. 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.

Our evaluation has demonstrated that the cost of bridging the gap between Epsilon's Java-based execution engines and IM's COM interface becomes significant as models grow in size. On the other hand using task-specific languages such as EVL is promising as, unlike Visual Basic, they have the potential to be executed in an incremental way.

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 [10]), 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.

Acknowledgments This work was partially supported by Innovate UK and the UK aerospace industry through the SECT-AIR project.

Appendix

Listing 4 presents the EVL implementation of the evaluation constraints of Section IV, and Listing 5 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 OperationsShouldeBeLessThanThree {
    check : self.`operation`.size <= 3
    message : "Class " + self.name + " (" + self.Id + ") has more than 3 operations
        . [#3]"
  }
}

context Operation {
  critique OperationsShouldHaveLessThanSevenParameters {
    check : self.parameter.size <= 3
    message : "Operation " + self.name + " (" + self.Id + ") has more than 7
        parameters. [#4]"
  }
}

context Package {
  critique PackagesShouldNotBeEmpty {
    check : self.ownedcontents.size > 0
    message : "Package " + self.name + " (" + self.Id + ") is empty. [#5]"
  }
}

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.
        [#6]"
  }
}

context Association {
  constraint AggregateStartMultiplicityShouldBeAlwaysOne {
    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. [#7]"
  }
}

context Association {
  constraint LowerBoundShouldBeSmallerThanUpperBoundStart {
    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 + "). [#8a]"
  }
}

context Association {
  constraint LowerBoundShouldBeSmallerThanUpperBoundEnd {
    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 + "). [#8b]"
  }
}

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. [#9a]"
  }
}

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. [#9b]"
  }
}
```

Listing 4. 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
```

