

# Miki: A Wiki for Synchronous Modeling of Software Requirements

Yijun Yu    Marian Petre    Thein Than Tun  
Computing Department  
The Open University  
United Kingdom

## ABSTRACT

Eliciting a high quality requirements model that can be traced down to implementations is a challenge. Keeping models updated for evolving software systems is a further challenge. Formal modelling methods are one approach – but one that is too rigid and costly for many small software engineering companies [1]. We propose a light-weight alternative, using a wiki as the synchronous bridge between requirements capture and more formal modeling features of the IDE such as `Eclipse`.

## 1. INTRODUCTION

Requirements engineers face the challenge of eliciting from stakeholders high quality formal models for reasoning and analysis purposes. In practice, however, requirements are often informally stated, requiring substantial effort to digest. Requirements modeling methods usually demands a rather heavy use of formal modeling tools, usually a big barrier for small software engineering companies to adopt [1].

One way to alleviate this elicitation problem is to interface a full-fledged modeling tool through its plugin with a wiki system. Available to every Web browser, wikis have been applied to software projects for documenting various artifacts ranging from source code, design blueprints, bug reports, mailing lists, etc [7]. Through collective uses of the wikis, a software development team can elicit requirements as informal text in a rather flexible format. Linking to related domain knowledge through wiki names requires little effort. Integrating with change management systems such as subversion is also possible. `Trac`<sup>1</sup> is an example wiki system used by software development teams. It can associate wiki pages with snippets and revisions of the source code. On the other hand, a formal requirements model still needs to be elicited manually from these informal descriptions.

An additional problem is *updating* the links from the for-

<sup>1</sup><http://trac.edgewall.org/>

mally stated requirements down to solutions in the sources of evolving software systems [3]. A loose integration such as the `EclipseTracPlugin`<sup>2</sup> still requires users themselves to upload the models created by heavy `Eclipse` modeling tools into more flexible `Trac` wiki attachments.

This paper demonstrates a technical solution that integrates the modeling tools tightly within a wiki system such that analysts, end-users, other types of stakeholders only need a Web browser to create and maintain modeling diagrams. In addition to the `Trac` wiki format – which is suitable for documenting informal requirements – our extended macro also accepts formal requirements modeling languages of existing `Eclipse`-based tools as the input. From the input, stakeholders can get the formal models in both graphical and textual syntaxes. The synchronously update feature of `Miki` makes it less demanding to bridge formal requirements models and the informal wiki pages.

Our limited experience in the prototype `Miki` system is reported in this paper, with a position that *a wiki enhanced with formal modeling features that could produce easy to maintain models that would help achieve the objective of flexible modeling and sketching of requirements.*

## 2. THE MIKI SYSTEM

Figure 1 overviews the `Miki` system as a service that synchronizes textual and graphical models from a wiki users' perspective. The example illustrated here is that of analysing the *Sluice Gate* problem using the Problem Frames method [6]. Other modeling languages such as *i\** [10] and argumentation [5] are equally supported by our web-based Open Requirements Engineering lab<sup>3</sup>, running as an instance of `Miki`.

Wiki is an effective means to document requirements collaboratively and informally among the group members. To illustrate the effectiveness on the other extreme of the flexible modeling spectrum, i.e. modeling formal requirements, we compare `Miki` with (`Trac` + `Eclipse`) in three operational use cases (creating, updating and deleting). In each of them, the formal requirements models are shown updated in the wiki pages, reflecting users' modifications in a light-weight fashion, in terms of bandwidth, time and effort savings. The requirements engineers in the SecureChange research project use the `Miki` prototype and have confirmed the usefulness in eliciting early requirements: they could share and edit

