Developing Usability Engineering Tool Suites: Towards Identifying Productivity Gains

ABSTRACT

This paper explores a method for bringing together artifacts from different usability engineering (UE) and software engineering (SE) tools via a shared model. This shared model understands the connections between various artifacts, especially user interface designs, facilitating traceability of any artifact back to the requirements which define it, and creating productivity gains by aligning UE and SE activities.

This unique approach to linking all of the artifacts in the software development process increases team productivity by making artifacts more accessible and contextual for team members. Through the model, teams are able to tackle problems that are difficult without a model-based approach, such as: determining which aspects of a project will be affected by changes to any single component of the design, ensuring that when a change is made to the user interface design on one platform (e.g., web) the changes are propagated in a consistent manner to other platforms (e.g., Android app on a phone), and managing the development of thousands of new user interfaces (e.g., on a new ship design). The principles are illustrated through a tool suite called LiquidApps® for graphical user interfaces, but the principles apply to virtually any type of interface.

INTRODUCTION

Software development processes are continuously evolving to improve the speed and quality of the software they produce. In this ever-changing environment, it is important that development teams remain agile enough to adapt to new technologies which support these evolving processes. This is especially true with usability and software engineering tools since they are capable of speeding up the evolution of their own design.

Best practices for software development (e.g., modeling, documentation, and testing) (1) have expanded over the years to include new artifacts such as task models, user stories, use cases, and user interface designs. At each stage in the software development processes, these artifacts are produced and contribute to the overall design. A usability engineer may need to work with multiple artifacts to conceptualize a design but the existing tools make working with multiple artifacts at once and keeping those artifacts synchronized complicated because they treat the artifacts as separate steps.

Consider a task model artifact being used by a software developer to code a process; if the task model is filed away after the code is written, and subsequent engineering changes are made to the code as the system evolves, there may be no incentive to update the task model to match code changes, and thus the task model’s contribution to the SE process is ended. Changes made later in development might fail to take the task model into account, resulting in usability problems in the final product. Figure 1 shows how artifacts produced at each stage need to be reinvented or resynchronized at the next stage.

Software developers are more productive and more efficient when there is an incentive to keep
artifacts in sync. This work explores the premise that developers will benefit from a tool which captures artifacts from across the software development lifecycle and presents their added value to designers and developers in the appropriate context during all stages of development. For instance, a change made to a user interface design may flag all associated requirements as needing to be manually re-verified.

Such a system works between the seams of the artifacts, linking all artifacts together to increase productivity and produce a stronger final product. Individuals performing both UE and SE see increased productivity when working with linked artifacts versus working with independent artifacts. For example, if usability assessment identifies a problem, and a UE stakeholder wants to implement a remedy that requires modification of transitions between user interface screens, then via linked artifacts we can identify how the planned change affects requirements for the requirements engineer, task models for the system engineer, code for the software engineer, and test cases for the test engineer.

**A SOLID FOUNDATION**

Representing and linking artifacts from across the software development process requires a system which is flexible enough to adapt to changes in the artifacts being represented, while providing the scalability and collaboration needed by teams performing complex software development. The Eclipse Modeling Framework (EMF) provides a solid foundation for generating such complex models. By designing the model on a proven framework, the model scales and adapts to future software development processes.

At its core, the model represents an object-oriented code structure, independent of any language or platform. Linking is built directly into the model to facilitate the attachment of metadata to arbitrary artifacts and objects within artifacts. This metadata is stored in the model and effectively customizes the model to reflect a particular Application Programming Interface (API) for platforms and platform-agnostic artifacts such as requirements and task models.

The model is called Fusion because it incorporates all aspects of the application design process: user interface design; requirements gathering; task modeling; code generation; documentation; and testing.
USER INTERFACE MODELING

The number of user interface frameworks has exploded in recent years (2) with new mobile platforms and web-based frameworks designed for mobile development. Some of these web-based frameworks are jQueryUI (3), Eclipse Rich Ajax Platform (RAP) (4), and Google Web Toolkit (GWT) (5). The challenge of designing a user interface design tool for mobile platforms is no longer purely about simplifying the experience; rather, it is now about building a user experience which works consistently across platforms.

Fusion is a meta-model, and is therefore completely platform agnostic. One must add (or contribute) models to define individual platforms to the Fusion model, including their components (or widgets), component properties, the relationship of each widget to a generic layer, and specialized property editors for designers.

Harmonia has designed a meta-model using the Eclipse Modeling Framework (EMF) (6), and created a desktop design tool called LiquidApps (7) (Figure 2) to provide a user interface to the Fusion model. The company also defined models for several popular platforms: Android, Java Swing, and RAP.

A benefit of using the meta-model approach is that a single tool can work across multiple platforms when that tool is defined to operate using only the meta-model. This is quite different than traditional Graphical User Interface (GUI) design tools, which are inherently tied to a specific target widget set (e.g., Java Swing, GWT) and the tool generates code or an intermediate language such as User Interface Language (UIL) (8), Extensible Application Markup Language (XAML) (9), or User Interface Markup Language (UIML) (10).

