Extensible Graphical Editors for Palladio

Misha Strittmatter[1], Michael Junker[1], Kiana Rostami[1],
Sebastian Lehrig[2], Amine Kechaou[1], Bo Liu[1], Robert Heinrich[1]

[1] Karlsruhe Institute of Technology (KIT)
{strittmatter | rostami | heinrich}@kit.edu
{michael.junker | amine.kechaou | bo.liu}@student.kit.edu

[2] Paderborn University
sebastian.lehrig@uni-paderborn.de

Abstract

Palladio is an approach to design and performance prediction of software architectures. An important part of the Palladio's tooling—the Palladio Bench—are its graphical GMF-based editors. In contrast to rudimentary tree-based editors, they enable a more intuitive creation of models even for less experienced developers. However, the maintenance of the GMF-based editors has become cumbersome because the requirement arose to support an increasing amount of new language features.

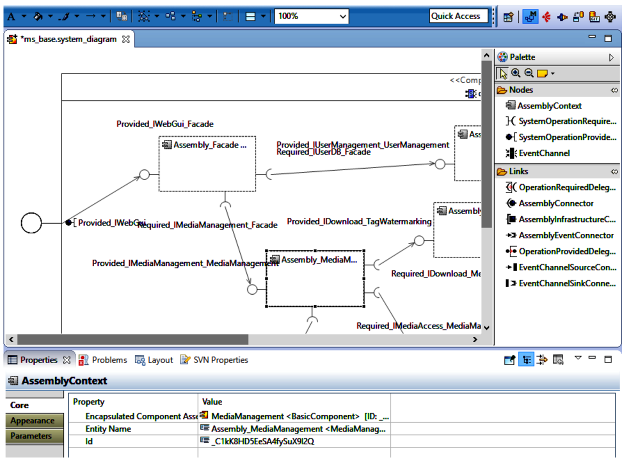
In this paper, we present the new generation of graphical editors for Palladio, which are based on the Sirius editor framework. Further, we present a concept of how to develop external extensions to the graphical language, which can be plugged into the new editors without the need to intrusively modify them.

1 Introduction

At the heart of the Palladio Approach [11] is the Palladio Component Model (PCM). The PCM is a metamodel, which defines a language for the modeling of component-based software architectures. Further, it supports modeling of abstractions of the software architecture's quality with a focus on performance. For better usability, there are graphical editors for the PCM, which enable the user to develop diagrams visually. These were developed with the GMF editor framework and come as part of Palladio's tooling—the Palladio Bench.

However, as the PCM evolved over time, extensions to the PCM were created. New features were implemented intrusively in the GMF editors, which led to a growth of the editor specification or a branched development. This was detrimental to the specifications' long-term maintainability. In addition, new frameworks arose that aspired to become GMF's successor (like Graphiti and Sirius). These innovations have several advantages over GMF.

We reimplemented the graphical Palladio editors using the Sirius framework. In this paper, we discuss the advantages of these new editors and their platform for users as well as developers (see Section 2). This is a continuation of our work graphical editors for Palladio [4] as well as modular, extensible tooling [3]. In this paper, especially of interest is the capability of Sirius-based editors to be extended by editor fragments (see Section 3). This type of extension aligns with our research about extensible and modular metamodels [9]. In Section 4, two exemplary editor extensions to the Palladio Sirius editors are presented. Section 5 concludes the paper.



AssemblyContext

Property	Value
Encapsulated Component Asse	MediaManagement <BasicComponent> [ID: _...
Entity Name	Assembly_MediaManagement <MediaManag...
Id	_C1kK8HD5EeSA4fySuX9I2Q

Figure 1: The GMF-based System Editor

2 Benefits of the new Editors

The new Sirius-based PCM editors are a reimplementation of the GMF-based editors. With exception of some features that are currently unsupported by simulators, the new editors cover all of the features of the old editors and more. The new features include modeling of data types in the repository, architectural templates [6] and rudimentary profile [2] support. Further new user-beneficial features result from the usage of Sirius. These include layer-based hiding and unhiding of diagram content, improved layouting and the customization of the look of graphical nodes.

Even unsaved model changes are now kept in sync over multiple diagrams. Saving changes will no longer discard all unsaved changes in other diagrams.