<sup>2</sup><http://trac-hacks.org/wiki/EclipseTracPlugin>

<sup>3</sup><http://computing-research.open.ac.uk/trac/openre>

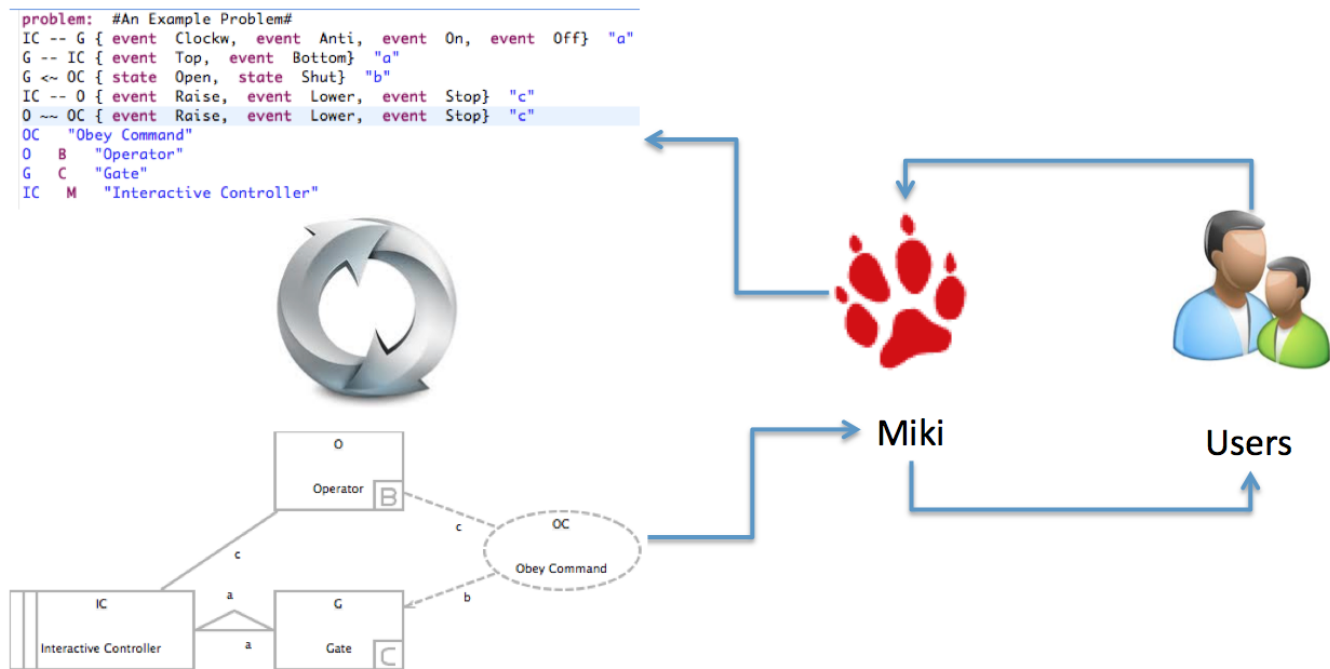


Figure 1: An overview of the Miki system

the requirements model using the bare browsers remotely and document the background knowledge on the same wiki pages; when they are meeting face-to-face in front of a desktop, the advantages of Miki over the Eclipse alternative is the provision and recording of secondary annotations in the free-formed wiki format [4].

The size of the model in textual form (DSL) is not a scalability issue for the tool support, however, visualising the whole model in one page could contain excessive information for users due to the automatic layout of large diagrams. According to our users, this problem can be alleviated by dividing the large textual model into chunks that can be treated as individual diagrams separately, while maintaining the traceability from the original model to these fragmented diagrams. Although all our modeling languages such as goal models and problem frames can be hierarchically used, in general not all requirements languages support this type of hierarchical views.

## 2.1 Creating a model

When a user wants to create a model using a domain-specific language (DSL) through Miki, she starts with entering an extended Trac macro (see Figure 2 for a screenshot):

```

{{{#!openpf
... domain specific language in $ext ...
#!$ext
}}}}

```

where `#!openpf` is the opening pragma for the macro we created in Miki; `$ext` (e.g., instantiate to an extension name such as “problem” in Figure 2) in the closing pragma `#!$ext` is the abbreviation of the DSL, used as the file extension name of the corresponding model; the content of the modeling file is filled by the text in-between the opening and the

closing pragmas. Once submitting the above macro in Miki, a modeling file `$wiki.$ext` will be created into an input folder with `$wiki` being the name of the wiki page embedding the macro. Our customised running Eclipse instance monitors constantly any change inside the input folder to discover and load the model file into the editor generated by the Xtext framework for the DSL `$ext` [2]. On top of standard applications of model-driven software development process [9], we have introduced two major automations for the purpose of synchronised modeling:

The first one converts the modeling text in the generated syntax highlighted editor into a color-coded file `$wiki.$ext.html` in an output folder using the following encoding per token:

```

<font color='‘#RRGGBB’>[<$style>]$token[</$style>]
</font>

```

The hexadecimal RRGGBB colors the `$token` in the RGB scheme, and the optional `$style` is chosen between “B” for bold and “I” for italics, depending on the choices made by the Xtext editor.