Because these tools store their user interfaces in artifacts specific to their respective platforms instead of a meta-model, the designs are just as immobile when it comes to transitioning to new platforms as if they were static images.

The model-based approach described by this paper makes it possible to design user interfaces for all platforms using the same tool without the caveat of limiting a design to only the common capabilities of all the supported platforms. The tool offers a consistent design environment.
for different platforms by providing a central binding meta-platform called Generic. The purpose of Generic is to act as a common ground for all of the platforms contributed to the tool while still allowing design to the access the full capabilities of each individual platform.

When a platform is contributed to LiquidApps, it describes its relationship to the Generic meta-platform shown in Figure 3. For example, Generic includes the concept of a widget called a button, which in turn features a property called text. Platform contributors may now define the Java Swing widget JButton to be equivalent to a Generic button, and map the JButton's text property to the Generic button's text property. When the Android platform is contributed, the Generic button's text property is mapped to the Android Button's text property. The tool need only recognize that there exists a widget paradigm called button and that it has an editable property called text in order to support consistent editing controls and platform-to-platform translation across platforms.

This approach is called platform-agnostic design; that is, relationships between two platforms can be described as the combination of their individual relationships to Generic. Thus tooling can be developed which examines the relationships to generic in order to support translation of an entire project from one platform to another. Since not all platforms support the same set of features, not all features are translated, but a significant productivity gain is realized for teams faced with transitioning a large project to a different platform.

**REQUIREMENTS**

User interface design is improved in both usability and capability via the model-based approach. The model-based approach also provides significant improvements in requirements management and analysis.

The Fusion meta-model is not limited to describing user interface components. It covers all lifecycle artifacts, including requirements. Each requirement may be linked to other components within the Fusion meta-model, such as tasks in a task model, or widgets in a user interface. Components in the Fusion meta-model connected like nodes in a graph. These links exist between the seams of the artifacts and contribute more value to the model than the
requirements were ever capable of contributing alone.

The added value of linked requirements may be seen first when designing a user interface in LiquidApps. Each time the designer selects a widget in the user interface design, all the requirements available in the model are filtered to only those which are attached to the selected widget and are displayed in a view below the user interface design canvas. Selection-based filtering of requirements means that the designer is presented with only those requirements which are contextually relevant to the widget they are currently designing. For example, a designer editing a label will be aware that a manager has added a requirement while editing that label rather than finding out later when reviewing requirements.

The increase in productivity and quality here is expected to be significant. Designers are no longer forced to leave their design tool to search for requirements in a separate tool and since they are presented in context, there is no mental shift between designing and verifying requirements.

The second place where linked requirements add significant value is when they are included in artifacts generated by LiquidApps. Even though some artifacts such as source code and documentation are stored in conventional file systems outside the Fusion model, they still benefit from linked requirements as described below.

Source code generation for user interface designs is a popular feature of many user interface design tools like NetBeans (11) and Visual Studio (12). A tool like LiquidApps that performs source code generation and has access to linked requirements through the Fusion model is able to incorporate the requirements into the generated source code as comments. These comments follow the code through the normal development process and provide the same value they did to the designer; as a result, the developer is more likely to accurately meet those requirements and do so without wasting time switching tools and searching a requirements database.

Requirements documents are a form of documentation generated by requirements management tools to make it simpler to carry requirements forward in the software development process. As we have seen with the Fusion model, these documents are not strictly necessary to carry requirements forward.
Documentation of a system is an important tool for communicating a design to stakeholders.

In order to generate such documentation, one must consider the value it provides and how to effectively communicate the information in document form. Harmonia anticipates that contextual display of requirements will be valuable to stakeholders as well designers and developers. Since the documentation may be generated from the Fusion model, it may be organized by user interface design, with the requirements attached to objects in each user interface displayed below the user interface.

Because the requirements document is generated, requirements which are attached to elements in multiple user interfaces may repeat throughout the document. With requirements appearing in context in the final document, the content is much easier to use as a reference document without needing to refer to an index or searching to find desired information.

**TASK MODELS**

Harmonia’s model-based approach also extends to Task Modeling. Task models are used both in the SE world (e.g., as activity diagrams in the Unified Modeling Language [UML]) and in the UE world as a powerful process for determining how end users will interact with an application based on how they currently perform their job function. Breaking down a potential user’s job function into small activities and decisions gives designers a much stronger understanding of the needs of their users. The problem with current task modeling applications like Visio or Rational’s products is the same as with requirements management tools. Commercial tools have no links from task models to user interfaces, leaving the designer to reference the task model manually while designing a screen the first time. Also from that point forward, the task model’s connection to the user interface design is lost.

In the Fusion model, links are maintained between user interface components and task model activities. These links are consulted in LiquidApps each time a designer requests that a user interface component be deleted. If the user interface component is linked to an activity in a task model, deleting the user interface component would mean that the end user would no longer be able to perform that activity using the application. LiquidApps prevents this situation by informing the designer of the conflict and requiring the designer to confirm the deletion.

