



Norwegian University of
Science and Technology

e4's Toolkit Model – a new view on GUI

Hallvard Trætteberg, Associate Professor
Dept. of Computer and Information Sciences
Norwegian Univ. of Science and Technology

www.ntnu.no

Hallvard Trætteberg, e4's Toolkit Model

Let me present myself: I'm Hallvard Trætteberg, Associate Professor at the Norwegian University of Science and Technology. My research field is model-based user interface design, and it's this interest that has prompted me to contribute the Toolkit Model to e4.

Agenda

- Motivation
- Toolkit Model
 - model
 - architecture
 - (Javascript support)
- Eclipse editors and views
 - toolkit model editor
 - palette view
 - script editing
- Concluding remarks

In this part of the e4 webinar I will present the motivation for the Toolkit Model, explain what it is, a bit of how it works and how it is used for developing user interfaces.

Motivation

- Hand-coding GUIs is hard
 - GUI frameworks can be complex
 - conceptual gap between concrete GUI and the GUI code
- What works?
 - GUI builders are very useful for part of the design task
 - ... but handles only look & feel, behavior must still be coded
 - HTML+DOM+Javascript has caught on
 - ... but XML is not the best tree-structured data model around
 - Model-based techniques are maturing and more wide-spread
 - ... but has not really been successfully applied to UIs (until now?)
- Can a combination of these be used for e4?

Programming is considered hard by many, and UI programming is not the easiest of programming tasks either, since the frameworks for GUI programming can be complex. A complicating factor is the conceptual gap between what the user interface looks like and how it behaves, as experienced by the end-user, on one hand, and the code you must write to create that experience, on the other hand.

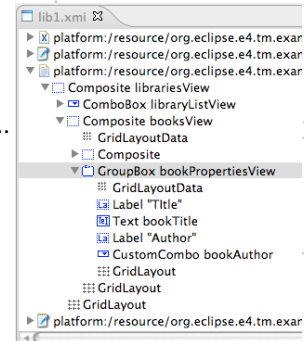
There are some techniques and tools that has become popular, that can provide inspiration for improving the situation:

- GUI builders are easy to use, since they are essentially WYSIWYG, when it comes to visual appearance. However, you usually have to generate and add code to implement the behavior you want.
- The HTML+DOM+Javascript combination has been very successful on the web, partly due to how code for the behavior is interleaved with a declarative representation of the user interface structure.
- Model-based techniques are becoming more wide-spread, but has not yet caught on for user interface development.

The Toolkit model tries to build on these, and will hopefully gain most of the advantages.

TM – model and architecture

- Ecore-based model of GUI elements
 - hierarchy of widgets
 - simple elements – label, text field, buttons, list, ...
 - composites – generic, group box, tab folder, ...
 - coming soon:
 - table and tree widgets
 - styling with CSS
 - 2d graphics
 - *natural extension of modeled workbench UI*
- EMF API & tools are used to manage TM instances
 - Java code with static and reflective API
 - generic editors and model-based techniques
 - transform with ATL, store with Teneo, share with CDO ...



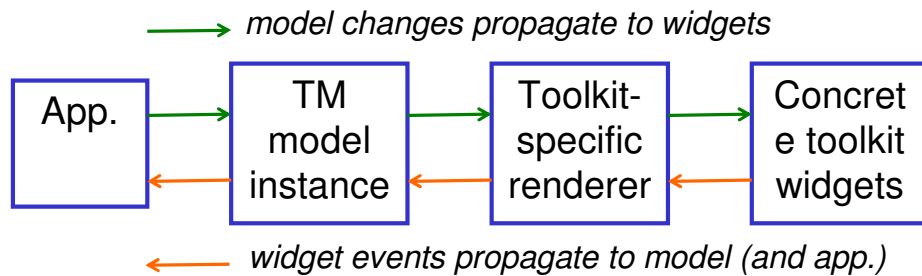
The Toolkit Model is an Ecore model of standard GUI elements, and thus provides a means for declarative representing the structure of a user interface. In this sense it is similar to HTML+DOM.

You hopefully know that Ecore is EMF's modeling language that is becoming more and more wide-spread within the Eclipse community. In fact, it is the basis for the modeled workbench user interface that was presented at the previous webinar. In some sense, the Toolkit Model is for applications, what the modeled UI is for the workbench. Instances of the Toolkit Model are similar to the HTML DOM, but by being built on widely used model-technology, even easier to manage.

