

Unicase – an Ecosystem for Unified Software Engineering Research Tools

Bernd Bruegge, Oliver Creighton, Jonas Helming, Maximilian Kögel
Siemens Corporate Technology & Technische Universität München
icgse08@creighton.de, {bruegge, helming, koegel}@in.tum.de

Abstract

Many research approaches aiming at control and mitigation of risks in global software development (GSD) are based on tool support. Following a rigorous research approach these tools need to be evaluated and therefore implemented. Existing tools lack support for research requirements. As a consequence researchers often have to build their own solution from scratch. This is a time consuming task associated with the risk of failure. This paper proposes a platform called unicase. The goal of unicase is to support researchers in building tools for evaluation of research approaches in GSD. Unicase is based on Eclipse technology and helps the researcher in the evaluation of tool-supported approaches to GSD including approaches to risk management.

1. Introduction

A lot of research in Global Software Engineering (GSD) has been carried out in tools supporting GSD projects to control the additional risk associated with distributed project set-ups. The scope of these tools is often visualization (e.g. [10], [11], [15], [16]) and collaborative modification (e.g. [9], [10], [14]) of a custom set of artifacts. In typical software engineering projects these artifacts are stored separately in various repositories including databases, file systems or software configuration management systems. This includes tools for requirements engineering, UML tools for the design, project management tools, developer tools and many more. In this paper we focus on two core issues in using commercial tools for research in GSD projects.

(1) Limited domain model

Innovative tool support for GSD is often based on the introduction of new element types into a domain (e.g. [10], [14]). Commercial tools are limited to a predefined model from their domain and usually not geared towards extensibility. Furthermore Traceability between specific artifacts from different domains, e.g. between risks and system models [6], is a precondition

for a variety of research approaches in GSD. A lot of effort has been spent in connecting tools to achieve the required traceability [13]. “The absence of standards has been shown to be a barrier to integration, as various tool developers remain unable to reach agreement on the appropriate point(s) in this space at which integration should occur. As a result, experience with tool integration has been largely at a tool-to-tool level, with little use of standard tool integration mechanisms.” [1]

(2) Limited Extensibility

Commercial tools often do not provide open interfaces and well-documented APIs. Many research approaches in GSD are based on the automatic gathering of data to apply empirical metrics. This requires a variety of data to be collected from multiple data sources [5]. These data sources, e.g. a task repository often, have to be instrumented by the researcher. Other research approaches require the enhancement of the existing functionality of a tool, e.g. to provide decision support to the project participants.

The above mentioned two core issues and their implications often force the researcher to implement a research prototype from scratch, which is tailored to the scope of their research approach (e.g. [9], [10], [11], [14], [15], [16]). The requirements of these proprietary research implementations often overlap. For example a central and collaboratively accessible repository for the used artifacts of a tool is often required due to the distributed character of GSD. SCM capabilities of this repository allow recreating every state of a project for analyzing purposes.

The reimplementing of these features with the usual time and cost constraints often results in poor reliability and usability. As a consequence this often leads to a lack of motivation for the project participants to use it and results may be distorted. A high quality of the implementation is also an inevitable prerequisite for an industrial case study.

This paper presents the requirements for a uniform platform for research in Global Software Engineering called “unicase”. Unicase addresses the core requirements for tools in GSD projects to lower the effort and risk for researchers to implement custom

tool solutions. The core idea of the project is adopted from the Eclipse ecosystem. The goal is to share the effort to implement basic requirements among interested parties. In the following we give an overview over related work, the requirements for such a platform. We describe the ecosystem idea and present an exploratory prototype. Finally we discuss future work.

2. Related work

In this section we describe related tools and approaches and show how they are different from our approach.

In general we found that any commercial, close-source solution we looked at does not provide the necessary extensibility for most research topics. This finding is not very surprising since they were not built for this purpose. They only provide very limited support for enhancing their model. Additionally if any they only provide a small set of extension points to add custom behavior. Furthermore cost for licensing or other legal implication is sometimes prohibitive for use in research. In particular we looked at the IBM Rational Software Architect, IBM RequisitePro, Telelogic Doors, Borland Caliber, Microsoft Team Foundation Server and IBM Jazz.

On the other hand the available open-source and/or academic tools have very similar deficiencies. We looked at a range of tools and we will go into detail for a typical representative of every category we identified. To give a short summary there are two categories of these tools, the special purpose research tools and the meta-CASE or integration tools.

The special purpose research tools are geared towards a specific research topic and have been designed to very specific requirements. They are not designed for extensibility.

