Metawidget White Paper

What Good is an OIM?

Richard Kennard

August 2011

http://metawidget.org

1. Introduction

Metawidget has coined the term Object User Interface Mapping (OIM). It's a new take on an old problem. Like any new approach, it can sometimes be difficult to identify where it can be applied. This white paper outlines a number of use cases for an OIM, citing examples of real world implementations. In short, this white paper helps answer the question: what good is an OIM?

2. Generate User Interfaces from Existing Business Objects

OIMs excel at generating widgets based on your *existing* UI framework, matched to your *existing* application business objects. Whether the application is a government tool for transaction processing or a pharmaceutical tool for database maintenance [1], OIMs are ideally suited (Figure 1).

One lead developer summarised: “Many frameworks or tools enforce the [tool] designer's vision on how solutions should be architected. What I liked about Metawidget is that I could drop it in whatever architecture I was using” [1]. Another concluded: “In 10 years of software development, I can't count the number of times I've needed a simple form for users to enter or update data... no one has extended [a solution] in a general way to apply to a broad audience. Certainly there have been 'form code generators', but creating the form at runtime from metadata is a far more elegant approach in my opinion” [1].

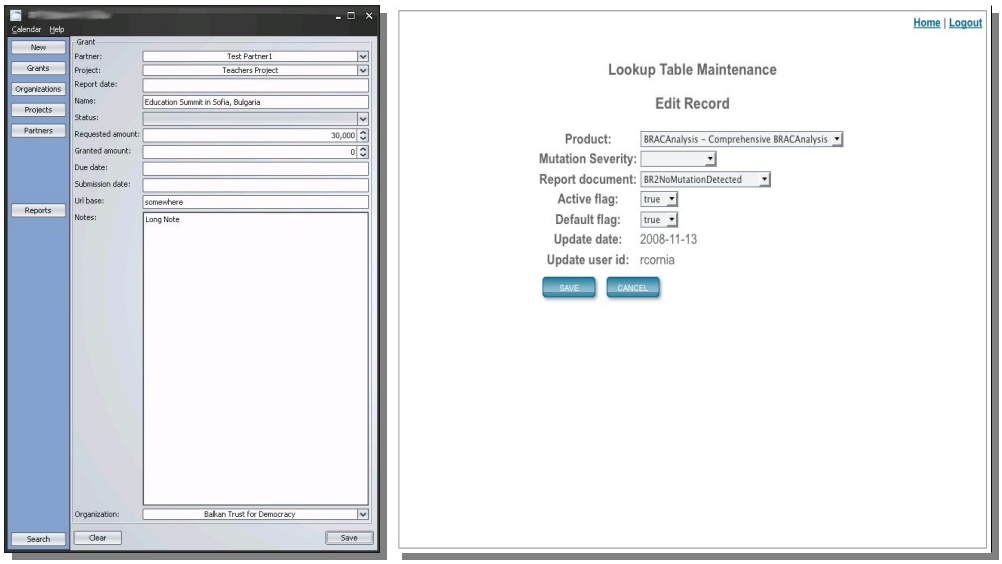

Figure 1: Transaction processing and database maintenance

OIMs are less cumbersome than visual IDE designers such as Matisse (Figure 2), or modelling languages such as Facelets. They require no repetitive definitions between UI and business objects. These are laborious to define and error-prone to maintain. OIMs are also more flexible than language-based tools such as Naked Objects. OIMs have an explicit focus on generating the *same* UI you would previously have coded by hand.