The Toolkit Model support most standard widgets, like labels, text fields, buttons, lists, group boxes, tab folders etc. More complex ones like tables and trees are missing, but will soon be supported.

TM – model and architecture

- The renderer (in the middle)
 - builds the initial UI from the model and keeps them in sync
 - the renderer is toolkit-specific, but the model is fairly portable



- The application only sees the (slightly abstract) model

Javascript support

- Wrappers for Ecore objects make them seem like native Javascript objects
- Extra behavior may be added
 - annotations to Ecore operations
 - separate files contributing methods to EClasses
 - EObjects may have their own methods
 - onChange<Property> methods used for event handling
- Standard Ecore classes have already been extended
 - create new objects from EClasses
 - navigate in EObject hierarchies by index or name
 - functional API for searching, mapping and filtering
- Some support for other EMF functionality

Javascript has several advantages: it's more flexible since it's interpreted, integrates well with Java and supports extending a base API with new methods. The latter means that a Java objects, as seen by the Javascript code, may have additional functionality that makes scripts easier to read and write than their Java counterpart. This is useful for scripting both TM objects and your own EMF-based data-objects.

Typical workflow

- Create GUI sketch with appropriate tool
 - e.g. wireframesketcher is no-nonsense and EMF-based
- Create TM instance
 - drag'n drop model fragments from palette
 - edit properties and add layout and style objects
- Create sample data
 - create Ecore model of domain data
 - create example data for data model
- Script model
 - populate with example data
 - react to user events

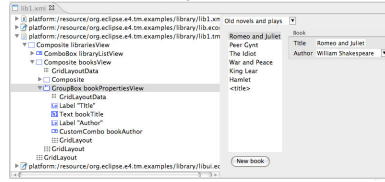
These steps are based on my own style of working. It's tempting to skip the first one, but TM's GUI builder support is currently not good enough to design the UI with it.

It's usually easiest to create new TM instances by copying from existing ones. The palette view supports dragging from both the model tree and the preview.

Not everybody will want to use Ecore for modeling application data, but in my experience it is a great time-saver.

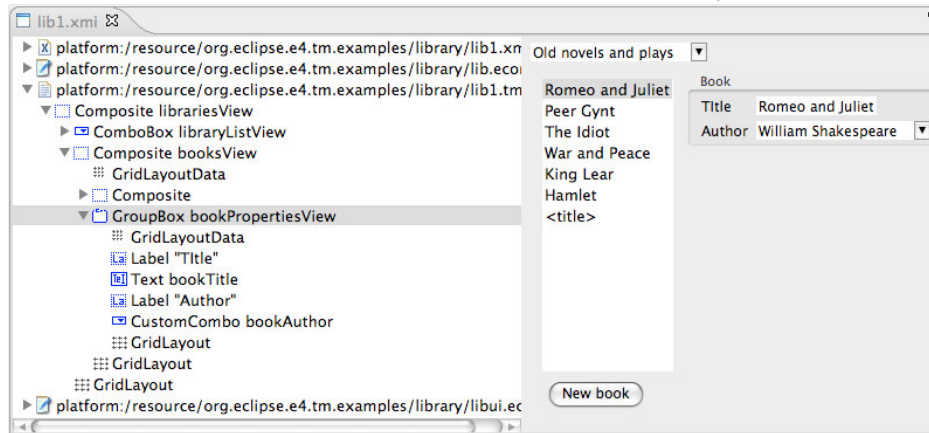
Editors and views

- Tree-based editor with preview
 - “poor man’s” GUI builder
 - can load EMF application data
 - supports scripting
 - GUI gradually becomes functional



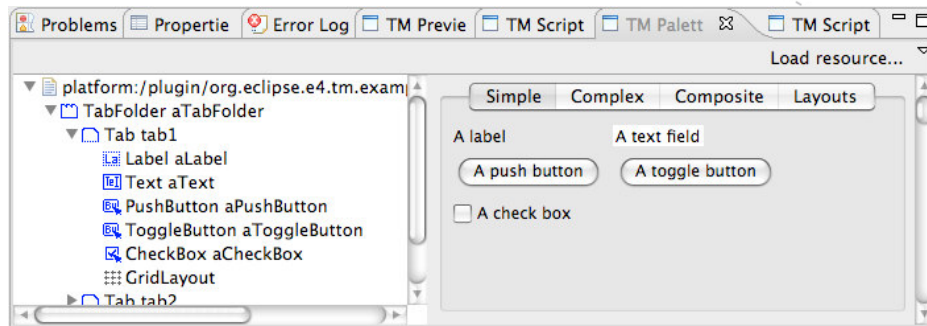
- Palette view – drag’n drop model fragments
- Script editor – edit Javascript for model element
- Script scrapbook – try out snippets on live model
- Preview pane – previews active editor content

