Engineering Hybrid Graphical-Textual Languages with Sirius and Xtext: Requirements and Challenges

Justin C. Cooper
Department of Computer Science
University of York
York, UK
justin.cooper@york.ac.uk

Dimitris Kolovos
Department of Computer Science
University of York
York, UK
dimitris.kolovos@york.ac.uk

***Abstract*—Embedding textual domain specific languages into graphical modelling workbenches can help deliver the best of both worlds. In this paper, we discuss common requirements for hybrid textual/graphical modelling workbenches, and we present requirements, existing approaches and open challenges for integrating graphical editors implemented using the state-of-the-art Sirius graphical modelling framework, and textual editors developed with Xtext within the Eclipse Modeling ecosystem.**

***Index Terms*—Hybrid Graphical Textual Modelling, Xtext, Sirius, EMF, Refactoring, Code Generation**

I. Introduction

The Eclipse Modelling Framework (EMF) is one of the most widely used open source modelling frameworks, and allows for the creation of domain specific modelling languages (DSLs). For a DSL to be used effectively, the language must have an appropriate concrete syntax. Several frameworks, built atop of EMF, allow for textual and graphical concrete syntaxes to be designed and implemented. These frameworks allow a language designer to precisely define the language syntaxes and create workbenches and editors for the language based on the Eclipse IDE.

Graphical, diagram-based, concrete syntaxes are generally suitable for defining structure and relationships between model elements. However defining fine detail, such as precise behaviour, can often be tedious and error prone to capture and maintain graphically. For such detail (e.g. action languages), textual syntaxes are generally more appropriate. By having a modelling workbench which supports both textual and graphical syntaxes, the language designer and users of the modelling workbench can experience the best-of-both worlds of textual and graphical modelling.

This paper summarises previous work on integrating graphical and textual syntaxes, based on the Sirius and Xtext EMF-based frameworks. Both frameworks are state-of-the-art, robust, actively maintained and widely used. We note that the integration of the two frameworks is still in its early stages, with the tooling having several limitations such as: a lack of refactoring support, difficulties of dealing with the embedded models and the required knowledge of lower-level implementation details of EMF, Xtext and Sirius. We present that, by building on previous work, and addressing the current tooling limitations, the technique of embedding textual languages into graphical workbenches may help to close the gap between models and code implementations.

The paper is organised as follows. In Section II, we provide background into the frameworks used to create the workbench and motivate the need for embedding textual DSLs into graphical modelling workbenches. In Section III, requirements are presented for hybrid textual-graphical tooling. Section IV provides an overview of related work on integrating the Sirius and Xtext frameworks. Section V presents the open challenges integrating Xtext and Sirius. Section VI concludes the paper and provides areas for future work.

II. Background and Motivation

In this section we briefly introduce Ecore, Xtext and Sirius, which are used in this work to create the abstract and concrete syntax of DSLs and associated tooling. We then motivate why a modelling workbench supporting graphical and textual DSLs is often needed.

A. EMF Ecore

As part of the Eclipse Modelling Framework, Ecore is an object-oriented metamodelling language that enables the definition of the abstract syntax (metamodel) for a language using concepts such as EClass, EAttributes, EOperations, EDataTypes and EReferences.

B. Xtext

Xtext is a framework which allows for the creation of textual syntaxes for Ecore metamodels. By defining a metamodel using Ecore and Xtext's EBNF-based notation, Xtext automatically generates all of the textual concrete syntax infrastructure, such as a linker and parser. Xtext also generates an Eclipse editor including developer assistance features such as syntax highlighting, code completion and error detection. Xtext is highly customisable as it utilises the Dependency Injection pattern, allowing features such as the syntax highlighting or refactoring behaviour to be easily customised.

C. Sirius

Sirius is a framework, based upon the Eclipse Graphical Modelling Framework (GMF), that allows a language designer to specify a graphical representation for an Ecore based

metamodel. Sirius provides a Viewpoint Specification Model (VSM), which describes the concrete syntax of an Ecore model. The VSM allows diagrams (node-edge and sequence-diagram-like), tables, matrices, trees to be created with minimal knowledge of the internals of its underlying GMF, as well as being able to create custom properties views and use Java to implement custom services.

D. Motivation

Hybrid graphical-textual editors have advantages such as the faster modelling of tasks (textual languages can reduce the number of clicks when creating and editing models, but graphical languages have been shown to reduce the time of linking model elements together) and the ability to edit models outside of a modelling environment [1].

