Towards Seamless Hybrid Graphical–Textual Modelling for UML and Profiles

Lorenzo Addazi[1], Federico Ciccozzi[1(✉)], Philip Langer[2], and Ernesto Posse[3]

[1] School of Innovation, Design and Engineering,
Mälardalen University, Västerås, Sweden
{lorenzo.addazi,federico.ciccozzi}@mdh.se
[2] EclipsSource, Wien, Austria
planger@eclipsesource.com
[3] Zeligsoft, Ottawa, Canada
eposse@zeligsoft.com

Abstract. Domain-specific modelling languages, in particular those described in terms of UML profiles, use graphical notations to maximise human understanding and facilitate communication among stakeholders. Nevertheless, textual notations are preferred for specific purposes, due to the nature of a specific domain, or for personal preference. The mutually exclusive use of graphical or textual modelling is not sufficient for the development of complex systems developed by large heterogeneous teams. We envision a modern modelling framework supporting seamless hybrid graphical and textual modelling. Such a framework would provide several benefits, among which: flexible separation of concerns, multi-view modelling based on multiple notations, convenient text-based editing operations, and text-based model editing outside the modelling environment, and faster modelling activities.

In this paper we describe our work towards such a framework for UML and profiles. The uniqueness is that both graphical and textual modelling are done on a common persistent model resource, thus dramatically reducing the need for synchronisation among the two notations.

Keywords: Hybrid graphical–textual modelling · Multi-view modelling · UML · Profiles · MARTE · Xtext · Papyrus

1 Introduction

Model-Driven Engineering (MDE) has been largely adopted in industry as a powerful means to effectively tame complexity of software and systems and their development, as shown by empirical research [8]. Domain-Specific Modelling Languages (DSMLs) allow domain experts, who may or may not be software experts, to develop complex functions in a more human-centric way than if using traditional programming languages. The Unified Modeling Language (UML) is the de-facto standard in industry [8], the most widely used architectural description language [11], and an ISO/IEC (19505-1:2012) standard. UML is general-purpose, but it provides powerful profiling mechanisms to constrain and extend

A. Anjorin and H. Espinoza (Eds.): ECMFA 2017, LNCS 10376, pp. 20–33, 2017.
DOI: 10.1007/978-3-319-61482-3_2

the language to achieve UML-based DSMLs (hereafter simply 'UML profiles'); in this paper we focus on them. Domain-specific modelling demands high level of customisation of MDE tools, typically involving combinations and extensions of DSMLs as well as customisations of the modelling tools for their respective development domains and contexts. In addition, tools are expected to provide multiple modelling means, e.g. textual and graphical, to satisfy the requirements set by development phases, different stakeholder roles, and application domains.

Nevertheless, support for graphical and textual modelling, two *complementary* modelling notations, is mostly provided in a mutual exclusive manner. Most off-the-shelf UML modeling tools, such as IBM Rational Software Architect [18] or SparxSystems Enterprise Architect [4], focus on graphical editing features and do not allow for seamless graphical–textual editing. This mutual exclusion suffices the needs of developing small scale applications with only few stakeholder types. For larger systems, with heterogeneous components and entailing different domain-specific aspects and different types of stakeholders, mutual exclusion is too restrictive and void many of the MDE benefits. When adopting MDE in large-scale industrial projects, efficient team support is crucial. Therefore, modelling tools need to enable different stakeholders to work on overlapping parts of the models using different modelling notations (i.e., graphical and textual).

Establishing a seamless modelling environment, which allows stakeholders to freely choose and switch between graphical and textual notations, can greatly contribute to increase productivity as well as decrease costs and time to market. Consequently, such an environment is expected to support both graphical and textual modelling in parallel as well as properly manage synchronisation to ensure consistency among the two. The possibility to leverage on different perspectives of the same information always in sync can also boost communication among different stakeholders, who can freely select their preferred visualisation means. A hybrid modelling environment for seamless graphical and textual modelling would disclose the following benefits.

Flexible separation of concerns with multi-view modelling based on multiple notations. The possibility to provide graphical and textual modelling editors for different aspects and sub-parts (even overlapping) of a DSML enables the definition of concern-specific views characterised by either graphical or textual modelling (or both). These views can interact with each other and are tailored to the needs of their intended stakeholders.

Faster modelling tasks. The seamless combination of graphical and textual modelling is expected to reduce modelling time and effort thanks to two factors.

