Reverse Engineering End-user Developed Web Applications into a Model-based Framework

Yogita Bhardwaj

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Science

Manuel A. Pérez Quiñones, Chair
Chris North
Marc Abrams

(May 09, 2005)
Blacksburg, Virginia

Keywords: Reverse Engineering, Model-based approach, End-user programming, Web application development, User Interface tools

Reverse Engineering End-user Developed Web Applications into a Model-based Framework

Yogita Bhardwaj

ABSTRACT

The main goal of this research is to facilitate end-user and expert developer collaboration in the creation of a web application. This research created a reverse engineering toolset and integrated it with Click (Component-based Lightweight Internet-application Construction Kit), an end-user web development tool. The toolset generates artifacts to facilitate collaboration between end-users and expert web developers when the end-users need to go beyond the limited capabilities of Click. By supporting smooth transition of workflow to expert web developers, we can help them in implementing advanced functionality in end-user developed web applications. The four artifacts generated include a sitemap, text documentation, a task model, and a canonical representation of the user interface. The sitemap is automatically generated to support the workflow of web developers. The text documentation of a web application is generated to document data representation and business logic. A task model, expressed using ConcurTaskTrees notation, covers the whole interaction specified by the end-user. A presentation and dialog model, represented in User Interface Markup Language (UIML), describe the user interface in a declarative language. The task model and UIML representation are created to support development of multi-platform user interfaces from an end-user web application. A formative evaluation of the usability of these models and representations with experienced web developers revealed that these representations were useful and easy to understand.

Acknowledgements

I am greatly thankful to my advisor Dr. Manuel A. Pérez-Quiñones for his immense support and able guidance throughout the course of this research. I am grateful to him for introducing me to an interesting area of research that I thoroughly enjoyed working on. Despite an extremely busy schedule, I received complete attention and timely guidance that helped me complete this thesis.

I also thank my committee members, Dr. Chris North and Dr. Marc Abrams, whose inputs and suggestions helped me in reasoning about the very foundation of this research.

I would like to thank my colleague Jochen Rode for giving me important suggestions and feedback on my work. I have learned immensely from him during the course of working with him.

Special thanks to my family whose unending love and support sustained and encouraged me throughout my graduate studies. I would not have been able to embark on this journey without their unassailable faith in me. I am particularly grateful for their invaluable suggestions and feedback during the writing of this thesis. I dedicate my thesis to them.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The World Wide Web has become a platform for more than just static content publication. We daily experience many web-based interactive applications like online surveys, shopping and content management applications. With the wide acceptance of such applications, we can expect an increased interest in developing even greater variety of web applications in the near future. At present, web application development is limited mainly to professional programmers. The main reason behind this is the complexity imposed by multitude of web technologies involved in creating a single web application like HTML, CSS, JavaScript, PHP, MySQL etc. [63], plus a myriad of other technology related issues (e.g. session management, cross-browser compatibility, security etc.). However, professionals, but non-programmers, often use computing tools as means to accomplish their work. Often, this entices them into end-user development activities. For our work, this creates a requirement for devising methods and tools that help an end-user in developing interactive web applications.

A research prototype, Click (Component-based Lightweight Internet-application Construction Kit), developed at Virginia Tech, explores the feasibility of end-user web engineering [62, 63]. The tool is designed to allow non-programmers to create web-based data collection, storage and retrieval applications. The main goal of Click is to hide technological complexities underlying web application development while offering a wide range of functionality, typical for small-scale web development. Myers et al. [3] mentioned that an important goal for future user interface tools is to have "a low threshold and a high ceiling". This means that the tools should be easy to learn and use (low threshold) as well as provide a lot of power to the users to accomplish broad range of activities (high ceiling). A shortcoming of most end-user tools is the tradeoff between expressive power and ease of use. Almost by definition, end-user development tools must have a low threshold (easy to learn) but suffer of a low ceiling (limited functionality). Our work seeks to remove the low ceiling of the end-user web development tool, Click, by smoothly transitioning the web application from the end-user developer to a professional developer.

When end-users face a ceiling on what they can accomplish with any tool, they often rely on some local expert who may be able to add advanced functionality to their applications. Spreadsheets stand out as one of the most successful tools in empowering end-users and unleashing their creative talent. Nardi et al. [1] conducted ethnographic interviews with various spreadsheets users. The study showed that spreadsheet development was a collaborative work by people with different levels of programming and domain expertise. Nardi et al. emphasized how non-programmers, who may be experts in their domain but lack programming skills, turn to expert developers for assistance with more sophisticated programming needs. "Spreadsheet co-development is a rule, not the exception". Similarly, we envision end-user and expert developer collaboration in web application development and there is value in providing means to make this collaboration successful.

