Declarative User Interface Generation

Richard Kennard
E-mail: richard@kennardconsulting.com
Web: http://www.kennardconsulting.com

Introduction

User Interfaces are a critical part of any user-facing software. For many years, projects have invested a high proportion of their time and resources in building effective interfaces [Myers92], so any tools or degrees of automation that can be introduced can have significant impact.

Furthermore, the recent explosion in the number of different devices upon which an application may run, from mobile devices to set-top boxes to traditional PCs, has increased the emphasis on flexible interface development.

It is a straightforward observation that all User Interface are, by their nature, representations of underlying systems - to greater or lesser degrees of abstraction. From there it is a logical step to propose they might be *automatically derivable* from underlying systems and, indeed, this has been a widely discussed topic in the research community with several approaches being developed.

However, despite its inherent value this work seems to have had limited impact within industry. This would imply there are certain 'barriers to adoption'. For my research, the objectives would be:

1. review the literature of existing research in the field of automatic user interface generation
2. identify barriers preventing this work having more of an impact in industry, and develop solutions
3. develop a prototype that embodies those solutions
4. evaluate the prototype against established criteria

Related Work

There is an exciting amount of research to draw on in the field of Automatic User Interface Generation from Declarative Models [Schlungbaum96]. Projects such as COUSIN [Hayes85], HUMANOID [Szekely93], TRIDENT [Bodart95], ADEPT [Wilson96], USIXML [Vanderdonckt04] and MIST [Bhatia06] have all explored a variety of approaches.

I propose to combine this work with industry developments such as adding metadata support into programming languages [Gosling05, CLI06], and successes in Object Relational Mapping [JSR06] - an area with similar challenges that was once described as 'the Vietnam of Computer Science' [Neward06] but has recently appeared to reach consensus.

Methodology and Approach

To elaborate on the three objectives defined in the introduction, I propose the following approach:

1. Literature Review

I would propose to review the existing research in the field, which dates back some twenty years – from desktop PC interfaces to, more recently, mobile devices. Whilst I hope to encounter summary papers that help distil some of this large amount of research, I would still anticipate the reviewing period could easily consume 6 months.

Though much of the existing research has been successful in and of itself, it does not seem to have had significant impact outside the research community, indicating there must be some 'barriers to adoption' within industry. I would seek to identify these barriers and propose solutions to them, ultimately developing a prototype that embodies those solutions.

2. Identify barriers and solutions

Whilst exactly what the barriers are and how to overcome them would be part of my research, as a starting point I propose three barriers:

i. Proprietary declarative modelling

Typically, user interfaces draw together many qualities of a system that are not formally specified in any one place, if at all. For example, a UI dropdown box may have a data type specified in an RDBMS schema but its possible choices may live within application code.

Whilst it may be possible for an automatic user interface generator to pull this information together from disparate sources, it is tempting to instead define a custom language or tool. This has the advantage of:

1. Defining the declarative model in one place
2. Allowing the declarative model to include concepts not formally specified elsewhere

Doing so, however:

1. Requires developers to learn the custom language or tool
2. Locks the declarative model into a proprietary format
3. Results in duplicating information in both the declarative model and the system itself

I would propose to have the automatic user interface generator derive its declarative model by pulling together native information from disparate sources, such as schemas from RDBMS' and types from strongly-typed languages, and additional metadata from those platforms that support it.

Such an approach presents a number of technical difficulties, such as researching the different mechanisms for interrogating a variety of disparate resources, and how to amalgamate the results into a uniform, useful whole. I would anticipate this work could take some 12 months.

ii. Proprietary user interface frameworks

User interfaces must ultimately be expressed in a form native to their target platform, such as HTML or X-Windows widgets. However, it is generally understood this is too low-level for developers to work most of the time, so many industry frameworks such as Java Server Faces [JSR06a], ASP.NET [CLI06] and Swing [JPS06] have been developed to provide higher-level abstractions.

These frameworks have mature and widely-used support for features such as:

1. custom controls
2. navigation and flow of the interface
3. layout managers

Any automatic user interface generator that directly generates its user interface effectively overlaps these frameworks and puts itself in *competition* with them: the generator needs to both generate interfaces *and* mirror, feature for feature, other frameworks against which industry will compare it.

Those user interface generators that broaden their focus to task-based modelling are increasingly vulnerable to this problem, as their scope also overlaps the domain of Rule Engines, Business Process Modelling Engines and traditional programming languages, all of which they must compete with.