In this line of work, we are interested in languages which are predominately graphical, but which would benefit from an embedded textual sub-language to define complex expressions or behaviour. This is a pattern we have encountered on several occasions in interactions with industrial collaborators, particularly where full code generation is required. Examples of such languages include:

- A state-machine language where states and transitions are specified graphically, but guards in transitions and actions in states are specified using a textual expression sub-language
- An object-oriented modelling language where classes, structural features and associations/inheritance are modelled graphically, but the body of operations is specified textually

III. Requirements for Hybrid Graphical-Textual Modelling Workbenches

A simple solution for supporting such scenarios is to model textual expressions as plain text properties that can be typed into the "Properties" view of a Sirius-based editor in a generic text box. The main shortcomings of such a basic solution are that users are provided with little guidance and feedback on the validity of the content of these textual properties, and that model management programs (e.g. model-to-text transformations, model-to-model transformations) need to parse the values of such properties explicitly to extract and navigate their abstract syntax graphs. To address these shortcomings we envision a solution that deeply integrates Sirius and Xtext to meet the following requirements.

A. Syntax-Aware Text Editing

Textual languages require a large number of features for a developer to use effectively. These include mechanisms found in many programming integrated development environments (IDEs) such as Eclipse, Intellij and Visual Studio [2]. Notable features include:

- Syntax Highlighting – keywords, variables, language features can be identified visually;
- Error detection markers – feedback on syntactic or semantic errors is provided to the user whilst typing;
- Auto-completion – syntax and context-aware completion of statements, such as providing a suggestion for the variable name the developer wants to use;
- Refactoring – e.g. renaming elements and their references consistently in an atomic operation

Such assistance mechanisms for a textual language are crucial for improving accuracy, reducing intensive reliance on manual checking of typed text and thus increasing the overall productivity of the developer [2].

B. References Between the Textual and Graphical Parts

In order for complex behaviour to be defined, the textual language must be able to reference elements that are modelled graphically elsewhere in the model.

C. Uniform error reporting

The modelling tool must report errors and warnings in both the graphical and the textual part of the model in a consistent and uniform way (e.g. in the IDE's "Problems" view), and - where possible - provide quick fix actions that can resolve these problems.

D. Integrated Abstract Syntax Graph

In order for the the model to be transformed into other artefacts such as code or other models, the tool should expose hybrid textual/graphical models to model management programs (e.g. model-to-model, model-to-text transformation engines) in the form of a unified abstract syntax graph that integrates elements from both the textual and the graphical part of the model.

IV. Related Work

In this section, we discuss work that Obeo, Typefox [3] and Altran [4] have previously conducted on integrating Sirius and Xtext to create hybrid model editors.

In [3], Obeo and Typefox have provided two case studies integrating Xtext with Sirius. Obeo and Typefox show that it is possible to synchronise graphical and textual concrete syntaxes for the same EMF model. The benefits of textual and graphical modelling are presented using a farming case study, whereby it is possible to specify complex behaviour such as cultivating corn graphically, but is much easier to define textually.

The problem with integrating textual and graphical modelling in the way presented by [3] is that models can then get very complicated and polluted with large amounts of detail, making the diagram very difficult to understand. When creating system implementations, it may be more appropriate to use textual DSLs and graphical modelling languages in a complementary way, that being, to specify high-level elements graphically in one modelling language, and more low-level details using a complementary, separate textual language.

In [3], a second example using Xtext and Sirius is presented, but in contrast to the first approach, two separate DSLs are created. A computer systems DSL is defined using Ecore and Sirius, but a separate textual DSL (defined in Xtext), used to describe constraints on the components of the computer

system. This textual DSL can reference elements in the Computer DSL by referring to the Ecore metamodel of the Computer DSL. The Sirius editor is then extended to provide an embedded textual editor with support for some textual language assistant mechanisms (such as syntax highlighting). The code created in the Xtext editor is then persisted in a textual attribute within model elements of the graphical language.

As an Xtext editor is embedded in the Sirius editor, the textual DSL code can be written with the assistance of syntax highlighting and code completion. The user can also reference and navigate to elements in the Sirius diagram. This approach allows for graphical elements to be referenced and also avoids large amounts of detail being present in the graphical model, which is a major drawback of the approach presented in [3]. This work has since been extended by Altran [4] to allow embedded editors to be implemented in Sirius diagrams as well as the Eclipse Properties view.