1.1 Motivation

This research focuses on transitioning an application developed by an end-user to expert programmers for further development. End-user web application development is usually an informal process. Small interactive web functionality may be added to large static websites on an as-needed basis (e.g. adding a simple feedback form). In such cases, the end-users may find it easier to follow an ad hoc development approach and deliver the results rather than create any useful documentation or design models before beginning implementation. Even if end-users create some documentation, it will most likely be an informal one, reflecting their lack of technical expertise in this area. Understanding an end-user developed application, without proper documentation, can require a considerable effort from the expert developer. Even as the developer spends time understanding the application, no effort may be made to create formal documentation to be used by other developers who will then repeat the effort. General software engineering principles suggest that this rush-to-code approach may be appropriate for the immediate needs of an end-user developed application but does not scale well for larger applications. This situation emphasizes the need to create artifacts that aid software comprehension to support smooth transition of an end-user developed application to an expert developer.

Berti et al. [59] identified the need for supporting end-user development that go beyond desktop applications (e.g. small devices). In their work, they propose that end-users develop applications using familiar and easily understood representations. The research challenge is to transform these representations into precise user interface specifications. The research presented in this thesis focuses on this transformation into a suitable representation that is not limited to the desktop metaphor. The web application domain is now facing the challenge of supporting multi-platform user interfaces (MPUI). Different devices, like WAP phones, PDAs, handhelds, are now capable of accessing the Internet anytime anywhere. There is a strong need for making web applications available from these different devices. The main challenge here is the task of building a user interface for the same application for all of these platforms. This is an inherently complex task as each device has its own form factor, technological limitations and context of use. A developer needs to control many variables for each platform while maintaining consistency and usability across all the different versions [32].

In our work, we provide support for model-based user interface design. Model-based framework for user interface design has recently captured the attention of the research community for its capabilities to address the issue of MPUI development. By converting end-user developed web applications to a model-based framework, we can facilitate further development of the web interface for multiple platforms.

1.2 Research Goals and Approach

The main goal of this research is to provide a toolset for reverse engineering web applications developed by end-users. The overall approach followed is to design the toolset to automatically generate abstract representations of a web application built by an end-user developer.

The four artifacts generated by the toolset include a sitemap, text documentation, a task model, and a canonical representation of the user interface. The sitemap is automatically generated to support the workflow of expert developers. The text documentation of a web application is generated to document data representation and business logic. The sitemap and text documentation are designed to facilitate collaboration between an end-user and an expert developer and serve as a tool for

discussion among them. These representations aid expert developers in easily understanding already created web applications so that they can assist the end-users in implementing features not provided by an end-user development tool. This will help in addressing the limitations of an end-user development tool faced by the end-users. A task model, represented using ConcurTaskTrees (CTT) notation, covers the whole interaction specified by the end-user. A presentation and dialog model, expressed in User Interface Markup Language (UIML), describe the user interface in a declarative language. Task model and UIML representation are created to support development of multi-platform user interfaces from an end-user web application.

Figure 1 shows how an end-user developed web application will be converted into four abstract design representations by the reverse engineering toolset, i.e. sitemap, documentation, task model and generic UIML for desktop family. Each of these representations will contribute to the goal of facilitating end-user and expert developer collaboration and development of MPUIs for the web application. Figure 2 shows a framework for building MPUIs for an end-user developed web application using the task model. It is an extension of the framework suggested by Ali et al. [34]. A separate UIML representation will also be generated by the reverse engineering toolset which is specific for the desktop family and will very precisely represent the end-user developed web application. This UIML representation will be used by the expert developers for further development of the web applications on the desktop platforms. Using the tool support available for UIML, the generic UIML can be rendered on to various platforms in the desktop family [32, 66]. Figure 3 shows how the generic UIML for the desktop platform can be used to generate user interfaces multiple desktop platforms, e.g. HTML based interfaces and JAVA interfaces.