(1) The single developer can choose the notation that better fits her needs, personal preference, or the purpose of her current modelling task. While structural model details can be faster to describe using graph-based entities, complex algorithmic model behaviours are usually easier and faster to describe using text (e.g., Java-like action languages).
(2) Text-based editing operations on graphical models, such as copy&paste and regex search&replace, syntax highlighting, code completion, quick fixes,

cross referencing, recovery of corrupted artefacts, text-based differencing and merging for versioning and configuration, are just few of the features offered by modern textual editors. These would correspond to very complex operations if performed through graphical editors; thereby, most of them are currently not available for graphics. Seamless hybrid modelling would enable the use of these features on graphical models through their textual view. These would dramatically simplify complex model changes; an example could be restructuring of a hierarchical state-machine by moving the insides of a hierarchical state. This is a demanding re-modelling task in terms of time and effort if done at graphical level, but it becomes a matter of a few clicks (copy&paste) if done at textual level.

Decoupling of modelling activities and modelling environment. Models can be edited using any text editor even outside the modelling environment.

In this paper we describe our work on providing a framework able to provide seamless hybrid graphical–textual modelling for UML profiles. The uniqueness of our framework resides in the fact that, differently from current practices, both graphical and textual editors operate on a common underlying model resource, rather than on separate resources, thus heavily reducing the need for synchronisation between the two. Our solutions are built upon open-source platforms and with open-source technologies.

The remainder of the paper is organised as follows. Section 2 provides a snapshot of the states of the art and practice related to hybrid modelling. In Sect. 3 we outline our framework, the intended benefits, and the differences with current practices. Details on the actual solution and exemplifications on a UML profile are provided in Sect. 4. The paper is concluded with evaluation results, in Sect. 5, and an outlook on current and future work, in Sect. 6.

2 States of the Art and Practice

mbeddr [14] is an open-source tool which supports extensions of the C language tailored for the embedded domain. The tool focuses on enabling higher-level domain-specific abstractions to be seamlessly integrated into C through modular language extensions. While mbeddr tackles the very relevant issue of bridging abstraction gaps between modelling and target languages, it does not address the seamless integration of different concrete syntaxes exploiting the same level of abstraction. Umple [22] merges the concepts of programming and modelling by adding modelling abstractions directly into programming languages and provides features for actively performing model edits on both textual and graphical concrete syntaxes. The support for synchronisation is limited, thus prohibiting certain kinds of modification at graphical level.

A plethora of other open-source tools such as FXDiagram [7], LightUML [10], TextUML [21], MetaUML [15], PlantUML [17] focus on textual concrete syntax for actively editing the modelling artefacts, while providing a graphical notation for visualisation purposes only. FXDiagram is based on JavaFX 2 and provides

on the fly graphical visualisation of actions done through the textual concrete syntax including change propagation; the focus is on EMF models. LightUML focuses more on reverse engineering by generating a class diagram representation of existing Java classes and packages. TextUML is similar to FXDiagram in the sense that it allows modellers to leverage a textual notation for defining models, in this case UML, and providing textual comparison, live graphical visualisation of the model in terms of class diagrams, syntax highlighting, and instant validation. MetaUML is a MetaPost library for creating UML diagrams through a textual concrete syntax and it supports a number of read-only diagrams, such as class, package, activity, state machine, use case, and component. Similarly, PlantUML allows the modelling of UML diagrams by using a textual notation; graphical visualisations are read-only and exportable in various graphical formats. None of these tools provides means for synchronised editing in both textual and graphical notations nor the possibility to allow customisation of the related concrete syntaxes. Besides FXDiagram, which is DSML independent, the others focus on specific DSMLs, hence providing fixed textual and graphical concrete syntaxes for the considered DSML.

Several research efforts have been directed to mixing textual and graphical modelling. In [1], the authors provide an approach for defining combined textual and graphical DSMLs based on the AToM3 tool. Starting from a metamodel definition, different diagram types can be assigned to different parts of the metamodel. A graphical concrete syntax is assigned by default, while a textual one can be given by providing triple graph grammar rules to map it to the specific metamodel portion. The aim of this approach is similar to ours, but it targets specific DSMLs defined through AToM3 and is not applicable to UML profiles. Charfi et al. [3] explore the possibilities to define a single concrete syntax supporting both graphical and textual notations. Their work is very specific to the modelling of UML actions and has a much narrower scope than our work. In [19], the authors provide the needed steps for embedding generated EMF-based textual model editors into graphical editors defined in terms of GMF. That approach provides pop-up boxes to textually edit elements of graphical models rather than allowing seamless editing of the entire model using a chosen syntax. The focus of that paper is on the integration of editors based on EMF, while ours is to provide seamless textual and graphical modelling for UML profiles. Moreover, the change propagation mechanisms proposed by the authors are on-demand triggered by modeller's commit, while we focus on on-the-fly change propagation across the modelling views. Related to the switching between graphical and textual syntaxes, the approaches in [6,23] propose two attempts at integrating Grammarware and Modelware. Grammarware is a tool by which a mixed model is exported as text. Modelware is a tool by which a model containing graphical and textual content is transformed into a fully graphical model. Transformation from mixed models to either text or graphics is on-demand.