Figure 1 shows a workbench created by adapting the case study in [3]. Note that "1" shows the Sirius diagram, "2" shows the Eclipse problems view (displays any errors in the model) and "3" shows the embedded Xtext editor in the "Properties" view, showing error detection, referencing to the graphical model, syntax highlighting and code completion.

Although we are focusing on creating domain specific languages with Sirius and Xtext, Xtext has been used to create hybrid graphical-textual languages with Papyrus in [1], for a hybrid-textual graphical UML (Unified Modelling Language) editor. Xtext has also been used in environments outside of Eclipse, such as in [5], where Theia (a desktop and web IDE) and Sprotty (a graphical visualisation framework) are used to create a hybrid graphical-textual domain specific modelling tool.

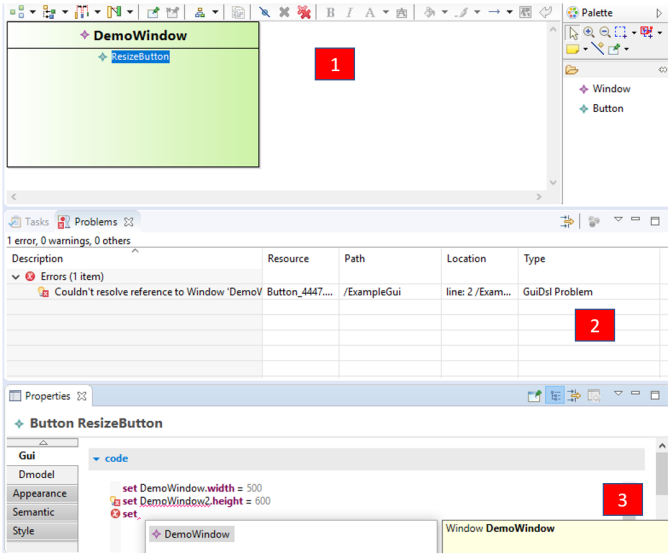

Fig. 1. Sirius Modelling Workbench with Embedded Xtext Editor

V. Open Challenges

Although previous work carried out by Obeo, Typefox [3] and Altran [4] has shown it is possible to embed a textual DSL created using Xtext into a graphical modelling workbench created using Sirius, there are still several open challenges preventing this tool from meeting the requirements set out in III. In this section, we briefly introduce and detail the open challenges.

A. Storage of textual DSL code

The current work stores the textual model data within a string attribute in the model. Storing the Xtext DSL code in a string attribute in the model has several disadvantages. One problem is when dealing with operations such as rename refactoring and the checking of errors. The model must be queried for all of the textual DSL, even if the DSL code contains no references to the object being refactored, and loaded into memory to ensure that all references are identified.

Figure 2 shows the metamodel of the graphical modelling language (represented by Sirius). Note that any textual DSL code must be stored as a string attribute such as guiDSL ("1") in Figure 2. "2" shows a derived attribute, a potential solution to the challenge described in subsection V-E.

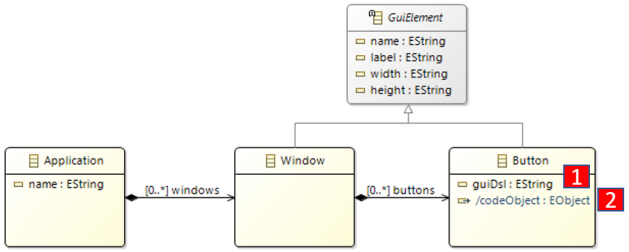

Fig. 2. Ecore metamodel to store textual DSL code

Another potential issue is model comparison. If the model is serialised as XMI and changes are made to the model, when comparing models, a user would see the textual DSL code within XML tags. This would mean there would be no formatting or syntax highlighting for the user trying to view changes to the textual DSL code. If other model elements have changed as well as the textual DSL code, this can make comparison and merging the code difficult.

An alternative to storing the textual DSL as a string attribute in the metamodel, could be to persist the textual DSL as separate files on the file system and store a link to the file in the model.

By storing the text in a file on the file system, this allows the Xtext framework to automatically identify and provide markers in the Eclipse Problems view without having to write any additional logic. Another advantage of persisting the textual DSL in a separate file rather than a string attribute is that the textual DSL can be edited in an environment outside of the embedded editor in Sirius. The embedded editor in Sirius does

not provide features such as an Outline view, meaning that if there was a large amount of code in the model, it may be more appropriate for a user to edit this code in a full-page editor rather than the embedded editor. If the DSL code is persisted in a file on its own, this solves this issue of comparison and merging as it is no different from the comparison and merging of normal textual code.