There are other situations involving these links which may be analyzed by a tool using the Fusion model. LiquidApps presents designers with Audit Reports which use these task model links to determine if there are any activities which do not have an associated user interface component. The end user will not be able to perform these activities in the final design unless they are attached to some user interface component. There may also be user interface components which do not have associated task model activities. These could be extraneous controls in a user interface design, or more interestingly, these user interface components could be discovered activities missing from the Task Models.

It is possible that a designer would remember to go back and update the task models, but it is more likely that the task models would simply remain out of sync with the user interfaces. In the Fusion model, links between user interfaces and task models help to identify these situations, making it more likely that the designer will update the task model to remain in sync with the user interface design. This permits generation of far more accurate documentation of the system.
and can help identify usability problems early in the design process.

REALIZATION OF FUSION MODEL
LiquidApps is a realization of the Fusion model and permits evaluation of the productivity and usability improvements which can be made by using a model-driven approach. Since the Fusion model is designed to work with artifacts for any software design platform, LiquidApps supports design for user interfaces across multiple platforms including Java Swing (Desktop), Android (Mobile), and Eclipse RAP (Web). The Fusion model is focused on incorporating artifacts from many sources and as such, LiquidApps synchronizes requirements with DOORS, imports task models from Imprint, and imports UIL (User Interface Language) Motif user interfaces.

RELATION TO EXISTING WORK
Existing Commercial and Open Source tools tend to focus on producing a single artifact of the software development lifecycle – such as requirements being managed by DOORS; code produced using an IDE; models designed using Visio; etc.

However, the idea of bringing together artifacts from across the SE lifecycle is not new. For example IBM’s Rational tool suite links requirements to UML models and permits model-driven code. Serena Prototype Composer (13) combines UI mock-up capabilities with business workflows to create prototype simulations.

On the UE side, there has been little work to link artifacts. The basis of the Fusion meta-model design comes from research on model-driven human computer interface design (14), which examines linking task models to user interface designs. There is one commercial product that links requirements to web-based user interface design, namely iRise (15). The tool attempts to involve stakeholders in the software development process more closely while focusing on user interface designs with attached requirements. However, iRise is designed for a specific platform, the web, in contrast to LiquidApps’ use of models to work cross-platform and does not offer the linking of artifacts between SE and UE that the model-based approach permits.

CONCLUSION
This paper advocates a model-based approach to building UE tools for several reasons. First, models permits adaptation of tools as SE processes and technologies evolve. The Fusion model adapts to incorporate the new artifacts being invented and ensures their value contributes to the final product by making those artifacts available and relevant throughout the development process. The use of a meta-model that has no inherent understanding of any

![Figure 5: Artifact Reuse with Fusion](image-url)
platform’s user interface components can be instantiated to represent virtually any platform as shown in Figure 3.

Second, linking artifacts in the Fusion model ensures traceability between artifacts and reduces overhead in using those artifacts to create better software. While the artifacts themselves are not significantly changed by being placed into the Fusion model, their value to the project is amplified by the links defined between them. Keeping these artifacts accessible throughout the development process makes it easier to keep the artifacts in sync with the final product, which results in higher productivity and better documentation for the design. The Fusion model allows for reuse of artifacts, including user interface components, across products within an organization.

The change in process introduced by the Fusion model provides incentive to ensure artifacts are in sync. In building LiquidApps to realize the model, it was determined that a tool can capture and link artifacts from across the software development lifecycle. To realize the full value of these links, a tool can present them to designers and developers in the appropriate context during all stages of development minimizing shifts in focus and maximizing productivity. The Fusion model acts between the seams of the artifacts, linking everything together, increasing productivity, and producing a stronger final product.

LiquidApps is a commercially available product which can be downloaded from (7).

ACKNOWLEDGEMENTS
LiquidApps was funded in part by MDA, NAVSEA05H, NAVAIR, ONR, PEO-IWS, PEO Ships, and US Army C2D.

REFERENCES


Christopher R. Eagan is a software engineer at Harmonia working primarily on the LiquidApps tool suite. The research he has done at Harmonia related to usability, collaboration, and modeling has contributed many of the ideas behind the Fusion model concept and realization.

Dr. Marc Abrams, Ph.D., is Chief Technology Officer at Harmonia, and co-founded Harmonia originally to bring research on human-computer interaction into practical tools. He has spent over fifteen years in pioneering research, design and development of new software tools to assist with developing highly usable human-computer interfaces for GUIs, voice, and mixed model interfaces. He was an Associate Professor of Computer Science at Virginia Tech.

Brendan D. Berry is a software engineer at Harmonia working primarily on the LiquidApps tool suite. As the primary architect behind the design of the Fusion model and originating many of its core ideas, he has created a system which supports the many goals of LiquidApps.

Stuart C. Mackey is a software engineer at Harmonia working primarily on the LiquidApps tool suite. He originated the idea of linking between artifacts which is the primary reason for the productivity gains delivered by the Fusion meta-model.