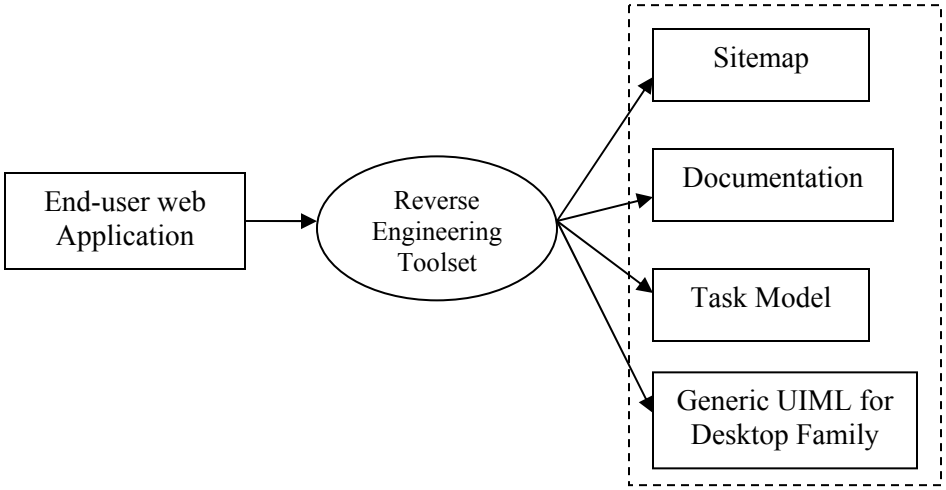


Figure 1: Using the Reverse Engineering Toolset to produce abstract design representations from an end-user developed web application

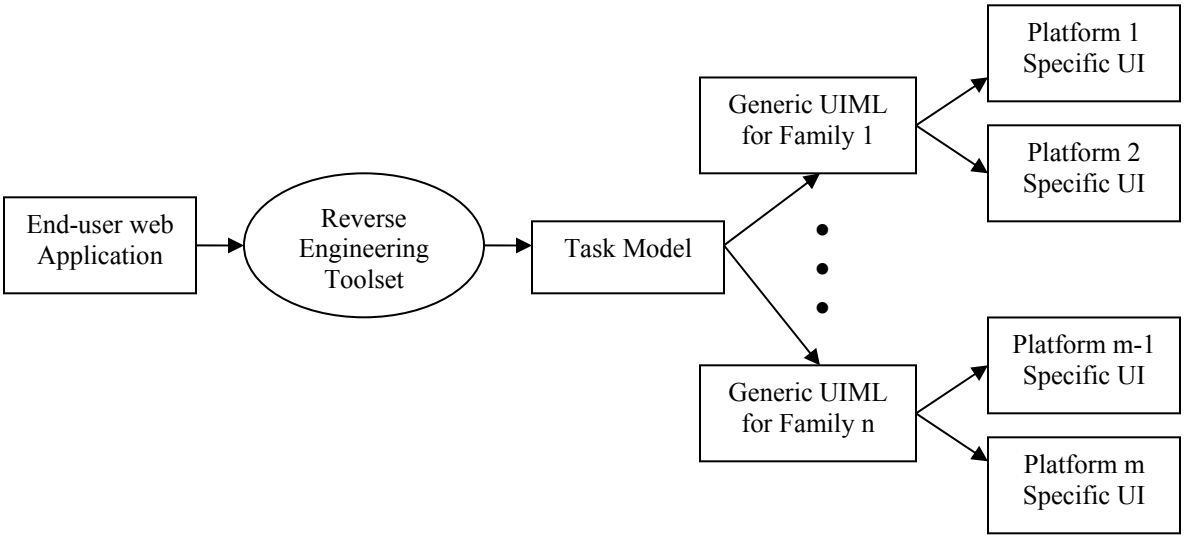


Figure 2: Framework for development of MPUIs for end-user developed web applications, extension of framework for developing MPUIs from Task Models by Ali et al. [34]

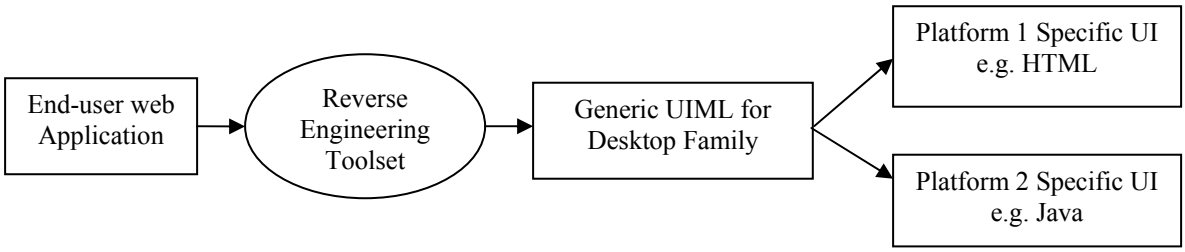


Figure 3: Generating MPUIs for desktop platforms from an end-user web application