The second automation generates an Eclipse modeling framework (EMF) model `$wiki.$ext.xmi` that could be loaded by a corresponding graphical modeling framework (GMF) editor that visualizes the same model graphically. A standard GMF editor plugin normally just visualises a graph in Eclipse, without saving it into a form for the use by an external wiki. Therefore the visualisation must be extended to fulfil the second half of our goal, i.e., to embed the graph as part of the wiki page. This is done automatically through several customizations to the GMF editors.

First we make use of the GUI-free API in the GMF runtime to save the model opened in the GMF plugin into graphical file formats such as `$wiki.$ext.png` and `$wiki.$ext.pdf`, and put them into the output folder. The Miki system constantly monitors the changes inside the output folder until these output files are saved from both the Xtext and GMF editors, then automatically import them as attachments to the `$wiki` page: the HTML output file is automatically inserted into the wiki page after translating the HTML encoding as native ones in the Python Trac extension, and the image output is embedded by the generated Trac macro `[[Image(attachment: $wiki.$ext.png)]]` on the same wiki page.

In brief, Figure 3 presents the architecture of the Miki system, exemplified by the data flow. As a result (see Figure 4), along with the rest of the informal wiki text, users will see both the textual and graphical representations of the formal model. The whole process is automated and takes no more than a few seconds.

## 2.2 Updating a model

Once the model was created and shared on the wiki, to update the model another user may simply edit the Miki macro and resubmit. Upon an update of the Miki page, the system checks the normalised `$wiki.$ext` model against the cached one for any difference. Without a difference, nothing will be created so that the user can see the updated wiki with the formal model instantly; otherwise, updated model diagrams will be regenerated from the updated macro, which can take only a couple of seconds longer than a normal update of the wiki page. Since we make use of the `org.eclipse.xtext.gmf.glue` extension to Xtext for the synchronous editing of Xtext, EMF and GMF models, an update of the graphs also takes less time than a complete regeneration of the graph models. In this way, updating the non-Miki macro parts of the page does not take extra time, making the Miki system as responsive as its Trac predecessor.

## 2.3 Deleting a model

If the Miki page is no longer needed, the model and auxiliary files generated from the Miki system shall be removed. This is done by the synchronous monitor sits in-between the running Eclipse and the Trac systems. Once a removal of the file `$wiki.$ext` happens, in a few milliseconds a notification from the file system will be caught by the running Eclipse monitor such that all `$wiki.$ext.*` files will be removed from the project resources, and the corresponding attachments in the Trac wiki page will be removed as well.

## 2.4 Comparing Miki with (Trac + Eclipse)

For all the three operational use cases, Table 1 compares the differences between Miki and its baseline predecessor (Trac + Eclipse) in the steps of basic usage scenarios, time a user spent, the network bandwidth; all based on the concrete Sluice Gate example. Here Miki stands for a browser client visiting the Miki wiki; Trac stands for a browser client visiting the Trac wiki, and Eclipse stands for an Eclipse instance that has the Xtext, GMF editor plugins instances for the supported DSL (in our cases the OpenPF<sup>4</sup> tool installed

<sup>4</sup><http://sead1.open.ac.uk/openpf>

on the client side).

Note that the time estimation of editing the wiki page is assumed to be 1 minutes just for an illustration, and other mouse click/shortcut key interactions require 1s each. The bandwidth estimation is based on the actual size of the DSL specification and generated diagrams.

## 3. RELATED WORK

Mozilla Ace for Cloud9 IDE<sup>5</sup> is one of the projects aiming to provide a syntax-highlighted text editor on the Web using the HTML5 canvas feature. Ideally it could be used to edit not only Javascript programs but also domain specific languages of various sorts. However, comparing to the mature model-driven tool development frameworks such as Xtext, Ace requires substantially more effort to customise the domain specific language for different requirements models. For requirements languages currently supported in Eclipse-based modeling environment, therefore it might require less investment to support a syntax highlighting online editor. To this aim, it would be nice if the Xtext framework could already generate a Web-based text editor automatically using the rich Ajax platform (RAP), as indicated by the following April Fool's day wish<sup>6</sup>. Lacking support of deploying a full Eclipse framework to the Web, however, this feature is not within the release plan of the Xtext project.