I would propose to have the automatic user interface generator leverage existing frameworks by acting as a plugin. Rather than directly generating HTML code or X-Windows widgets, it would generate Java Server Faces UIComponents, ASP.NET WebControls or Swing JComponents.

This allows developers to fall back to the established framework for areas where the automatic generation falls short. It also allows developers to enhance existing projects written in the frameworks on a screen-by-screen basis. Furthermore, I would propose to limit the scope to only generating well-defined portions of the user interface, without overlapping into navigation, business rules or process flow.

There are an exhaustive number of User Interface frameworks and developing plugins for all of them would not be

feasible. I would concentrate on the industry-leading frameworks, but even then I anticipate this work could take 12 months.

iii. Emphasis on static generation

Static code generation only helps with *writing* code. It does not alleviate unit testing and can even exacerbate the updating, documenting and handover because it quickly generates volumes of code that nobody - not even the developer who runs the generation tool - initially understands.

A more effective approach may be to generate the interfaces at runtime. An analogous case can be made to Object Relational Mapping (ORM) tools, which come in two flavours: those that generate the mapping code statically (by, say, pre-creating combinations of stored procedures in the RDBMS) and those that infer it dynamically (by introspecting a given object and dynamically generating SQL statements). The latter approach has emerged dominant in recent years, as attested to by industry specifications like the Java Persistence Architecture [JSR06].

A closer analogy may be to rule engines: an application takes an arbitrary number of runtime objects, hands them to a rule engine, and the rule engine returns a decision based on processing internal rules. For automatic user interface generators, the application could take a number of objects - including classes and XML files - hand them to the generation engine, and receive back a UI component for dynamic insertion into the application's UI framework.

A further advantage of runtime generation is that it can infer properties of a system only apparent at runtime. For example, a sub-object could be replaced with one of a different type, and the user interface could automatically change to reflect this.

I would propose to have the automatic user interface generator use introspection and other techniques to generate user interfaces at runtime. Such an approach would involve in-depth understanding of the introspection mechanisms of the underlying platform, so I would anticipate it could take 6 months.

3. Develop a Prototype

Given the research's emphasis on applicability to industry, I propose to build a prototype as an Open Source project with regular industry and research community input.

I would expect this work to produce several milestone builds of the software, each with increasing industry and research community exposure, over the anticipated 3 to 4 year time line of the research.

4. Evaluate the Prototype

Ultimately, the prototype would be evaluated on several criteria. The foremost would be effectiveness in real world use cases. For this I propose to utilize existing production systems as built by my company Kennard Consulting, as well as feedback from industry on the Open Source project.

Other criteria would be the quality of the software architecture, including its elegance and extensibility, and its applicability to a range of real and imagined scenarios. For this I propose to utilize feedback from the research community on the Open Source project.

References

[Bhatia06] Bhatia,, S., *Scientists in the MIST: Simplifying Interface Design for End Users*
[Bodart95] Bodart, F., *Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide*
[CLI06] *ECMA-335: Common Language Infrastructure (CLI)*
[JPS06] Java Platform, Standard Edition 6 API Specification
[JSR06] *JSR 220: Enterprise JavaBeans 3.0, Java Persistence API*
[JSR06a] *JSR 252: JavaServer Faces 1.2*
[Gosling05] Gosling, J. et al, *Java Language Specification*
[Hayes85] Hayes, P.J. et al, *Design Alternatives for UI Management Systems Based on Experience with COUSIN*
[Schlungbaum96] Schlungbaum, E. et al, *Automatic User Interface Generation from Declarative Models*
[Szekely93] Szekely, P., *Beyond Interface Builders: Model-Based Interface Tools*
[Myers92] Myers, B. A. et al, *Survey On User Interface Programming*
[Neward06] Neward, T., *The Vietnam of Computer Science*
[Vanderdonckt04] Vanderdonckt, J., A *User Interface Description Language for Specifying Multimodal User Interfaces*
[Wilson96] Wilson, S. et al, *Bridging the Generation Gap: From Work Tasks to User Interface Designs*

Explanation of Attachments

I attach the following documentation in support of my proposal:

Section 9: Academic Merit

I attach a certified copy of my degree transcript. The transcript confirms I received a First Class BSc (Honours) degree in Computer Science. I attach a copy of my degree certificate. The certificate is *not* certified as the original is in England and I could not get it sent over in time.

There is also a need to explain the Honours component of my degree: my University of Nottingham Honours degree was 3 years, with 33% of the final year being a research component. I believe this is typical for England, whereas I understand Australian Honours degrees are usually 4 years, with 100% of the final year being research.

