Metawidget White Paper

Case Study: JBoss Forge

Richard Kennard

September 2012

http://metawidget.org

1. Introduction

This white paper presents a case study of using Metawidget to provide automatic User Interface (UI) generation for JBoss Forge. JBoss Forge is a core framework for rapid-application development in a standards-based environment. Internally, Forge uses Metawidget to statically generate UIs for its applications. The generated UIs are able to take full advantage of Forge's rich metamodel. This includes metadata such as the business-specified order of fields, validation constraints, and the context of screens (search screen, viewing screen, editing screen etc). Forge further allows developers to customize its UI generation by leveraging Metawidget's pipeline architecture.

2. Organisation and Product Overview

JBoss is a division of Red Hat, a leading industry provider of enterprise middleware. Their products include Red Hat Enterprise Linux and JBoss Application Server. They are consistently ranked one of the leading industry vendors for providing software to the enterprise. JBoss Forge is their core framework and next-generation shell for tooling and automation at a command-line level.

JBoss originally began development of Forge in 2010. They developed a command-line shell for creating and modifying Java projects. The shell didn't simply manipulate text files. It possessed a deep understanding of Java semantics, based on Eclipse JDT, and could therefore manipulate Java code with sophistication and precision. It supported a wide variety of techniques and technologies, including: creating getter/setters for JavaBeans; configuring POMs for Maven projects; and deploying to cloud environments such as Red Hat's OpenShift. Forge further supported integration into IDEs (see Figure 1) and scripting for automation of repetitive tasks. Finally, Forge had a well-defined and flexible plug-in architecture that allowed developers to integrate their own technologies.

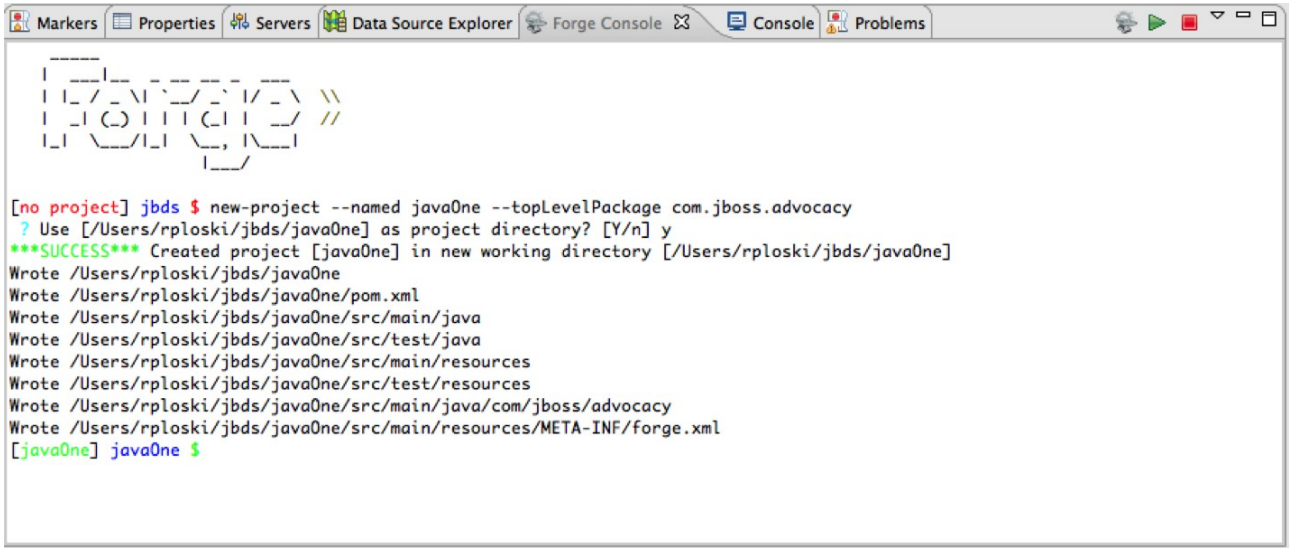

Figure 1: JBoss Forge running inside Eclipse IDE

In December 2010 the Forge team turned their attention to UI generation. They had created a shell that could generate much of the *back*-end of an application, but they knew they needed front-end generation too. The team began searching for an approach that would allow them to statically generate the UI for an application, whilst at the same time leveraging the existing technologies they supported. This was where Metawidget came in.