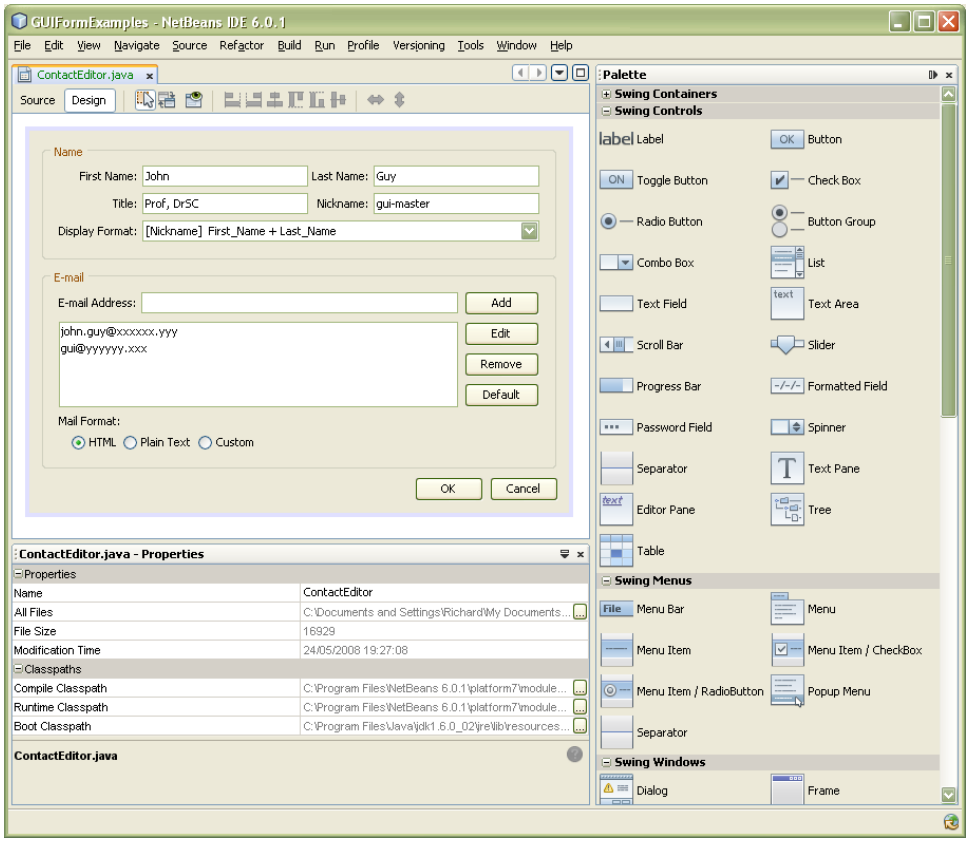

Figure 2: NetBeans Matisse

The introduction of an OIM can reduce the amount of repetitive, error-prone UI code in an architecture by up to 70% [2]. One architect remarked: “it's a fairly established software engineering principle that the more you have to repeat something the higher the error is, the higher the chances there's going to be an error in the code” [3]. Another: “We had an experience in our last project, that a lot of view related bugs would come from missing required fields, wrong formatting and changing the model and not changing the view. Also, keeping those in synch, required a lot of effort, not complex, but we had most of our junior programmers dedicated to fixing those silly problems. [After we introduced Metawidget] it simplified a lot and this category of bug has simply disappeared” [1].

3. Generate User Interfaces from Unconventional Sources

Beyond conventional UI generation, the five characteristics of an OIM [2] allow it to generate UIs from arbitrary sources. This opens new possibilities for functionality.

One health application allowed users to describe portions of their required UI themselves, using a simplified Domain Specific Language. The generated portion was then inserted into the application dynamically, based on the logged in user's roles and permissions. The team reflected: “With our requirements Metawidget was basically the only option. Our usual techniques would not have done the job” [4].

Another theatre system application exposed a plugin API that generated UIs for each plugin on-demand, at

runtime. The team lead stressed: “One of the really important reasons I used Metawidget for this was that I have plug-ins that provide data objects for, and I/O support for, different types of [theatre] systems. I wanted to be able to just add [plug-ins] and have them be immediately editable without [them] needing to implement any UI code at all” [1].

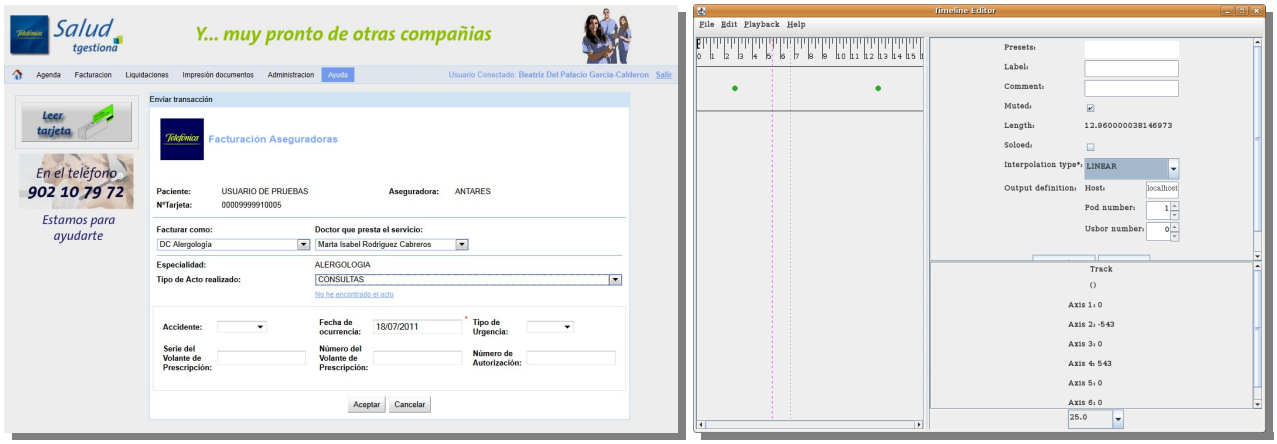

Figure 3: Health application and theatre system

The ability of OIMs to inspect, and collate, metadata from arbitrary back-end sources permits a new level of dynamically generated UIs.