SEURAT [10] is a research prototype for a new approach to rationale management. While the functionality for maintaining rationale is sophisticated, there is no support for collaborative work at all, which is a prerequisite for a case study in this area of research since rationale management is a collaborative activity.

Fujaba [14] is limited to a specific system model only. It provides only limited collaboration support because its Software Configuration Management is text based [7].

The purpose of the Sysiphus project is to provide a uniform platform for research in software engineering. The platform has grown for over 7 years now. It has been the foundation to several research topics and case-studies ([2], [4], [9], [6], [7]), recently in projects with over 50 participants. Stability and Usability became a serious issue, in a survey of 30 users 65%

considered Sysiphus to be a useful tool in the current implementation compared to 100% who considered Sysiphus to be useful if it was more usable and stable. Sysiphus unfortunately does not meet the usability, reliability and scalability requirements for a bigger (industrial) case study.

The available model elements of ArchStudio are again limited. Architectural artifacts and product lines are supported only [15].

The second category of open source and/or academic tools are Meta-CASE or integration tools. In other words they either are CASE Tools to design custom CASE-tools or they try to integrate case tools from different domains.

Marama (formerly Pounamu) [12] is a Meta-CASE tool that allows specifying and generating CASE-tools. Although it supports any kind of model element and provides mechanisms for generating visual editors it does not provide support for collaborative work. The artifacts are persisted to the local file system and it is up to other methods and tools to enable collaboration.

Moflon [13] supports the integration of models from different tools by facilitating traceability links among the models from different tools. However for the integration of n tools it requires a non-polynomial number of transformers and a linear number of adapters to be implemented for full integration. This is prohibitive for any research that spans models from multiple tools.

3. Requirements

We performed exploratory interviews during the SE 2008 and ICSE 2008 conferences among researchers developing tools evaluating various approaches in SE. We also collected qualitative feedback from our industrial partners. This chapter summarizes our initial set of core requirements for a software engineering research platform. Our next step will be quantitative evaluation.

Model Repository

The core requirement is a central repository, which stores artifacts. As they are part of the project model, we will further call these artifacts “model elements” ([4]). The repository is able to store these model elements. The expressiveness of the model is at the level of MOF or an appropriate subset of it, such as ECORE (cite). Adopted from Sysiphus [2] we initially classify the model of unicas in 4 parts (see Figure 1): (1) The requirements model contains model elements like Scenarios or Use Cases [3] and describes the system under construction in the application domain. (2) The system model describes the design using

artifacts like UML ([8]) classes or flow charts. (3) The collaboration model contains management artifacts like tasks as well as rationale models and annotations. The structure of a organization in terms of groups and members is modeled in the (4) organization model. The model is easily extensible. Research approaches often require new model elements or attributes to be added. This is even possible during project run-time.

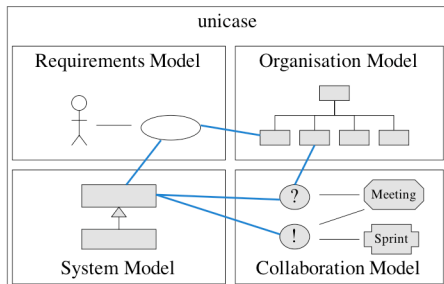


Figure 1: Linked elements from different parts of the model

Traceability

Related model elements in the repository can reference each other (see Figure 1). We call these references model links. Model links are typed. A task for example can be connected to a person with a link of the type “assigned to” expressing that the task is assigned to him. Model links are traceable. Unicase shall support the following two types of traceability [7]: (1) Intra-model traceability allows setting and following links inside a domain model, e.g. an association in a UML class diagram [8]. Inter-model traceability additionally supports links between domain models, e.g. from an issue [9] in the rationale model to a requirement which is the object the issue refers to. Furthermore, we distinguish three different kinds of traceability: (1) Vertical traceability is traveling through different levels of abstraction, e.g. from a requirement to a detailing use case. (2) Horizontal traceability usually follows inter-model links on the same abstraction level. (3) Temporal traceability allows tracing model evolution over time. This requires a model-based SCM described in the next section.

SCM and Distributed Collaboration

A precondition for tool research in GSD is support for distributed collaboration. One of the responsibilities of a SCM system according to [17] is workspace control. That is the ability to work on the same set of artifacts in different workspaces at different locations. Workspace control includes synchronization. Generally two different paradigms for control of concurrent modification of artifacts are well-known, pessimistic

and optimistic concurrency control. Pessimistic concurrency control is often realized by locking and does not perform very well in large-scale distributed settings. An effective approach to workspace control therefore must build on optimistic concurrency control and as a consequence needs mechanisms for resolving conflicts that may arise.