3. Integration of Metawidget

Dan Allen, Principal Software Engineer at Red Hat, summarised the team's decision to adopt Metawidget. “In searching for a solution [for JBoss Forge], we aimed to bridge the gap between server-side and client-side technologies in a way that would leverage the intelligence resident in the standards-based programming

model. In particular, we wanted a project that could integrate well with the Java EE technologies and, in addition, Red Hat's companion extensions. Combinations included JSF and RichFaces, JPA and Hibernate, and Bean Validation and Hibernate Validator. The UI, particularly the forms, should align with the metadata the server-side technologies define, without intervention from the programmer. Furthermore, the alignment should not be limited to only these standard sources".

JBoss evaluated several approaches but "found Metawidget uniquely qualified. It gathers all the information it can from the existing architecture, whether it be standards-based or otherwise, and uses that information to generate consistent interface controls. With each technology introduced into the application, Metawidget's intelligence only grows stronger, thanks to its plug-in architecture".

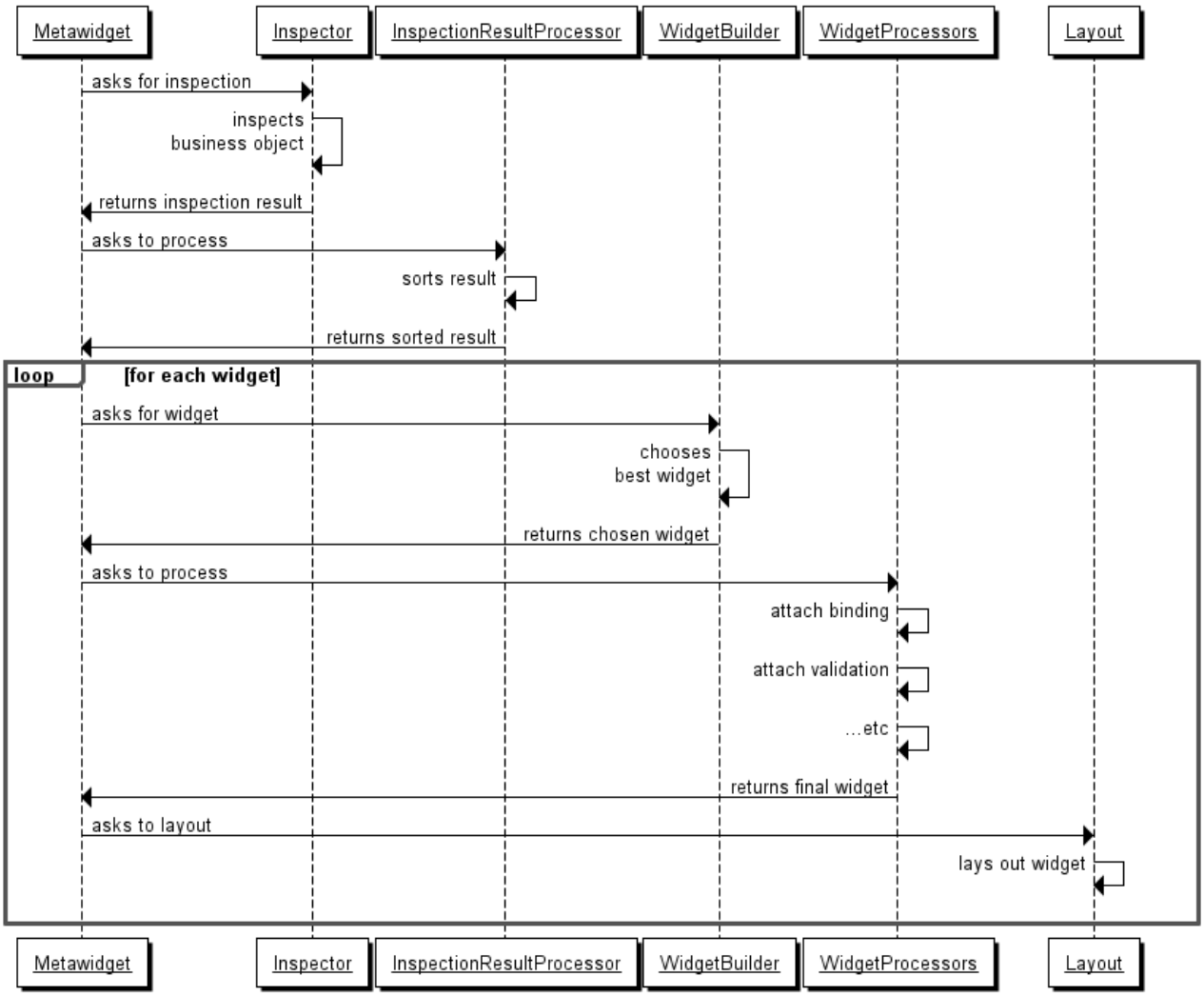

Figure 2: Metawidget pipeline applies equally to runtime and static environments

Metawidget's plug-in architecture is the result of extensive research into development teams and their practices. Years of developer interviews[1], adoptions studies[2] and case studies[3] led to the derivation of Metawidget's general purpose architecture for UI generation[4]. This architecture was designed to be extensible in many dimensions, in order to support a wide variety of technologies and development styles.

The architecture can be summarized as a five-stage pipeline (see Figure 2). A unique property of this pipeline is that it applies equally well to either runtime or static UI generation. Indeed, the exact same Metawidget `Inspectors` and `InspectionResultProcessors` can be plugged-in to drive either runtime generation for dynamic UIs, or static generation for frameworks such as JBoss Forge. All Metawidget Inspectors contain a pluggable `PropertyStyle`. The Forge team developed a `ForgePropertyStyle` capable of reading Forge's static metamodel and feeding it transparently into the rest of the pipeline. This allowed Forge to leverage Metawidget's existing support for inspecting JPA annotations, Hibernate Validator annotations, Java 5 generics, and much more.