Ossher et al [8] propose to support flexible modeling using the BITkit tool built on top of Adobe Flex, which needs to be preinstalled. Miki could enable yet another level of flexibility, i.e. to synchronously edit the models using the wiki browsers alone. The Eclipse Sketch project<sup>7</sup> provides yet another route to support flexible modeling, where any type of graphical syntax could be supported through the recognition of the model from hand drawings. Since Sketch is supported by Eclipse, it is our belief that such graph recognition tool could be combined with the Miki system while such inputs could be provided by a whiteboard plugin for the wiki.

## 4. CONCLUSIONS AND FUTURE WORK

In this work, we have discussed the motivations of lowering the barriers of adoption for flexible modeling tools through the use of wikis, which can provide a synchronous bridge between requirements capture and the more formal modeling features of the IDE such as Eclipse. A proof-of-concept implementation has been detailed with a running example to explain how we tightly integrate any Eclipse-based graph modeling tool into the Trac wiki. Currently the tool supports any Eclipse-based modeling representations that can be supported by Xtext, EMF and GMF. An instance of the Miki system is exemplified at the website <http://computing-research.open.ac.uk/trac/openre>. Our initial experience that the text-based informal representations in Miki can be used effectively in eliciting formal requirements models, saving bandwidth, time and effort.

Without further empirical evidence of how frequent such use

<sup>5</sup><http://ace.ajax.org/>

<sup>6</sup><http://kthoms.wordpress.com/2010/04/01/bringing-xtext-to-the-web/>

<sup>7</sup><http://www.eclipse.org/sketch>

**Table 1: Comparing basic use cases between (Trac + Eclipse ) and Miki**

Operation	Trac + Eclipse	Sec.	Byte	Miki	Sec.	Byte
Installing	installing Eclipse instance	600	200 M	-		
Creating	starting Eclipse instance	20	0	-		
	editing DSL using Trac	30	346	editing DSL using Miki	30	346
	copying/pasting DSL from Trac to Eclipse	4	0	-		
	saving DSL in Eclipse as EMF	6	0	-		
	loading EMF in Eclipse as GMF diagram	3	0	-		
	screenshot from GMF to an image	1	0	-		
	uploading an image to Trac	10	10.6 K	submitting the Miki page	10	380
	inserting link to attachment in Trac	10	346	-		
Updating	starting Eclipse instance	20	0	-		
	editing DSL using Trac	10	350	editing DSL using Miki	10	350
	copying/pasting DSL from Trac to Eclipse	4	0	-		
	saving DSL in Eclipse as EMF	6	0	-		
	loading EMF in Eclipse as GMF diagram	3	0	-		
	screenshot from GMF to an image	1	0	-		
	uploading an image to Trac	10	10.6 K	submitting the Miki page	10	380
Deleting	deleting Trac page	1	380	delete Miki page	1	380
	deleting Trac attachments	1	0	-		
	deleting Eclipse model files	3	0	-		
	refreshing the Eclipse resource navigator	1	0	-		

cases happen in formal requirements modeling, here we only position that  $Miki > (Trac + Eclipse)$  as long as formal requirements need to be collaboratively edited. Further study is required to tell whether Web-based interaction limits the ability to edit the diagrams directly rather than having to edit the textual domain-specific languages.

It is remained to be experienced how Miki would help users engaging in the early stage of modeling. We plan to investigate the cognitive dimensions [4] when evaluating Miki's effectiveness in shortening the conceptual distances between users. We intend to conduct an empirical studies to evaluate the benefits with respect to supporting flexible modeling tools through the Miki system by logging the number of interactions. To support a spectrum of models ranging from as informal as a wiki page to those that are amenable to formal reasoning and execution, for example, we intend to study how to support the direct diagram editing in the Web-based interfaces. Also we would investigate the opportunity to integrate other flexible modeling tools with Miki .

## Acknowledgement

This work is in part supported by the EU FP7 SecureChange project (<http://securechange.eu>).