Projective editing is another way to enable different editing views for the same models, as provided by mbeddr and MelanEE [2]. Concrete syntaxes are not stored, only the abstract syntax is persistent. Thereby, the modeller edits

the abstract syntax directly, and then selects specific concrete syntax projections of it. The main benefit is the possibility to project the model in various concrete syntaxes depending on the modeller. On the other hand, it complicates modelling activities, since it requires to act directly on the abstract syntax of the model through editors that are much more complex than parser-based text editors. Jetbrains MPS [9], on which mbeddr is based, provides a projective approach similar to MelanEE. Similarly to our approach, Jetbrains MPS uses a single abstract syntax, but it does not entail "real" text editors, rather providing text-like form-based editors, which hinders the use of traditional text-based tool features (e.g. for regex search&replace, diff/merge for versioning).

To summarise, current solutions for mixed textual and graphical modelling present at least one of the following limitations:

- one of the notations is read-only, intended as a mere visualisation means;
- one of the two notations is enforced for a specific, self-contained portion of a DSML only;
- concrete syntaxes are predefined and not customisable;
- synchronisation among different concrete syntaxes is not automated and on-the-fly but rather manual or on-demand.

3 A Hybrid Modelling Framework Based on Xtext and Papyrus

The goal of our work was to provide a hybrid modelling framework for UML and profiles based on de-facto standard open-source tools, i.e. Eclipse Modeling Framework [5] (EMF) as platform, Papyrus [16] for UML (graphical) modelling, and Xtext [24] for textual modelling. In Fig. 1, we depict the differences between existing solutions for hybrid modelling and our framework.

Existing approaches, notably the one by Maro et al. [12], tackle the provision of hybridness by keeping graphical and textual modelling fully detached. Graphical and textual modelling are performed on two separate models, which are separately persistent in two physical resources. Given a UML profile, a corresponding Ecore-based DSML representing the profile is automatically generated

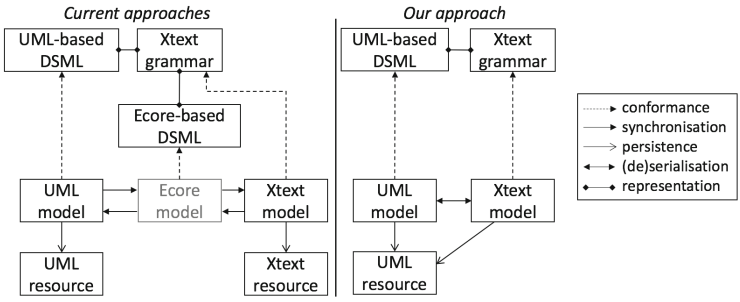

Fig. 1. Current approaches compared to our approach

or manually provided. EMF provides automation for this task, but the resulting Ecore model needs often manual tuning in order to be made usable. Graphical modelling is performed using the UML editors and the model persists as UML model resource. On the other hand, textual modelling is performed using generated Xtext editors and the textual representation persists as an Xtext resource. Moreover, Xtext works internally with an Ecore model resource, which is kept in sync with the textual resource by Xtext itself.

In order to keep graphical and textual models in sync, semi-automated mechanisms in the form of synchronisation model transformations are provided. These model transformations are in fact also generated, thanks to higher-order model transformations (HOTs); this provides a certain degree of flexibility in terms of evolution of the UML profile and automatic co-evolution of the synchronisation mechanisms. Nevertheless, HOTs would not work in case the generated Xtext grammar is customised. This practice is very often needed in order to make the grammar (and related editors) fit the developer's needs.