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

```
            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]")
            numberOfTotalErrors += 1
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint5(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim p
    Dim packages = dictionary.Items("Package")
    Do While packages.MoreItems
        p = packages.NextItem
        Dim pName = p.Property("Name")
        If (p.ItemCount("OwnedContents") = 0) Then
            errorBuilder.AppendLine("[VB],Package " + pName + " (" + p.Property("Id") + ") is empty.,[#5]")
            numberOfTotalErrors += 1
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint6(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")
        Dim superClasses = c.Items("SuperClass")
        'Dim numOfSuperClasses = c.ItemCount("SuperClass")
        Dim numOfNonInterfaces = 0
        Dim s
        Do While superClasses.MoreItems
            s = superClasses.NextItem
            If (s.Property("IsInterface") = "FALSE") Then
                numOfNonInterfaces += 1
            End If
        Loop
        If (numOfNonInterfaces > 1) Then
            errorBuilder.AppendLine("[VB],Class " + cName + " (" + c.Property("Id") + ") has multiple inheritance.,[#6]")
            numberOfTotalErrors += 1
        End If
    Loop
        Return errorBuilder.ToString
End Function

Private Function CheckConstraint7(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim associations = dictionary.Items("Association")
    Do While associations.MoreItems
        a = associations.NextItem
        Dim aName = a.Property("Name")
        If (a.Property("Aggregate") = "Start") Then
            If (a.Property("EndMultiplicityUML") <> "1") Then
                errorBuilder.AppendLine("[VB],Aggregation " + aName + " (" + a.Property("Id") + ") has multiplicity different than 1.,[#7]")
                numberOfTotalErrors += 1
            End If
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint8a(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim associations = dictionary.Items("Association")
    Do While associations.MoreItems
        a = associations.NextItem
        Dim aName = a.Property("Name")
        Dim startMultiplicity = a.Property("StartMultiplicityUML")
        If (Regex.IsMatch(startMultiplicity, "(-)?[0-9]+\.{2}(-)?[0-9]+")) Then
            Dim splitMultiplicity = startMultiplicity.Split(New String() {".."}, StringSplitOptions.None)
            Dim lowerBound = splitMultiplicity(0)
            Dim upperBound = splitMultiplicity(1)
            If (lowerBound > upperBound) Then
                errorBuilder.AppendLine("[VB],Lower bound is bigger than upper bound in the start of association " + aName + " (" + a.Property("Id") + ").,[#8a]")
                numberOfTotalErrors += 1
            End If
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint8b(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim associations = dictionary.Items("Association")
    Do While associations.MoreItems
        a = associations.NextItem
        Dim aName = a.Property("Name")
        Dim endMultiplicity = a.Property("EndMultiplicityUML")
        If (Regex.IsMatch(endMultiplicity, "(-)?[0-9]+\.{2}(-)?[0-9]+")) Then
            Dim splitMultiplicity = endMultiplicity.Split(New String() {".."}, StringSplitOptions.None)
            Dim lowerBound = splitMultiplicity(0)
            Dim upperBound = splitMultiplicity(1)
            If (lowerBound > upperBound) Then
                errorBuilder.AppendLine("[VB],Lower bound is bigger than upper bound in the end of association " + aName + " (" + a.Property("Id") + ").,[#8b]")
                numberOfTotalErrors += 1
            End If
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint9a(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim associations = dictionary.Items("Association")
    Do While associations.MoreItems
        a = associations.NextItem
        Dim aName = a.Property("Name")
        Dim startMultiplicity = a.Property("StartMultiplicityUML")
        If (Regex.IsMatch(startMultiplicity, "(-)?[0-9]+\.{2}(-)?[0-9]+")) Then
            Dim splitMultiplicity = startMultiplicity.Split(New String() {".."}, StringSplitOptions.None)
            Dim upperBound = splitMultiplicity(1)
            If (upperBound <= 0) Then
                errorBuilder.AppendLine("[VB],Upper bound in the start of association " + aName + " (" + a.Property("Id") + ") must be a positive integer.,[#9a]")
                numberOfTotalErrors += 1
            End If
        End If
    Loop
    Return errorBuilder.ToString
End Function

Private Function CheckConstraint9b(dictionary As Object)
    Dim errorBuilder As New StringBuilder
    Dim a
    Dim associations = dictionary.Items("Association")
    Do While associations.MoreItems
        a = associations.NextItem
        Dim aName = a.Property("Name")
        Dim endMultiplicity = a.Property("EndMultiplicityUML")
        If (Regex.IsMatch(endMultiplicity, "(-)?[0-9]+\.{2}(-)?[0-9]+")) Then
            Dim splitMultiplicity = endMultiplicity.Split(New String() {".."}, StringSplitOptions.None)
            Dim upperBound = splitMultiplicity(1)
            If (upperBound <= 0) Then
                errorBuilder.AppendLine("[VB],Upper bound in the end of association " + aName + " (" + a.Property("Id") + ") must be a positive integer.,[#9b]")
                numberOfTotalErrors += 1
            End If
        End If
    Loop
    Return errorBuilder.ToString
End Function
```

Listing 5. Evaluation constraints implemented in Visual Basic

References

[1] Barnes, J.: High integrity Ada: the SPARK approach. Addison-Wesley Professional (1997)

[2] 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)

[3] Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language. Morgan Kaufmann (2014)

[4] GmbH, F.E.S.T.: Enabler Administration, Release 7.0 Service Pack 1 (2006)

[5] IBM: IBM - rational rhapsody family. Online (2017), http://www-03.ibm.com/software/products/en/ratirhapfami

[6] IBM: Rational doors. Online (2017), http://www-03.ibm.com/software/products/en/ratidoor

[7] No Magic Inc.: MagicDraw. Online (2017), https://www.nomagic.com/products/magicdraw

[8] 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 (2006)

[9] 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: Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009). pp. 1–4 (2009)

[10] Ogunyomi, B., Rose, L.M., Kolovos, D.S.: Property Access Traces for Source Incremental Model-to-Text Transformation, pp. 187–202. Springer International Publishing, Cham (2015)

[11] Project, T.J.: Jawin - a java/win32 interoperability project. Online (2005), http://jawinproject.sourceforge.net/

[12] PTC: PTC Integrity Modeller. Online (2017), http://www.ptc.com/model-based-systems-engineering/integrity-modeler

[13] Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education (2008)

[14] The MathWorks, I.: Simulation and model-based design. Online, https://www.mathworks.com/products/simulink.html