Implementing the editors with Sirius has many advantages for the development. Similar to GMF, a DSL is used to specify the editors. However, with Sirius, no code is generated from the specification, but it is interpreted. This solves the problem of maintaining manual changes in the code when the editors are regenerated. In Sirius, the inflexibility that is caused by not having generated code to modify, is alleviated by providing extension points and the ability to extend the specification afterwards. In contrast to an API-driven editor framework like Graphiti [1], the DSL-based approach greatly reduces development effort although it may be a little less flexible.

Besides a lively community, a huge benefit of development with Sirius is the support for dynamic evaluation. An editor can be tested and used in the same Eclipse instance as it is developed in. After saving, changes to the editor specification immediately come into effect in the editor. An exception to this are External Java Actions, which are deposited in separate plugins and have to be loaded in the Eclipse platform. In contrast to GMF this is a great benefit because in GMF it is necessary to first generate the editor code and then start a new Eclipse to test the changes.

Another benefit is that Sirius supports editor extension. An editor specification can be referenced by another editor specification, which can be located in another plug-in. The extending editor specification can then add new layers, which may contain new graphical elements or even alter the visuals and behavior of existing ones. Being able to extend the graphical language that is implemented by the Sirius Editors enables us to support independent language extension. The next section will present the potential of the extensibility of Sirius and relate types of metamodel extensions to types of editor extensions.

3 External Editor Extension

Sirius offers the possibility to extend a diagram representation by further layers without altering the implementation of the base diagram intrusively. Analogously to metamodel extensions [2, 5], editor extensions are also packaged in their own Eclipse plugins. This is shown in Figure 2. The basic editor should not know anything of its extensions. In addition, editor extensions should not know each other except when an extension extends another one. Although the figure shows only one extension, the plugins of a metamodel and all of its extensions form a directed acyclic graph. The editor and its extensions mirror this structure.

In the remainder of this section, a brief explanation is given on how different types of metamodel extensions are mapped to editor extension types and how the editor extensions are implemented with sirius. Detailed information as well as guidelines can be found in the master thesis of Michael Junker [10].

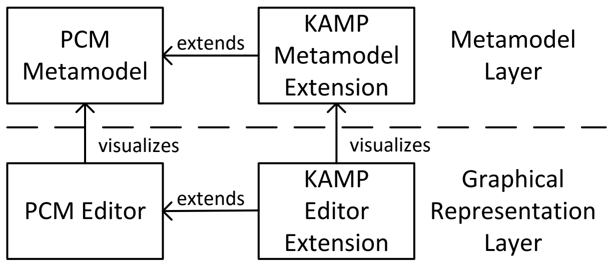


Figure 2: Plugin Dependencies of the Base Metamodel and Editor with Exemplary Extensions

Using the *diagram extension* feature of Sirius, a new layer can be added to an existing editor. Within this layer, new notation elements can be added and existing ones can be altered. The extension model (i.e., the instance of the metamodel extension) needs to be added to the current diagram in order to work with it. This is automatically accomplished for every model that exists in the same modeling project as the base metamodel instance loaded. For external model files, a toolbar button can be implemented loading the file and adding the model resource to the current Sirius session.

If a new metaclass is added by a metamodel extension, this class can be represented as a new notation element in the diagram extension. As the instances of such new metaclasses are stored in external models, they can be retrieved using *Java services*. When extending the palette or context menu, it is necessary to take into account that the extended instances have to be stored externally. In case the metamodel extension adds a further attribute or containment to existing metaclasses in the base metamodel, the import mechanism of Sirius can be used to import an existing notation element. Furthermore, bordered nodes can also be added to imported containers as well as to nodes. It is also possible to change the style of an existing element, e.g., if a new Boolean attribute is added. This is done by adding conditional styles to notation elements. Since the Sirius 4.0 release in Eclipse Neon, extensions of the properties view are also possible from within the viewpoint specification. A last type of extension concerns references into the metamodel extension or from the extension to the base metamodel. Similar to adding a metaclass, references can be represented as notation elements (in most cases as connections).

4 Exemplary Editor Extensions

In the following, we present two extension of the new Sirius editors. The first extension presents an editor to support usage model extension for business processes, while in the second one we show how to extend the repository and system editor to visualize change impact propagation.

4.1 Business Process

The Integrated Business Information Technology Impact Simulation (IntBIIS) [7] is a Palladio extension, which enables the users to model the interaction of business processes and software systems. Business processes are modeled as an extension of UsageModel (i.e., BPUsageModel). A business process can be modeled as a set of hierarchically nested activities. An activity can in turn be an actor step, that an actor performs manually, or a system step, that is done automatically. EntryLevelSystemCalls represent the system steps in business processes. Furthermore, the use of device resources in the business process can also be modeled. A device resource is a device needed to perform an actor step such as a forklift. Using loops and branches the activities can be nested [7].