My final year, individual dissertation involved:

- proposing an original idea
- researching and implementing the idea in software
- writing a 50,000+ word dissertation of findings

I elaborate on this in Section 15 below.

Section 10: Prizes, Awards & Academic-Based Competitions

I attach a copy of my University Medal for Highest Academic Achievement (ie. coming top of my year). This is *not* certified as the original is in England and I could not get it sent over in time.

I was honoured at JavaOne 2006 (an industry conference) for my contributions to the Java Development Kit (JDK). I attach a printout of the Web page with my name and the URL highlighted.

Section 13: Demonstrated Research Capacity

I believe I have 'gained leadership in a profession'. The Java platform is governed by a body known as the Java Community Process (JCP). The JCP elect board members, and the board members approve Expert Groups to research and formalize industry specifications for future versions of the platform. For any given specification, Expert Group members are selected by the Specification Lead based on merit.

I serve on the Expert Group for Java Specification Request 299 (JSR-299), which is a specification to unify Web application development by closing the gap between Java Server Faces-based front ends and Enterprise JavaBean-based back ends. I serve on this group as an individual, alongside companies such as Google and Oracle.

I attach a printout of the JSR-299 Web page with my name and the URL highlighted.

The Expert Group recently released our Early Draft Review (one of the milestones of the JSR process), and I received special mention from the Specification Lead for my contributions to date.

I attach a printout of the special mention with my name and the URL highlighted.

Section 14: Relevant Work and/or Professional Experience

I have over ten years professional experience in the complete software development life cycle across a broad range of technologies. I have developed software for all types of organisations, from local business to multi-national corporations in the United Kingdom, the United States, and Australia.

I attach my resume.

Section 15: Research Experience

My final year, individual dissertation for my BSc (Hons) in Computer Science in 1996 involved:

- proposing an original idea
- researching and implementing the idea in software
- writing a 50,000+ word dissertation of findings

I proposed exploring the idea of a 'distributed, free-form, multi-user spreadsheet': I proposed that a spreadsheet could be abstracted down to a single cell which had knowledge of both which cells it relied upon (to gather input for its calculation) and which cells relied upon it (to notify whenever it updated).

The idea was that a spreadsheet abstracted down to this lowest, single-cell level could then be recombined into larger spreadsheets. These larger spreadsheets, rather than being constrained within a single file (like traditional spreadsheets) could be located across multiple files in multiple locations. The resulting spreadsheet would be inherently distributed and highly multi-user – there could be as many users as there were cells.

An example use case would be: a landlord writes a letter to his tenant informing them of water rates. The water rate value could be a cell embedded into the sentence of his word processed document. Because it is a single cell, it could be embedded within the flow of the text and formatted indistinguishably from the rest of the paragraph (freeform). It could be updated in real-time from a cell on the water board's Web site (distributed). Finally, both the landlord and the water board could update their cells independently (multi-user).

I researched and developed prototypes for this concept. Much work was directed into developing a loosely coupled API and communication mechanism between cells. Particular challenges centred around detecting infinite loops in the overall spreadsheet: because each cell operated autonomously, there was no entity with an overall picture of the spreadsheet that could easily identify infinite loops. I proposed a solution that involved carefully monitoring the *order* of requests and responses to distinguish infinite loops from normal activity.

I implemented the prototype as a Microsoft COM+ component using Microsoft Visual C++. The finished single cells could be embedded within the text of Microsoft Word documents, on Microsoft PowerPoint slides, or in any COM+ container.

The Web was in its infancy in 1996, but Microsoft Internet Explorer 3 had just been released which allowed COM+ components to be embedded within Web pages (the technology was later renamed ActiveX). Ultimately, this allowed me to demonstrate a distributed, free-form, multi-user spreadsheet that dynamically updated calculations from Web sites into Word documents.

The finished dissertation was in excess of 50,000 words (not including code), and detailed my exploration and findings from initial proposal through to working prototype. The work was highly regarded, earning me a First (76%) and ultimately contributing to my Award for Highest Academic Achievement of the year.

Section 18: Academic Referees

Due to my industry focus over the past ten years, I do not have current academic referees.

I have nominated instead my technical lead from a previous employer, and my proposed PhD supervisor Robert James Steele, with whom I been liaising over the past several months.

My tutor for my BSc dissertation was Mark P Jones, who has since moved from the University of Nottingham to become Associate Professor at Portland State University in Oregon, USA. His e-mail address is mpj@cs.pdx.edu, though I could not reach him in time for this application.