One major drawback to this approach is the model must keep a reference to the file located in the Eclipse workspace. This reference may become broken if files/model elements are renamed (depending on the implementation of the linking) and care needs to be taken when deleting model elements that the files are also removed.

B. Scoping and referencing

Only global scoping is currently supported in the solutions provided in the previous work i.e. every model element represented in Sirius and every model element represented in the Xtext model (such as variable declarations) are all in scope to every other element. Xtext does provide functionality to create custom scoping rules for the textual DSL, but this may lead to several complex scoping rules needing to be created which are specific to individual Xtext DSLs.

C. Rename refactoring

Rename refactoring is challenging for several reasons. Firstly the embedded Xtext editor is not an Eclipse editor, it is a custom widget which is not easy to extend. For example, there is no simple mechanism for attaching handlers to it.

Xtext allows for the customisation of rename refactoring to be completed by providing a RenameStategy interface. Another challenge is that the primary rename refactoring engine relies on the Xtext DSL persisting on the file system. Currently the Xtext DSL is stored within a textual attribute in the model represented by Sirius. The ease of implementing rename refactoring is determined by which mechanism the Xtext DSL code is persisted in the model.

In the case of the textual DSL using references to files on disk, this is fairly straightforward and the refactoring engine provided by Xtext can be augmented with the custom rename strategy to allow the refactoring of the Sirius model.

In the case of the textual DSL being persisted in a string attribute in the model, each model element containing code must be iterated through to find the textual DSL code, creating an in-memory model of the textual DSL code and then persisting at least some code on the file system. These approaches need to ensure that all temporary files are cleared.

D. Displaying of error/warning markers

The solutions in the previous work display error/warning markers if the user has the embedded Xtext editor open. If a user closes the embedded editor with errors, these errors are no longer visible. Similarly, if an error occurs in the Xtext DSL code, without the embedded editor being open, for example a referenced model element being deleted, a user would not know an error exists unless they opened all the model elements containing the Xtext DSL code and then inspected the highlighted errors.

The problem with displaying error/warning markers is closely related to the storage of the textual DSL code. As noted in V-A, if the code was stored in separate files on the file system, the Xtext framework will automatically check if there are any errors and display these in the Eclipse problems view without any additional configuration or logic being written. This can be seen in Figure 1 whereby the DSL code has been copied into a file and the Xtext framework (builder) has detected the issue and reported it in the Problems view.

Alternatively, if the code was to be persisted in a string attribute of a model element, logic must be written to iterate through each model element containing code and create appropriate markers.

E. Accessibility of textual model

There is currently no mechanism to easily access the textual DSL model. This is necessary for performing operations such as code generation.

One solution is to parse the textual DSL code stored in the string attribute and assign this to a derived reference in the model. When accessing the reference as part of a model management operation, such as a model-to-text transformation, the respective sub-model will be returned, allowing it to be queried and used as part of the transformation.

VI. Conclusion

In this paper we have motivated the need for a modelling workbench supporting hybrid graphical-textual concrete syntaxes. We have discussed how such a modelling workbench can be used to close the gap between models and code by embedding textual action languages into a graphical model editor. We then defined the requirements for a modelling workbench to allow action languages to be embedded into models. We note that previous work has been carried out into integrating Sirius and Xtext, however there are open challenges that must be solved before this technique can be used to close the gap between models and code.

We are currently working on resolving the challenges discussed in this paper and creating suitable case studies to test whether this approach can effectively close the gap between models and code.

References

[1] L. Addazi, F. Ciccozzi, P. Langer, and E. Posse, "Towards seamless hybrid graphicaltextual modelling for uml and profiles," 06 2017, pp. 20–33.

[2] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," vol. 38, no. 1, pp. 5–18.

[3] Xtext sirius integration - white paper. [Online]. Available: https://www.obeodesigner.com/resource/white-paper/WhitePaper_XtextSirius_EN.pdf

[4] Xtext sirius integration. [Online]. Available: https://altran-mde.github.io/xtext-sirius-integration.io/

[5] A domain-specific language ide in the cloud using open-source only. [Online]. Available: https://github.com/TypeFox/theia-xtext-sprotty-example