The new elements of the BPUsageModel should also be available in a business process Layer of the editors of usage model. To this end, we created a new Sirius editor, in which we defined a new viewpoint for BPUsagemodel. As the BPUsageModel is an extension of the UsageModel, we have to import elements from the UsageModel. This way, the new elements of the BPUsageModel can be used in the UsageModel.

4.2 Karlsruhe Architectural Maintainability Prediction (KAMP)

To predict the ripple effect of a change request in a software system, the Palladio add-on Karlsruhe Architectural Maintainability Prediction (KAMP) [8] can be used. To this end, the user of KAMP should mark the initial change request based on a Palladio model. The output of KAMP is a list of all artifacts (e.g., code or test cases), that are potentially affected by the change. The task list contains all components, interfaces, data types, provided role, required roles, signatures, and assembly connectors that the user should take into account when implementing the change. However, it is more convenient for the user, if she can observe the changing artifacts in a graphical Palladio editor (e.g., by changing the color to red).

Using the diagram extension mechanism, we extend the editors of the repository and the system model to reflect the change in the editor of the modificationmarks model [8]. With the help of Java services, the modificationmarks model can be accessed. We import all changing elements (e.g., components or data types) in a new layer. Using conditional styles, we can define a new style, that highlights the elements which are affected by the change request.

5 Conclusion

In this paper, we presented the advantages of the new graphical Sirius-based editors for Palladio. The improved maintainability and ability to develop plugin-like extensions will lead to earlier graphical editor support for new features. The ability to externally extend editors will prevent the core editors from degrading and by that ensures long-term maintainability. To demonstrate the external extensibility, we presented two exemplary extensions: modeling of business processes, which interface with the Palladio usage models (IntBIIS) and visualization of change propagation prediction (KAMP).

However, a lot of manual effort was put into the GMF-based editors to polish their usability. The new Sirius-based Palladio editors have not yet reached that level of maturity. Therefore, the upcoming Palladio 5 release will include both, the new and the old editors. More information about the new editors can be found in our wiki[1].

Acknowledgments

This work was partially funded by the Helmholtz Association of German Research Centers. We'd like to thank Richard Rhinelander for his valuable input.

References

[1] C. Brand et al. "Development of High-Quality Graphical Model Editors". In: *Eclipse Magazine* (2011).

[2] M. E. Kramer et al. "Extending the Palladio Component Model using Profiles and Stereotypes". In: *Palladio Days 2012 Proceedings (appeared as technical report)*. KIT, Faculty of Informatics, 2012, pp. 7–15.

[3] M. Strittmatter et al. "Towards a Modular Palladio Component Model". In: *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days*. Vol. 1083. CEUR Workshop Proceedings, 2013, pp. 49–58.

[4] C. Stritzke and S. Lehrig. "Why and How We Should Use Graphiti to Implement PCM Editors". In: *Symposium on Software Performance*. Vol. 1083. CEUR, 2013, pp. 1–10.

[5] R. Jung et al. "A Method for Aspect-oriented Meta-Model Evolution". In: *Proceedings of the 2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling*. ACM, 2014, 19:19–19:22.

[6] S. Lehrig. "The Architectural Template Method: Design-Time Engineering of SaaS Applications". In: *PhD Session at the Advanced School on Service Oriented Computing*. 2014.

[7] R. Heinrich et al. "Integrating business process simulation and information system simulation for performance prediction". In: *Software & Systems Modeling* (2015), pp. 1–21.

[8] K. Rostami et al. "Architecture-based Assessment and Planning of Change Requests". In: *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015, pp. 21–30.

[9] M. Strittmatter et al. "A Modular Reference Structure for Component-based Architecture Description Languages". In: *Proceedings of ModComp*. CEUR, 2015, pp. 36–41.

[10] M. Junker. "Flexible Graphical Editors for Extensible Modular Meta Models". MA thesis. Karlsruhe Institute of Technology (KIT), 2016.

[11] R. H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. to appear. MIT Press, 2016. 408 pp.

[1]`https://sdqweb.ipd.kit.edu/wiki/PCM_Development/Sirius_Editors`