As a concrete example of the need to customize a DSML grammar, consider the UML-RT language [20]. UML-RT has two core concepts: *capsules* and *protocols*. Capsules are active classes and have a well-defined interface consisting of *ports* typed by protocols. Capsules may have an internal structure consisting of *parts* that hold capsule instances linked by connectors bound to the corresponding capsule ports. All interaction between capsule instances takes place by message-passing through connected ports.

UML-RT has a UML profile. If we start from the UML-RT profile, we obtain an Xtext grammar that contains rules like these:

```
1  Capsule returns Capsule:
2      'Capsule'
3      '{'
4          'base_Class' base_Class=[uml::Class|EString]
5      '}';
6
7  Class returns uml::Class:
8      Class_Impl | Activity | Stereotype | ProtocolStateMachine | StateMachine_Impl
9      | FunctionBehavior | OpaqueBehavior_Impl | Device | Node_Impl
10     | ExecutionEnvironment | Interaction | AssociationClass | Component;
11
12 Class_Impl returns uml::Class:
13     'Class'
14     '{'
15         ('name' name=String0)?
16         ('visibility' visibility=VisibilityKind)?
17         'isLeaf' isLeaf=Boolean
18 ...
19         ('useCase' '(' useCase+=[uml::UseCase|EString]
20             ( "," useCase+=[uml::UseCase|EString])* ')' )?
21 ...
22         ('ownedAttribute' '{' ownedAttribute+=Property
23             ( "," ownedAttribute+=Property)* '}' )?
24         ('ownedConnector' '{' ownedConnector+=Connector
25             ( "," ownedConnector+=Connector)* '}' )?
26 ...
27     '}';
```

This clearly entails a great amount of information related to UML but not relevant to UML-RT. In fact, the rule for `Class_Impl` includes clauses for each and every feature of the UML Class metaclass, many of which we removed for

the sake of space. Of these clauses, many, such as `useCase`, are irrelevant to the DSML, and only a few, such as `ownedAttribute` and `ownedConnector`, are relevant, but they do not reflect the concepts of UML-RT, and even the concrete syntax may not be desirable. For UML-RT, we would like to obtain a grammar with rules that reflect the DSML's concepts directly and hides away any additional UML structure that may be used to represent the concept. For example, instead of having a single clause `ownedAttribute`, we would like to have clauses for ports and parts, in a rule like this:

```
1  Capsule returns Capsule:
2      'capsule' name=EString
3      '{'
4          (ports+=RTPort)*
5          (parts+=CapsulePart)*
6          (connectors+=Connector)*
7          StructuredTypeCommonCoreFragment
8          BehaviourFragment
9      '}';
```

Xtext is designed for being used with EMF-based modeling languages. The UML implementation in Eclipse is EMF-based and thus Xtext can be used to implement textual concrete syntaxes for UML. However, Xtext is not designed to work with UML profiles. This raises the need for explicit complex synchronisation between the two, both at abstract and concrete syntax level. We provide a different approach to make Xtext work with UML profiles (right-hand side of Fig. 1), by exploiting a single underlying abstract syntax (UML-based DSML), two concrete syntaxes (graphical given by UML and textual given by Xtext), one single persistent resource (UML resource), and thereby reducing the need for ad-hoc heavyweight synchronisation mechanisms. Synchronisation is instead performed by Xtext in terms of serialisation and de-serialisation operations between the UML model and the Xtext model, in the same way as Xtext naturally does between the Ecore model and the Xtext model. Our solution provides the following improvements to the current state of the practice:

- **Grammar customisability.** The Xtext grammar can be customised and refactored to fit the developer's needs. This does not jeopardise the (de-)serialisation mechanisms as long as it does not break the conformance of models to the UML profile specification (i.e., metamodel).
- **Cross-profile hybridness.** Virtually, any UML profile can be leveraged without the provision of ad-hoc complex synchronisation transformations. In practice, for complex profiles, (de-)serialisation might need additional input from the hybrid DSML developer (e.g., stereotypes application transformation described in Sect. 4.2).
- **On-the-fly changes propagation.** Model changes done in one view (e.g., UML graphical) are seamlessly reflected and visible on-the-fly in the other view (e.g., Xtext textual); existing synchronisation mechanisms propagate changes on-demand following a specific request from the developer.
- **Cross-notation multi-view modelling.** Different Xtext grammars and editors representing different sub-sets (even partially overlapping) of the UML profile (or several profiles) can seamlessly work on the same UML resource,

along with UML editors. Also in this case, the precondition is that the Xtext grammars enforce model conformance to the entailed profiles.