The workspace interaction schema of unicase is checkout, update and commit as in popular source code repositories, such as SubVersion. This provides workspace isolation and users can work without being disturbed by other users’ changes that may be preliminary anyway. Nevertheless the unicase platform additionally supports pushed updates to facilitate real-time collaboration if required.

An SCM system is also essential for many research approaches in terms of data collection.

Model Element Editor

Unicase provides three default views to browse and modify model elements. (1) The tree view allows users to hierarchically browse the whole model (see Figure 2). The hierarchy can be modified by drag and drop.

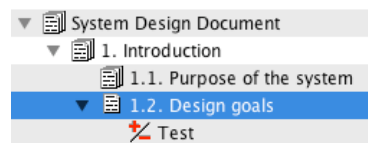


Figure 2: Example for a tree view

(2) The table view provides a tabular access to model elements and offers features like search and filter. (3) The model element editor view displays one model element and allows modifying it. Therefore it provides widgets for all standard attributes including text fields, text areas, checkboxes, date picker, but also widgets to set and modify references. The available widgets are easily extensible to add custom widgets like an impact probability matrix for risk management ([6]). As the underlying model is extensible, the editor is able to display any new model element “out of the box”. Nevertheless the appearance and the arrangement of widgets is adaptable by the use of UI hints.

Graphical Visualization

Many research approaches require model elements to be visualized. Unicase provides a uniform draw pane, which supports visualizing and modifying model elements and links between them. Default visualization is available for any model element. Custom visualization can be added easily. The draw pane provides the features of a state-of-the-art CASE tool diagram editor (see Figure 3). This includes resizing, arranging, coloring and zooming. The editor is

extensible e.g. for algorithms for complex diagram arranging.

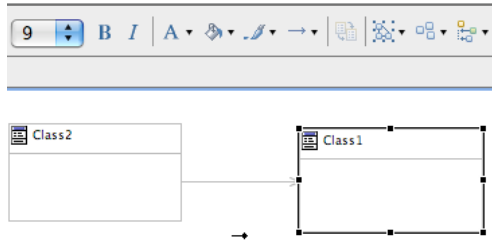


Figure 3: Example for a draw pane

Tool Instrumentation

As a research platform unicas provides comprehensive possibilities of tool instrumentation. The system is able to track when users read and modify elements, as well as which features they use. Custom events can be easily implemented.

Non-Functional Requirements

Unicas is easily extensible. Researchers are able to rapidly implement and integrate features for their research approach. The core system is reliable and is robust against the failure of extensions. Unicas provides a native look and feel as well as a common and natural usability concept. The common part of the framework is licensed under the Eclipse Public License.

4. Ecosystem

The idea of the unicas ecosystem is adopted from the Eclipse ecosystem. Instead of developing standard “infrastructural” software and libraries over and over again in different projects and at different companies, the idea is to collaboratively design, implement and maintain this software, thus significantly reducing the effort for every single participant. By “infrastructural”, standard software we understand software that is not a competitive advantage in industrial terms or that is not part of the core research in academic terms.

We define the unicas platform as such software. The requirements mentioned before are common requirements of Software Engineering Tools we investigated. Many other tools have implemented similar features involving methods, techniques and algorithms that are well known and that have already been published. Implementation is feasible and does not require expert knowledge that is not widely available. So neither from an academic nor from an industrial point of view there is any advantage in restricting access to these.

From an industrial point of view such a platform provides cutting edge research on a stable platform. From an academic point of view a platform that is highly extensible and in use in the industry is a unique opportunity for industrial case studies with minimum threads to validity in terms of case study setup.

The ecosystem is open to anyone. Partners can only participate and use the developed technologies or they can actively contribute to the platform. By actively contributing the involved partners gain high impact on the requirements and the design of the platform.

The developed platform will be released under the Eclipse Public License that allows using any part of the platform in commercial or non-commercial products and even more importantly to develop proprietary possibly secret extensions upon it without the need to publish these extensions or their source code. This provides maximum flexibility for the platform use and re-use.

Currently we have two industrial contributors and one academic contributor apart from our chair. Our goal is to reach out for a larger community of contributors and participants - industrial and academic.

5. Exploratory Prototype

To demonstrate the feasibility of our approach we are currently developing an exploratory prototype. Based on this prototype we will implement an initial core of the system, which features the core requirements described in section 3 and then can be extended and modified by project participants. This section describes the design and implementation of the exploratory prototype.