1.3 Thesis Outline

Chapter 2 is a review of the literature on user interface design using the model-based framework, abstract representations and tool support available for forward engineering of MPUIs. The literature reveals that task model is considered to be an important starting point in most forward engineering approaches. Task models have also been considered useful in usability evaluation of UIs. The CTT [39] notation is most widely used to represent task models. The reverse engineering toolset for Click will generate a task model for a web application using the CTT notation. The transformation of an end-user developed web application to a precise specification for MPUI development is done by creating a User Interface Markup language (UIML) [31] representation for the web application. UIML is a canonical representation for UIs with a goal of making UI development possible for even non-programmers. UIML also has tool support for authoring and rendering user interfaces [66] making future development using the UIML representation for the web applications easier.

Chapter 3 provides an overview of web applications developed in Click and a description of the reverse engineering toolset that automatically creates a sitemap, text documentation, task model and UIML representation. Chapter 4 presents detailed results from a formative evaluation of the generated representations. This informal usability evaluation with expert web developers was performed to evaluate whether the reverse engineered abstract representations were useful as well as easy to understand for the developers. The web developers recruited for the evaluation showed a positive response towards the entire development approach. Overall, the generated abstract representations were successful in satisfying the research goals. Chapter 5 summarizes the work done and identifies potential for future research.

Chapter 2: Literature Review

This chapter introduces the model-based framework for user interface design and various tools and methodologies developed that address some of the challenges in user interface design. Specific focus is on the development of MPUIs. Various forward engineering approaches for generating MPUIs are discussed followed by a synopsis of the related work on the reverse engineering of web applications. This chapter also introduces Click, an end-user development tool, which is used in this research. A detailed discussion on the architecture and features of the web applications created using Click is presented.

2.1 Requirements for reverse engineering

Most web applications are developed under tight schedules and in a rapidly evolving environment. The development is often ad-hoc in nature and the applications are poorly structured and poorly documented. Maintenance of such applications becomes problematic once the complexity of the web application grows. Creating appropriate design and architecture models is the key to managing this complexity and supporting evolution of web applications. Researchers have identified the need to reverse engineer already existing web applications into abstract design models. Two different streams in reverse engineering web applications exist. One approach reverse engineers a web application into UML models [57]. The other involves reverse engineering web applications into a model-based framework which is specifically designed for user interfaces.

Although UML has been widely accepted as a standard language to specify software models, it was not designed with user interfaces in mind. This is precisely the reason why no particular model in UML is very successful in capturing details of user interactions and interface design [5]. Also, the traditional software engineering approaches have not focused on user interfaces very much. Moore [6] emphasizes the need to focus on the user interface of an application during the software maintenance and reengineering to new platforms. It is interesting to note how an average of 48 percent of the code for an application is devoted to its user interface [7] and a huge amount of time

is spent on different phases of UI development. The survey on user interface programming by Myers and Rosson [7] also reported that interface builders reduce the time spent and amount of code required to be written by the developers. It is therefore important to design tools and techniques that effectively support development and maintenance of user interfaces. This applies to web application design as well.

A workplace study conducted by Landay et al. [2] showed that web interface designers build websites by creating low-fidelity sketches and refine the interface through further brainstorming. An important finding of this study was that sketching an interface design comes naturally to website designers and is useful because it provides abstraction over implementation details letting them focus on overall design. This study showed that sitemaps were used by the web designers on a regular basis. They used sitemaps as a primary artifact for information design. Sitemaps usually consist of blocks representing individual pages and limited navigation information represented through lines and arrows. Sitemap can, therefore, be an important and useful artifact that a reverse engineering approach could create for a web application.

Moore [6] lists certain requirements for the ideal representation for reverse engineering user interfaces such that the recovered abstract design representation is correct and complete. The abstract design representation should provide enough detail to allow appropriate user interface techniques to be chosen when building UI for the new domain. The representation should be such that forward engineering requires little effort or can be done completely automatically. Sitemap by itself may not be adequate for this purpose and we would need additional representations capable of addressing the requirements set forth by Moore. Model-based user interface design approach seems to be a good choice for these requirements.