Other indirect benefits stem from the aforementioned ones. An example is the fact that code generators can reuse a single, shared abstract syntax for both graphical and textual representations of a model, without relying on additional transformations which result in added maintenance costs. Another example is that different stakeholders can view and edit model parts of their collaborators in their preferred syntax (or in a syntax that is optimised for them). In this way, potential inconsistencies can be identified very early already during the modeling process and communication among different stakeholders is greatly improved.

In the next section we describe our hybrid modelling solution from a technical perspective, providing concrete exemplifications of the aforementioned benefits.

4 Technical Solution

Our hybrid graphical–textual UML modelling framework is achieved by combining Papyrus for UML and Xtext. Existing approaches combining UML modelling and Xtext, mentioned in Sect. 2, rely on two completely separated sets of abstract syntax, concrete syntax, and persistent resources. This results in separate graphical and textual modelling, where partial hybridness is achieved by explicit synchronisation between the two concrete syntaxes. Synchronisation is complicated by the fact that graphical and textual abstract syntaxes are separated. Complex exogenous DSML-specific model transformations are needed to realise it. In our solution we provide a more flexible hybrid solution, based on one single abstract syntax (UML-based DSML only, instead of UML-based DSML for graphical and Ecore-based DSML for textual, in Fig. 1), two separated concrete syntaxes (UML model and Xtext model in Fig. 1, needed to overcome limitations of projective approaches), and one single persistent resource.

One major challenge of providing such a solution is that the resource management in Xtext entails the creation and maintenance of a separated Xtext-specific resource. We provide a solution for making Xtext work on the same UML resource as Papyrus, by acting on how the content of the Xtext textual editor is retrieved from and pushed to its underlying resource. Since we are interested in UML profiles, another major challenge is represented by expressing UML stereotypes and their applications in Xtext grammars since there is no concept in Xtext that corresponds to profiling. We solved this challenge by providing a way to define alternative rules, following a superclass/subclass relationship pattern, which enables editing of both stereotype-specific and base UML element properties.

In the next sections we describe in detail how we tackled the two challenges.

4.1 Extending Xtext Resource Management

Xtext does not provide out of the box support for the direct manipulation, including persistence, of UML resources. Xtext models are in fact stored as Xtext

resources as plain textual artefacts and managed by the so called `XtextResource`, which is an Xtext-specific implementation of the EMF `resource`. Serialisation and de-serialisation of textual models to and from in-memory Ecore models are managed by dedicated serialiser and parser, which are automatically generated from the related Xtext grammar. Defining an Xtext-based textual language for UML (or any UML profile) causes Xtext to change the default resource associated to the ".uml" file extension from `UMLResource` to `XtextResource`. Intuitively, this change affects all editors in the modelling environment working on UML resources, such as those provided by Papyrus. As soon as an Xtext textual editor is created for files with extension ".uml", UML models would be stored as plain text, hence not manageable by Papyrus model editors.

In order to solve this issue, we reversed the dependency relationship imposed on other editors by Xtext. More specifically, we enhanced the Xtext textual editor content management so to enable its interaction with UML resources too. In practice, the Xtext textual editor relies on a dedicated provider class to access the resource underlying a model, i.e. `DocumentProvider`. When a UML model is opened using an Xtext textual editor, the enhanced `DocumentProvider` retrieves the content of the associated UML resource, serialises it, and populates the textual editor with it. Analogously, each time a textual model is saved in the Xtext textual editor, the enhanced `DocumentProvider` propagates the applied changes to the underlying UML resource by first parsing the editor's content and then building or modifying the UML model to be stored.

4.2 Modelling UML Stereotypes Application in Xtext

Xtext does not provide out of the box support for UML profiles. In order to enable Xtext-based textual languages and related editors to feature UML profiles and stereotypes application, we operated on the way Xtext creates and maintains grammars and parsed models.

Given a grammar specification, Xtext creates a corresponding metamodel defined in Ecore, which we call "grammar metamodel", describing the structure of the grammar's abstract syntax tree. This metamodel can be imported in case the grammar relates to an existing grammar metamodel. Parsing of textual models conforming to an Xtext grammar is stored in-memory in terms of the so called grammar model, which conforms to the grammar metamodel.