## 5. REFERENCES

- [1] Jorge Aranda, Steve M. Easterbrook, and Greg Wilson. Requirements in the wild: How small companies do it. In *RE*, pages 39–48. IEEE, 2007.
- [2] Moritz Eysholdt and Heiko Behrens. Xtext: implement your language faster than the quick and dirty way. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *SPLASH/OOPSLA Companion*, pages 307–309. ACM, 2010.
- [3] Orlena Gotel and Anthony Finkelstein. Extended requirements traceability: Results of an industrial case study. In *3rd IEEE International Symposium on Requirements Engineering (RE'97), January 5-8, 1997, Annapolis, MD, USA*, pages 226–235. IEEE Computer Society, 1997.
- [4] T.R.G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.
- [5] Charles B. Haley, Robin C. Laney, Jonathan D. Moffett, and Bashar Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Software Eng.*, 34(1):133–153, 2008.
- [6] Michael Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*. Addison-Wesley, New York, December/Winter 2001.
- [7] Panagiotis Louridas. Using wikis in software development. *IEEE Software*, 23(2):88–91, 2006.
- [8] Harold Ossher, Rachel K. E. Bellamy, Ian Simmonds, David Amid, Ateret Anaby-Tavor, Matthew Callery, Michael Desmond, Jacqueline de Vries, Amit Fisher, and Sophia Krasikov. Flexible modeling tools for pre-requirements analysis: conceptual architecture and research challenges. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *OOPSLA*, pages 848–864. ACM, 2010.
- [9] Bran Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003.
- [10] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *3rd IEEE International Symposium on Requirements Engineering (RE'97), January 5-8, 1997, Annapolis, MD, USA*, pages 226–235. IEEE Computer Society, 1997.