Tree-based editor with preview



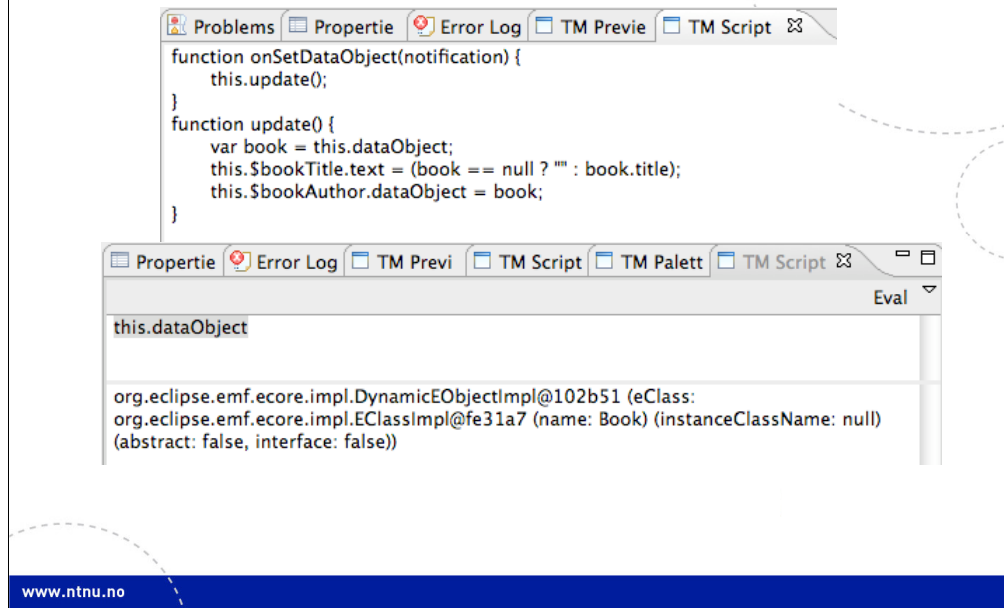
Not just a poor-mans GUI builder. It supports scripting and loading application data. The editor essentially allows editing a running application. Both the tree view and preview reflects the current runtime state of the application.

Palette view – drag'n drop model fragments



A read only view of a TM instance, with preview. Drag'n drop of model fragments and GUI elements is supported. By placing it neatly underneath the editor, it may be used as a palette of model fragments.

Script editing – edit and try



There are two scripting-related views, that automatically track the currently active editor. The first allows editing the script of a widget, the other executing scripts in the context of the selected UI element. The editor is currently just a plain text field, but we would like to have a syntax-aware editor.

Relation to other e4 components

- No explicit relation to other e4 components
- A natural extension of the modeled workbench UI
 - while the modeled UI defines the high-level structure of applications (editors and views), TM can be used to model the content
 - same idea and architecture, they could (and should?) be merged
- Same motivation as XWT, but different choices
 - TM has explicit model (schema), while XWT has not (relies on Java's reflection to map from XML to Java names)
 - XWT is necessarily close to SWT, while TM can choose to differ
- Javascript support
 - extends existing Java support to EMF
 - could (should?) be aligned with scripting for the platform

Although not part of the early e4 components, it fits nicely in as a natural extension of the modeled workbench UI.

TM has the same goal as XWT, but is chosen a different representation (Ecore vs. XML) and is more independent of SWT.

Scripting EMF and scripting the platform is different, but could/should be integrated.

Potential of Toolkit Model

- Platform independence
 - by being more abstract, it can better supports many platforms
- Separation of structure, behavior and style
- Scripting a model provides flexibility
 - much of the UIs behavior can be changed without rebuilding
- Model-based techniques
 - generate UI from domain-specific models
- Toolkit Model as basis for thin client
 - UI sessions are manageable with EMF technology
 - store UI sessions with Teneo and share with CDO
 - find and update with Query and Transactions

 *Toolkit Model-based UIs are “cloud ready”*