Beyond inspection, the Forge and Metawidget teams created a suite of static `WidgetBuilders` and `Layouts` (see Figure 2). These were capable of generating static UIs for disparate technologies such as Java Server Faces, Spring MVC and JQuery (for JBoss AeroGear). The `WidgetBuilders` also supported third-party extension libaries, such as RichFaces[5]. The objective was to statically generate best practice UIs. It was important Forge follow industry standards and generate pure Java EE 6 code, with no proprietary frameworks or 'magic' behind the scenes.

Many of these static `WidgetBuilders` and `Layouts` were generic, so the Forge team contributed them back to the Metawidget Open Source project. However the team also developed some Forge-specific `WidgetBuilders`. They were keen that their generated UIs support not just the usual Create, Retrieve, Update and Delete (CRUD) but the full gamut of entity relationships, including: one to many, many to one, one to one and many to many. In addition, retrieval (the 'R' in CRUD) meant not just loading by id, but being able to search and browse by filter criteria. By developing specialized `WidgetBuilders` with knowledge of Forge internals, the team were able to deliver this rich functionality.

Atop this, the team utilized `WidgetProcessors` to refine the generated widgets based on the context of their use. For example, whilst the 'create' and 'view' screens displayed all domain model fields, the 'search' screen displayed only a subset of them, for use as filter criteria.

---

1 http://metawidget.org/media/whitepaper/MetawidgetWhitePaper-DuplicationInUserInterfaces.pdf

2 http://metawidget.org/media/whitepaper/MetawidgetWhitePaper-AdoptionStudies.pdf

3 http://metawidget.org/media/whitepaper/MetawidgetWhitePaper-TelefonicaHealthPortal.pdf

4 http://metawidget.org/media/downloads/Derivation%20of%20a%20General%20Purpose%20Architecture%20for%20Automatic%20User%20Interface%20Generation.pdf

5 http://jboss.org/richfaces

In a novel twist, the Forge team discovered they could re-purpose Metawidget's five-stage pipeline to generate not just UI controls, but fragments of Java code as well. The pipeline is designed to be agnostic to any particular UI technology, be it XML-based such as Java Server Faces, or programmatic such as controlling the Swing API. This agnosticism enables the pipeline to apply equally well to outputting blocks of Java code. The Forge team were able to reuse the exact same `Inspectors` and `InspectionResultProcessors` that drove their page generation to drive generation of their backing beans and UI controllers.

Finally, the team were able to leverage cutting-edge frameworks such as Bootstrap CSS[6] to give their UIs a best-of-breed appearance (see Figures 3 and 4).