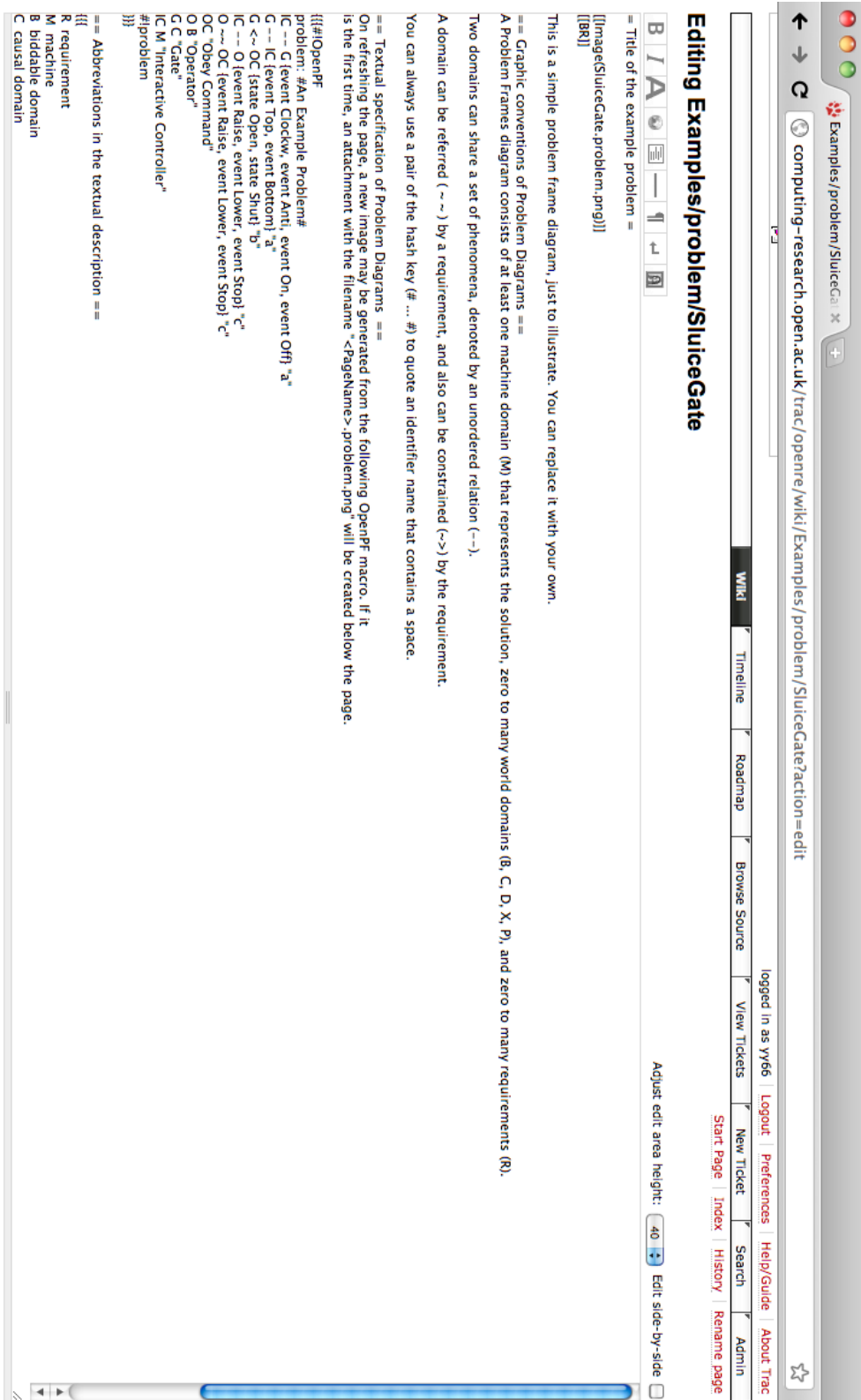


Figure 2: Input to Miki through the Trac editor page

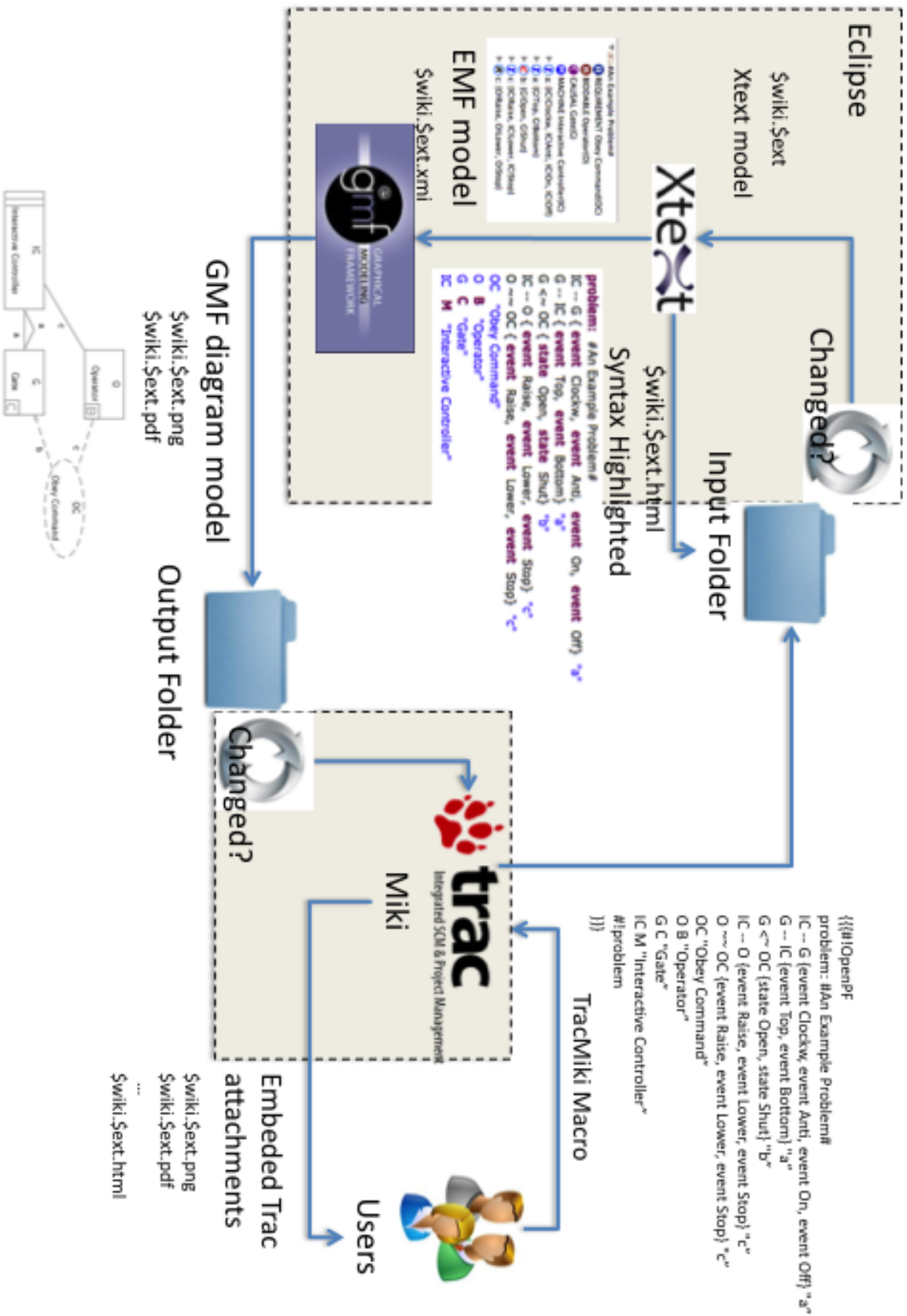