We chose to build unicas based on the Eclipse Rich Client Platform (RCP). The platform provides several mechanisms to assure extensibility, stability and performance. Furthermore Eclipse provides a native “look and feel” as well as a mature concept for usability, which is widely established in the software development community. Most importantly however Eclipse features many existing frameworks and technologies especially in the areas of modeling and visualization. With its focus on software engineering tasks, it is a perfect match for several requirements of unicas, which otherwise had to be developed. Tools, which are implemented in Eclipse, can easily be integrated in the Eclipse IDE. We conducted a survey among users of the Sysiphus system, which meets comparable requirements to unicas (see section 2). 85% of the participants considered Sysiphus to be more useful if it was integrated in Eclipse IDE. Eclipse offers a mature plug-in concept. The plug-in mechanism makes it very easy to extend unicas by

adding new plug-ins. Furthermore this concept ensures reliability, as the so-called runtime platform is robust against plug-in failure. The exploratory prototype consists of a set of plug-ins described in the following, for additional information about Eclipse technologies see [18]:

EmfStore

The EmfStore plug-in is usually executed in a headless Eclipse environment and provides a repository for instances of the model defined in the model plug-in (see below). The EmfStore persists instances of the model with Teneo. Teneo is an object-relational database-mapping framework based on EMF and Hibernate. The EmfStore also keeps track of all model versions and allows clients to checkout a copy as well as to receive changes of the model. The EmfStore implements a change-based SCM as presented in [7]. Access control is based on a user, group and role model and the EmfStore is responsible for authenticating and authorizing user operations on the model upon checkout, update and commit operations. The EmfStore is capable of handling any model elements that are subclasses of the ModelElement class defined in the Model.

Model

The Model plug-in defines the model for representing artifacts. The model is based on EMF and expressed as an EMF Ecore diagram. From the Ecore diagram we generate the model plug-in code following the EMF model driven development approach. To implement new model elements one can just modify the Ecore diagram using a graphical editor and then regenerate the model. The regenerating the model will neither require changes to the UI nor to the server, thus making the implementation of new models a straightforward task.

The model also features a model instance generator that will generate a full-blown instance of a model with a configurable number of model elements, document width and height and a random seed. The generated model instances are a basis for extensive integration testing.

Model Element Editor

The goal of the generic editor is to meet the requirement “model element editor”, that means basic features for browsing and editing the model. Therefore it provides three views: a tree view (see Figure 2), a table view and a detailed model element editor view (see section 2).

To implement the tree view we used the Common Navigator Framework (CNF) (see Figure 4). This is the

standard component used in Eclipse to browse trees like file systems or packages. To populate and visualize tree and table views we used the emf.edit framework, which is automatically generated for any custom model.

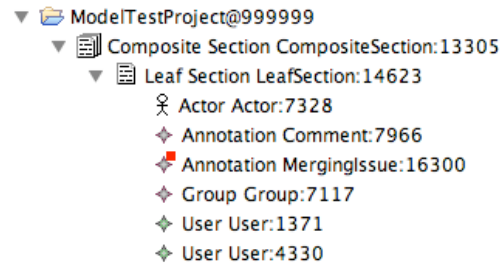


Figure 4: Generated project in the tree view

Existing solutions for model editors have at least one of the following four issues: (1) They are not usable to edit large models (e.g. the properties view), (2) they are not able to modify one element of the model but are build to edit the whole model at once, (3) They are bound to a specific model or require additional information about the desired visualization.

Therefore we implemented a new solution visualized by the Eclipse Forms technology. The editor uses the property descriptors generated in the emf.edit framework to reflectively build up the GUI at runtime. Basically it places a widget on the main form for every attribute of the EMF object using emf.databinding for synchronization between model and UI. Thereby it provides a view to edit a single model element without explicitly knowing its type and without any additional modification by the developer (see Figure 5).

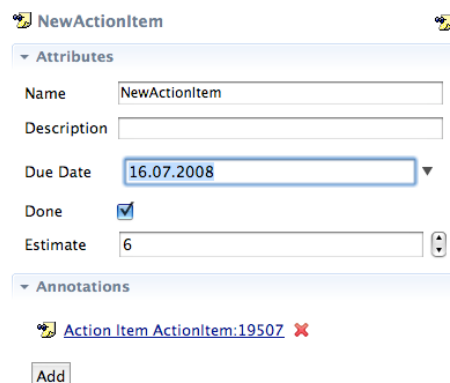


Figure 5: Reflectively generated editor view

Draw Pane

To provide a graphical visualization of the model we used the Graphical Modelling Framework (GMF) (see Figure 3). The purpose of GMF is to map EMF

models to visualizations of the Graphical Editing Framework. This allows to effectively generate custom draw panes. We had to modify the standard GMF editor to use an abstract model source instead of saving diagrams only to files. This was achieved by implementing a custom IResource.