Let us walk through the steps to provide support for UML profiles and stereotypes application in Xtext-based textual languages. Below, we depict an excerpt of the Xtext grammar providing a textual language, *MarText*, for the UML profile for MARTE [13].

```
1   import "http://www.eclipse.org/uml2/5.0.0/UML" as uml
2   generate marText "http://www.eclipse.org/papyrus/uml/marte/MarText"
3
4   Model returns uml::Model :
5           'model' {uml::Model} name=ID ('{'
6                   packagedElement+=Component*
7           '}')? ';'
8
9   Component returns uml::Component :
10          HwProcessor | HwCache |
11          'component' {uml::Component} name=ID ('{'
12                  packagedElement+=Component*
13          '}')? ';'
14
15  HwProcessor returns HwProcessor :
16          'processor' {HwProcessor} name=ID ('{'
17                  ('cores:' nbCores=INT ';')? &
18                  ('caches:' '{'
19                          packagedElement+=HwCache*
20                  '};')?
21          '}')? ';'
22
23  HwCache returns HwCache :
24          'cache' {HwCache} name=ID ('{'
25                  'level:' level=INT ';'
26          '}')? ';'
```

First, we import the UML metamodel as baseline for the Xtext grammar to access UML metaclasses during the definition of the grammar rules (line 1 of the MarText grammar). For each stereotype in the profile, we define a dedicated grammar rule for enabling the textual editing of stereotype properties (e.g., `HwProcessor` stereotype rule at line 15 of the MarText grammar).

While enabling the editing of stereotype properties, we still need to offer the possibility to edit the properties of the base UML element to which the stereotype can be applied. To do so, we first looked at how multiple alternatives for a given grammar rule are represented in the grammar metamodel[1]. Given a rule A, with rules B and C as alternatives, Xtext defines three corresponding metaclasses such that A is a superclass of B and C. We leverage this superclass/subclass relationship pattern by defining a stereotype-specific rule as alternative to the rule for the base UML element to which the stereotype can be applied (e.g., `Component` is superclass of `HwProcessor` and `HwCache`, in lines 9–10 of the MarText grammar). The developer can thereby access both stereotype-specific and base UML element properties as with Papyrus UML model editors.

In order to propagate stereotypes application among the two notations, we acted on how `DocumentProvider` retrieves and stores contents of the UML resource. We defined an endogenous in-place model transformation, which maps the application of stereotypes to UML base elements by following the superclass/subclass relationship pattern mentioned above and based on the MARTE profile metamodel definition. Going from textual to graphical, the transformation navigates the Xtext model and sets stereotypes to base UML elements in the UML resource accordingly. An example depicted in Fig. 2 is represented by `processor` processorA in the textual model, which leads to the application of the

[1] The interested reader can refer to the Xtext specification [24] for further details about the overall inference process.

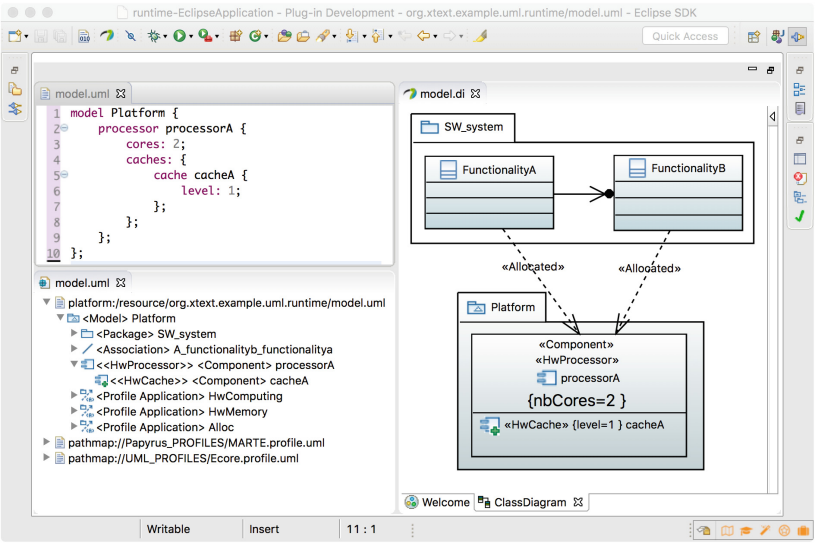

Fig. 2. MarText (top-left), Papyrus tree-based (bottom-left), and Papyrus graphical (right) editors in Eclipse.

stereotypes «`Component`»«`HwProcessor`» to processorA in the graphical model. Going from graphical to textual, the transformation navigates the UML resource and reproduces the stereotyped element, without explicitly reporting base UML element info, in the textual format. An example depicted in Fig. 2 is represented by «`Component`»«`HwProcessor`» processorA in the graphical model, which leads to the definition of processorA as `processor` in the textual model.