Figure 3: The architecture of the Miki system

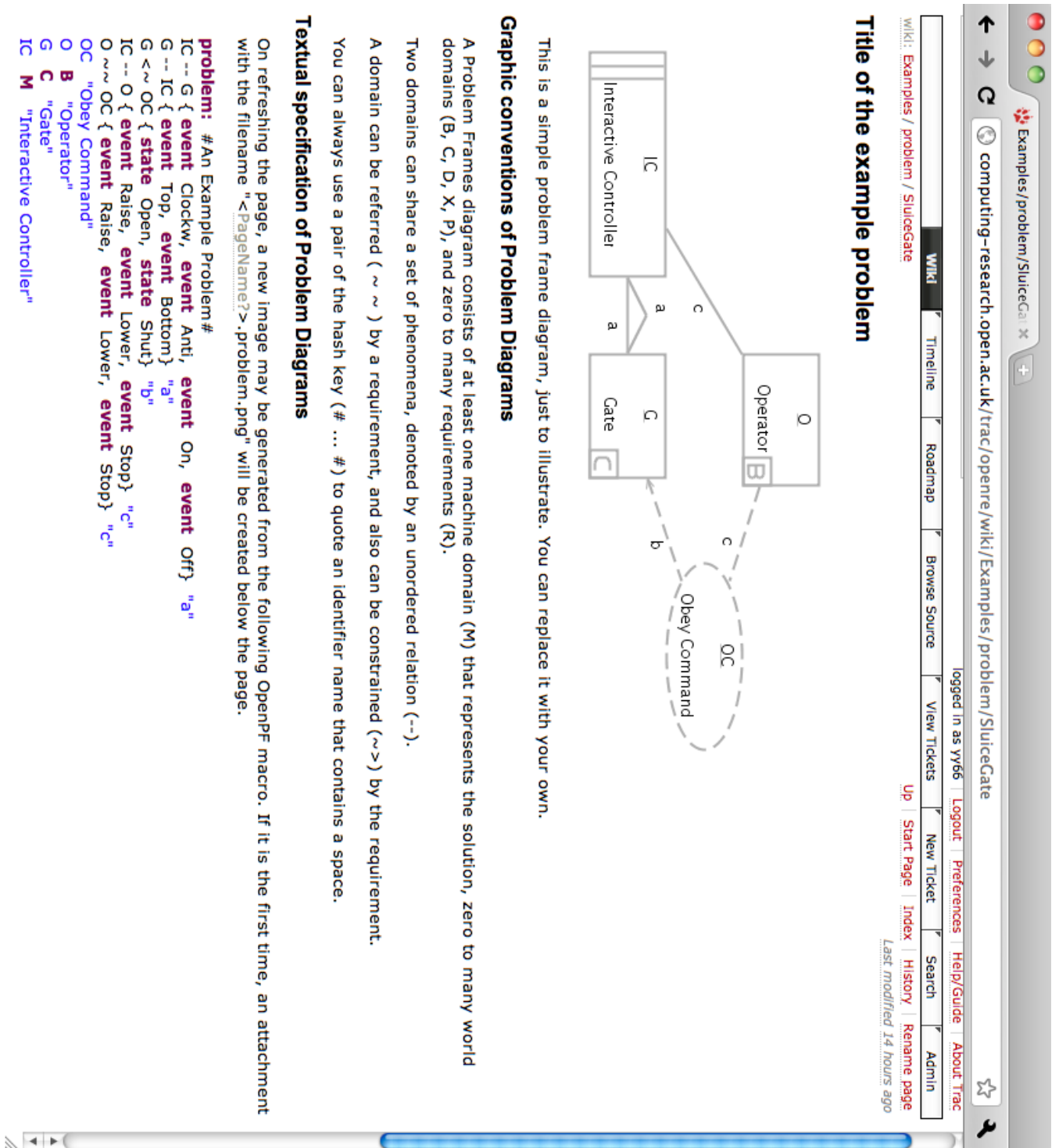


Figure 4: An example Miki page as output