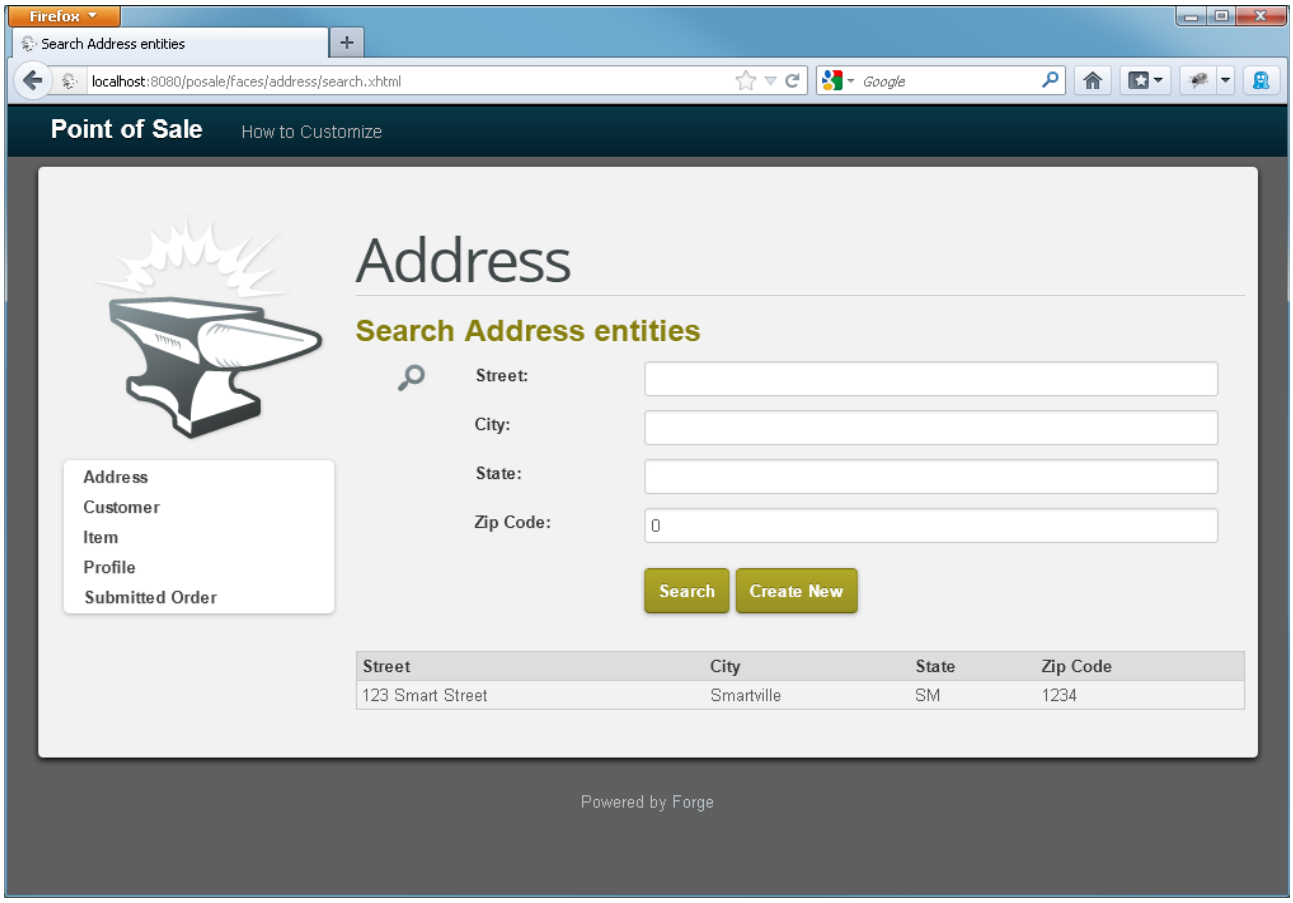

Figure 3: Metawidget-generated search screen

6 http://twitter.github.com/bootstrap

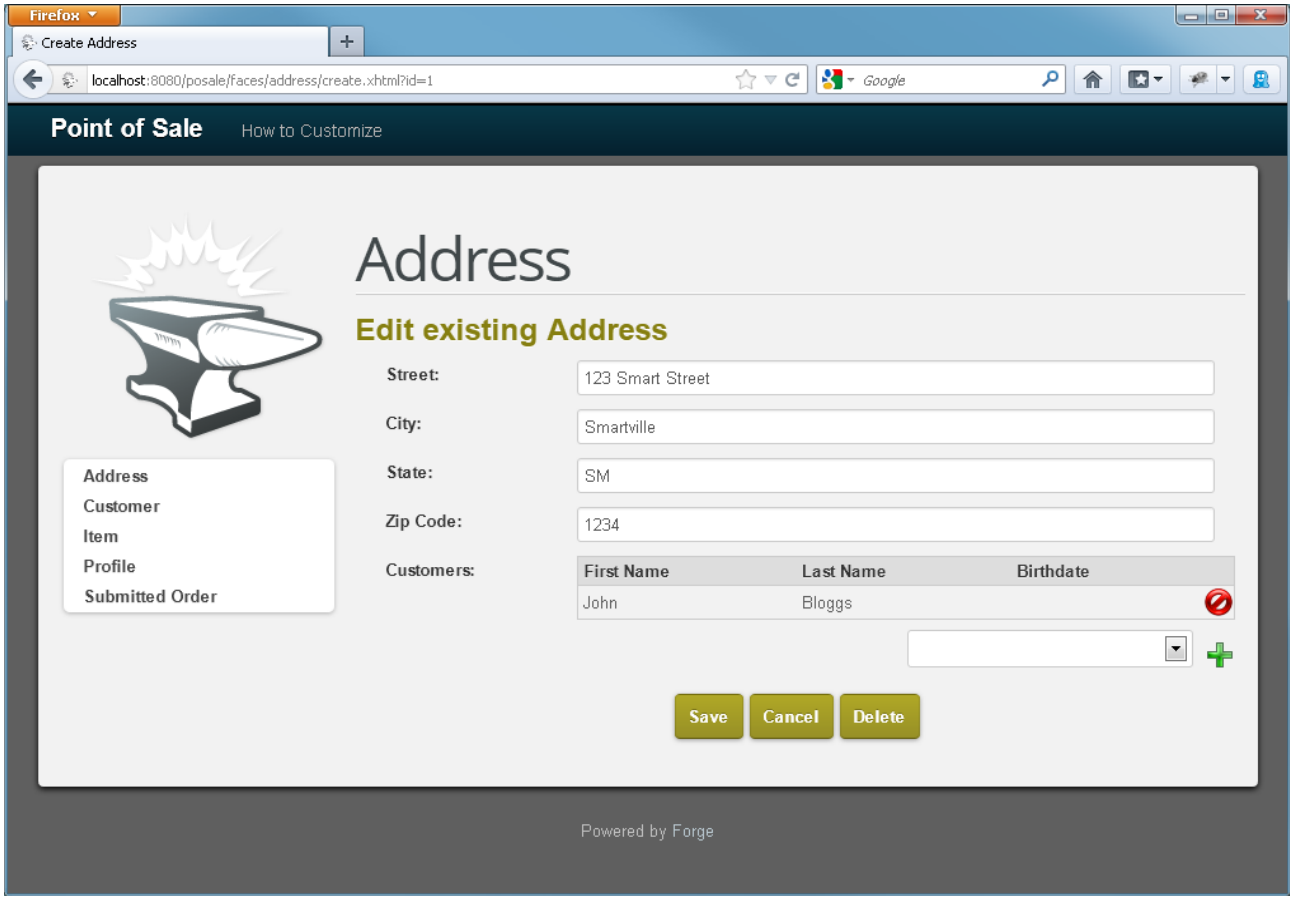

Figure 4: Metawidget-generated create screen

4. Conclusion

In closing, we asked the Principal Software Engineer how he would sum up the team's experiences with Metawidget? “In summary, we found Metawidget to be an excellent solution. It's clear that a lot of work has gone into understanding different enterprise architectures, both standard and non-standard, to ensure Metawidget integrates well with them. The result is a project we think will be very relevant and applicable to our enterprise customers in the future”.

Dan also had ideas for the future: “Personally, I was intrigued by the idea of plugging in less orthodox technologies to generate UIs from novel sources, such as generating a UI from a rule engine. We have a lot of customers using Drools [Red Hat's rule engine], and it would be great to open new possibilities for them. This is where Red Hat's focus on interoperability, as well as choice, plays such an important role. Metawidget both benefits from this competitive advantage and helps to propagate it.”

5. Resources

JBoss Forge: http://jboss.org/forge

Metawidget: http://metawidget.org