5 Evaluation and Discussion

In Sect. 3, we listed a set of four improvements to current practices brought by our framework. We provided them as follows.

Grammar customisability and cross-profile hybridness. The framework works on a manually edited and customised Xtext grammar for MARTE. Moreover, the solution does not entail complex profile-specific synchronisation transformations between textual and graphical notations. The only transformation needed, for propagating stereotypes application across the notations, is generalisable since based on the superclass/subclass relationship pattern between base UML elements and applicable stereotypes. That is to say, the mechanism itself is cross-profile, while a profile-specific instance of it, as the one we used for MarText, can be generated by a specific profile metamodel definition, in some cases with the help of the hybrid DSML developer.

On-the-fly changes propagation. Model changes done in one view are seamlessly reflected and visible in the other views (graphical, textual and tree-

based views in Fig. 2). To appreciate how changes are propagated on-the-fly, the reader can refer to the movie at http://www.mrtc.mdh.se/HybridModelling/demo_movie.zip.

Cross-notation multi-view modelling. We showed how an Xtext-based textual language (MarText), with related grammar and editor, representing only a sub-set of the `HwLogical` package of MARTE can seamlessly work on a UML resource containing other UML and MARTE concepts (e.g., UML elements in `SW_system` package and MARTE «`allocated`» relations in Fig. 2). For instance, MarText would be suitable for a platform modeller, who might not need or want to view functional details. This is possible thanks to our enhanced Xtext resource management, which, instead of overwriting the in-memory model with plain text, propagates changes directly to the UML resource.

Additionally, we made an experiment to compare modelling times using the different notations in four scenarios: *Create 1*, *Modify 1*, *Create 2*, and *Modify 2*. *Create 1* and *Modify 1* are run on the platform package depicted in Fig. 2. In *Create 1* we model the platform package, and in *Modify 1* we add an additional `HwCache` cacheB element to the model and assigned to processorA. *Create 2* and *Modify 2* are represented by modelling and modifying (renaming all states from 'state_**x**' to '**x**') a UML state-machine composed of 6 states (1 initial, 1 final, 1 join, 3 normal states) and 5 transitions among the states. The modelling tasks were performed individually by a set of developers, with similar experience in UML modelling with Papyrus and Xtext-based textual languages. All developers got a 2-hours preparation time to study the Xtext languages for MarText and UML state-machines[2]. Table 1 shows the experiment results. We provide the arithmetic mean of the individual sets of values.

Table 1. Mean times for performing the tasks in minutes

Notation	Modelling task				
	Create 1	*Modify 1*	*Create 2*	*Modify 2*	Total
Graphical	1.06	0.27	**0.52**	0.18	2.03
Tree-based	0.46	0.23	2.15	0.22	3.06
Textual	**0.24**	**0.08**	1.42	**0.09**	1.83
Hybrid	(0.24)	(0.08)	(0.52)	(0.09)	**0.93**

Textual editing results faster when creating stereotyped elements and setting their properties (*Create 1*). This is due to the possibility to customise Xtext grammars to only require a minimum amount of information to be entered by the modeller (while the underlying base UML elements are created by our stereotypes application transformation). The same goes for the modification of an existing model by inserting a new model element (*Modify 1*).

[2] The Xtext language for state-machines is not in the scope of this paper and was created for experimental purposes only.

The creation of state-machines resulted to be faster with the graphical notation (*Create 2*). This is mainly due to a swifter creation of transitions between states using the graphical view. Transition modelling is also the reason why the tree-based notation resulted way worse than the other two in this scenario (a much higher amount of "clicks" is needed for creating transitions). The textual notation resulted to be faster than the others when renaming model elements, as expected. This is due to textual regex search&replace; while for a bigger model the times for renaming elements would linearly increase if done through tree-based or graphical notations, for the textual notation this is not the case, since regex search&replace would not be affected in the same way by a higher number of hits. Looking at the total modelling times, we can see how combining graphical and textual notations (column 'Total' row 'Hybrid' in Table 1) allows to get the most out of them, resulting faster than all the others.

6 Outlook

In this paper we outlined the initial steps towards a hybrid seamless graphical–textual modelling framework for UML profiles based on Papyrus and Xtext. The uniqueness of our framework is that both graphical and textual modelling act on a single common persistent model resource, thus requiring lower synchronisation effort than current approaches. By seamlessly combining graphical and textual modelling, the framework can mitigate the drawbacks of both as well as emphasise and combine their benefits. We showed several of them, such as flexible separation of concerns, multi-view modelling based on multiple notations, convenient text-based editing operations, and faster modelling activities.