6. Future work

We are currently implementing a base version of unicas. The first release is planned in October 2008. The framework will then be evaluated in student project as well as with our industrial partners. There are two initial research approaches we try to evaluate: (1) What is the impact of changes in the project model and how can changes be propagated to project members effectively? (2) How can the applied Software Lifecycle Model be adopted to mitigate process related risks?

Our next goal will be to allocate new partners for the project both in the academic and the industrial field. Further we plan to conduct a qualitative survey at ICGSE 2008. The target group for the unicas platform includes GSD researchers especially in the field of decision support who require tool support. The survey is meant to enhance and prioritize the requirements set for unicas.

10. References

- [1] Anthony I. Wasserman, *Tool integration in software engineering environments*, Springer, Berlin, 21.1.2006.
- [2] B. Bruegge, A. H. Dutoit, and T. Wolf. *Sisyphus: Enabling in formal collaboration in global software development*. In Proceedings of the First International Conference on Global Software Engineering, October 2006.
- [3] B. Bruegge, *Object-Oriented Software Engineering – Using UML, Patterns and Java*, Pearson, Munich, 2nd edition, 2004.
- [4] T. Wolf. *Rationale-based Unified Software Engineering Model*. Dissertation, Technische Universität München, July 2007.
- [5] F. Shull and R.L. Feldmann, “Building Theories from Multiple Evidence Sources.” *Guide to Advanced Empirical Software Engineering*, 2008, pp. 337-364.
- [6] J. Helming, *Entwicklung eines Werkzeugs für das eingebettete Risikomanagement*. Diploma Thesis, Technische Universität München, Dezember 2006.
- [7] M. Koegel. “Towards Software Configuration Management for Unified Models“. ICSE CVSM’08 Workshop Proceedings, p. 19-24, May 2008, Leipzig.
- [8] OMG Unified Modeling Language Specification Version 2.0, May 2004.
- [9] T. Wolf, A. H. Dutoit, “A Rationale-based Analysis Tool”, 13th International Conference on Intelligent & Adaptive Systems and Software Engineering, Nice, France, July 1-3, 2004.
- [10] J. E. Burge, David C. Brown, *SEURAT: Integrated Rationale Management*, In Proceedings of International Conference on Software Engineering, p. 835 ff, May 2008, Leipzig.
- [11] A. Lucia, R. Oliveto, G. Tortora, *ADAMS Re-Trace: Traceability Link Recovery via Latent Semantic Indexing*, In Proceedings of International Conference on Software Engineering, p. 839 ff, May 2008, Leipzig.
- [12] J. Grundy, J. Hosking, J. Huh and K. Li, *Marama: an Eclipse meta-toolset for generating multi-view environments*, Formal demonstration paper, 2008 IEEE/ACM International Conference on Software Engineering, Leipzig, Germany, May 2008, ACM Press.
- [13] C. Amelunxen, A. Königs, T. Rötschke, A. Schür: *MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations*, in: A. Rensink, J. Warmer (eds.), *Model Driven Architecture - Foundations and Applications: Second European Conference*, Heidelberg: Springer Verlag, 2006; *Lecture Notes in Computer Science (LNCS)*, Vol. 4066, Springer Verlag, 361--375.
- [14] S. Burmester, H. Giese, W. Schäfer, *Model-Driven Architecture for Hard Real-Time Systems: From Platform Independent Models to Code*, Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA’05), Nürnberg.
- [15] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, *An Infrastructure for the Rapid Development of XML-based Architecture Description Languages*, In Proceedings of the 24th International Conference on Software Engineering (ICSE2002), Orlando, Florida.
- [16] T. Reinhard, S. Meier, R. Stoiber C. Cramer, M. Glinz, *Tool Support for the Navigation in Graphical Models*, In Proceedings of International Conference on Software Engineering, p. 823 ff, May 2008, Leipzig.
- [17] J. Estublier, D. Leblang, A. van der Hoek, R. Conradi, G. Clemm, W. Tichy, and D. Wiborg-Weber. *Impact of software engineering research on the practice of software configuration management*. ACM Trans. Software Engineering Methodologies, 14(4):383-430, 2005.
- [18] Eclipse Projects, <http://www.eclipse.org/projects/listofprojects.php>, June 2008