4. Generate User Interfaces with Consistent Behaviour

OIMs integrate with your existing back-end architecture, and deliver results integrated to your existing front-end framework. They provide a consistency of behaviour and layout across all the screens of your application. This goes far beyond traditional technologies such as CSS or Swing Look and Feels.

OIMs generate widgets at runtime from centralised code, ensuring: consistent choice of widgets for data types (i.e. date pickers for all dates, spinners for all integers); consistent insertion of validators and converters (i.e. for currency); consistent enforcing of data limits (i.e. minimum values for integers, maximum lengths for strings); and consistent layout of sub-entities (i.e. separated by section headings, or separated into tabs). UIs created by an OIM are more consistent and more robust than those created by hand. For example, few developers bother to put `maxlength` attributes on every one of their text fields: it is too laborious and likely to change over time, requiring error-prone maintenance. With an OIM, such limits are enforced automatically.

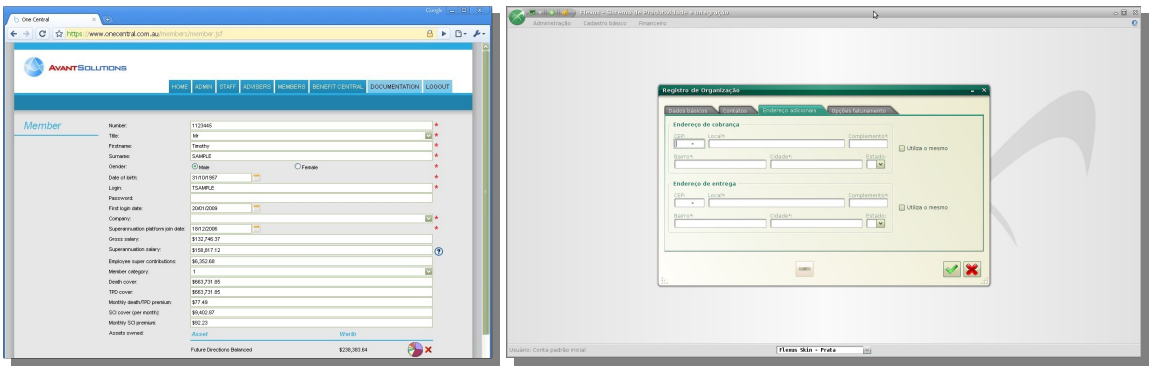

Figure 4: Superannuation application and ERP application

One ERP application (Figure 4) used Metawidget to ensure consistent behaviour across a large number of screens in a large team. The team remarked: “Metawidget is really useful in the way it is the foundation to build your solution... A great advantage [it offers] is UI standards. It is really hard to keep consistency, visual or functional standards when building UI in a large team. However when it is generated dynamically, the rules are centred and even the customisation is somehow controlled”.

5. A Platform for Adopting New Technologies

OIMs provide a common platform for technologies participating in UI development. Vendors can leverage this to lower the barrier of entry in adopting their technology, or to open their technology to a new market.

The Bean Validation plugin allows Metawidget to inspect Bean Validation annotations and translate them into UI data limits. This automatically allows Bean Validation to be applied to diverse UI frameworks including Swing, SWT, Spring and Java Server Faces.

The RichFaces plugin allows Metawidget to generate RichFaces components for arbitrary back-end architectures. This automatically allows RichFaces to bind to technologies such as Drools and jBPM. If an application is already using Metawidget, adding the RichFaces plugin will automatically upgrade *every* widget in the application across *every* screen to use rich UI widgets such as date pickers and spinners.

6. Conclusion

This white paper has outlined a number of use cases for an OIM, and cited examples of real world implementations. We hope this will help you identify areas Metawidget could be used within your own applications. You can then apply the new approach of an OIM to the old problem of repetitive, error-prone UI code - saving both you and your organisation time and money.

7. References

1. Kennard, R. 2011, 'Adoption Studies'. Retrieved from http://metawidget.org/media/whitepaper/MetawidgetWhitePaper-AdoptionStudies.pdf
2. Kennard, R. & Leaney, J. 2010, Towards a General Purpose Architecture for UI Generation. *Journal of Systems and Software.*
3. Kennard, R., Edmonds, E. & Leaney, J. 2009, Separation Anxiety: stresses of developing a modern day Separable User Interface. *2nd International Conference on Human System Interaction.*
4. Kennard, R. 2011, 'Case Study: Telefónica Health Portal'. Retrieved from http://metawidget.org/media/whitepaper/MetawidgetWhitePaper-TelefonicaHealthPortal.pdf