We are currently working on a façade-based approach for improving the encapsulation and reusability of the support for profiles. The idea is to provide means for defining profile-specific custom implementations of complex stereotype elements (as in the case shown in the paper regarding UML-RT), which create and maintain the UML "boilerplate elements" behind them. In the presented solution we only provide limited support for this. Moreover, we are working on a parametric automated generation of Xtext grammars from UML profiles in order to support the creation of hybrid DSMLs.

Acknowledgements. We would like to thank Simon Redding, Francis Bordeleau, and Matthias Tichy for the fruitful discussions and support. This work is partially supported by the Papyrus Industry Consortium(https://wiki.polarsys.org/Papyrus_IC), the EUREKA network Hybrid Modeling project(http://www.eurekanetwork.org/project/id/10700), and the KK-foundation MOMENTUM project(http://www.es.mdh.se/projects/458-MOMENTUM).

References

1. Pérez Andrés, F., De Lara, J., Guerra, E.: Domain specific languages with graphical and textual views. In: Schürr, A., Nagl, M., Zündorf, A. (eds.) AGTIVE 2007. LNCS, vol. 5088, pp. 82–97. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89020-1_7

2. Atkinson, C., Gerbig, R.: Harmonizing textual and graphical visualizations of domain specific models. In: Proceedings of the Second Workshop on Graphical Modeling Language Development, pp. 32–41. ACM (2013)
3. Charfi, A., Schmidt, A., Spriestersbach, A.: A hybrid graphical and textual notation and editor for UML actions. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 237–252. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02674-4_17
4. SparxSystems Enterprise Architect. http://www.sparxsystems.eu/enterprise architect/. Accessed 17 Feb 2017
5. Eclipse Modeling Framework. https://www.eclipse.org/modeling/emf/. Accessed 17 Feb 2017
6. Engelen, L., van den Brand, M.: Integrating textual and graphical modelling languages. Electron. Notes Theor. Comput. Sci. **253**(7), 105–120 (2010)
7. FXDiagram. http://jankoehnlein.github.io/FXDiagram/. Accessed 17 Feb 2017
8. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: 2011 33rd International Conference on Software Engineering (ICSE), pp. 471–480. IEEE (2011)
9. Jetbrains MPS. https://www.jetbrains.com/mps/. Accessed 17 Feb 2017
10. LightUML. http://lightuml.sourceforge.net/. Accessed 17 Feb 2017
11. Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A.: What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. **39**(6), 869–891 (2013)
12. Maro, S., Steghöfer, J.P., Anjorin, A., Tichy, M., Gelin, L.: On integrating graphical and textual editors for a UML profile based domain specific language: an industrial experience. In: Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2015, pp. 1–12. ACM, New York (2015). http://doi.acm.org/10.1145/2814251.2814253
13. UML profile for MARTE. http://www.omg.org/spec/MARTE/. Accessed 17 Feb 2017
14. mbeddr. http://mbeddr.com/. Accessed 17 Feb 2017
15. MetaUML. https://github.com/ogheorghies/MetaUML. Accessed 17 Feb 2017
16. Papyrus. https://eclipse.org/papyrus/. Accessed 17 Feb 2017
17. PlantUML. http://plantuml.com/. Accessed 17 Feb 2017
18. IBM Rational Software Architect. http://www-03.ibm.com/software/products/en/ratsadesigner/. Accessed 17 Feb 2017
19. Scheidgen, M.: Textual modelling embedded into graphical modelling. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 153–168. Springer, Heidelberg (2008). doi:10.1007/978-3-540-69100-6_11
20. Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object Oriented Modeling. Wiley & Sons, Chichester (1994)
21. TextUML. http://abstratt.github.io/textuml/. Accessed 17 Feb 2017
22. Umple. http://cruise.eecs.uottawa.ca/umple/. Accessed 17 Feb 2017
23. Wimmer, M., Kramler, G.: Bridging grammarware and modelware. In: Bruel, J.-M. (ed.) MODELS 2005. LNCS, vol. 3844, pp. 159–168. Springer, Heidelberg (2006). doi:10.1007/11663430_17
24. Xtext. http://www.eclipse.org/Xtext/. Accessed 17 Feb 2017