Model-based user interface design, researched heavily in the early 90s, sought to provide declarative models that ease the development of user interfaces. The focus of the research was mainly to automatically generate high quality user interfaces from abstract representations. This technique did not find wide acceptance because the automatic UI generation tools were based on many heuristics that made the generation process unpredictable and hard to control [3]. But, model-based user interface design has again

caught interest of the research community as it looks promising in addressing the current challenges of developing user interfaces for multiple platforms.

2.2 Model-based User Interface Development Paradigm

Model-based user interface development paradigm is an approach for constructing user interfaces using a set of declarative models that capture various aspects of a user interface at a semantic level. A designer creates a UI using abstract representations that are converted into concrete representations through an automated or mixed initiative approach. The four main development steps are as summarized below:

- *Tasks and concepts:* This level represents high level tasks that users can perform and the data and operations the application can support. These are represented using the task models and domain models.
- *Abstract User Interface:* This is a specification of the structure and behavior of the interface in terms of interaction objects and presentation elements. This abstract specification may be context (user, platform and environment) and modality independent.
- *Concrete User Interface:* Abstract user interface is concretized for a particular context of use into this representation. At this level of abstraction, interaction widgets are identified and interface navigations are defined. This representation is independent of any computing platform.
- *Final User Interface:* This is the final implementation that runs on a particular platform and is ready to be deployed.

2.3 Models

Many models have been defined for capturing different aspects of a user interface. The definitions given below are somewhat generic representing a basic idea on which these models are built. Different methodologies proposed under the model-based user interface development paradigm have used them differently according to their approach.

Task Model

Task models describe tasks that users need to perform in order to reach a goal. Task modeling is now considered as an important starting point in the model-based design to develop interfaces that effectively support user tasks. Task models are sometimes represented as a combination of the user and the system task model [39], i.e. a view of how users would perform the activities as well as how the system perceives the tasks are performed. In the interface design process, task models help the designers to identify how to design techniques that would best support the tasks.

Domain Model

Domain model refers to the application's business logic and data objects that are manipulated by the users during their interaction with the system. The model should represent relationship between various objects of a system and their different attributes. Benyon [49] surveyed the domain models prevalent in the user interface design while describing strengths and weaknesses of each. Benyon takes a holistic approach by defining the domain model to be an abstract representation of the system. He also claims that task models, object oriented models and data models all are candidates for a domain model. In model-based tools, domain models have been defined and used in a variety of ways.

User Model/Context Model

This model represents a run-time model of the application. A context of use is defined as a combination of environment, user and platform, and can greatly affect the usability of the application. Environment model specifies the environmental conditions like lighting conditions, sounds etc. affecting the interaction with the system. User model would specify the target users of the application and their preferences. Platform model describes the software-hardware profile on which the application is run. This is a very dynamic model and may affect other interface models like task model, presentation model and the dialog model at run time [23].

Presentation Model

A presentation model specifies how the tasks and concepts are presented to the user in terms of interface widgets that aid users in interacting with the system. This model also specifies attributes i.e. style and the layout of these widgets on the interface.

Dialog Model

The dialog model defines the way in which a user interacts with the interface sometimes referred to as the behavior of the interface. It represents the actions that a user may initiate via the interaction widgets and the reactions that the application communicates via the interface.

The task and domain models are considered to be abstract models that are specified at the tasks and concepts level of UI development process. Presentation and dialog models can be defined at both an abstract and concrete level to precisely specify a user interface and interactions. The reverse engineering toolset focuses on generating the task model in order to capture the abstract interface design of a web application. It also generates the presentation model and dialog model from the web applications at a level of abstraction that is independent of final UI implementation toolkit. The following sections present a survey on other specification formats for these models and various modeling tools, IDEs that help in generating UIs using these models.

2.4 Modeling tools and IDEs

Szekely [4] presents a retrospective on the model-based approaches and how these have evolved over time. It is interesting to analyze future problems that need to be addressed while keeping this study in perspective**.** The foundation of model-based tools was based on the work in UIMS (User Interface Management System) which started in early 80s. The main idea behind UIMS was to provide a separation between the user interface and the underlying application logic. These two components were specified by a UI designer and UIMS would manage the interaction between the user and the application. The first generation of model-based approaches focused mainly on user interface specification at a higher level of abstraction to streamline the UI design process that was mainly an ad hoc process till then. These tools provided an environment for

creating and editing abstract specifications and automatically translating this specification into an executable program or interpreting it directly at run-time to generate the user interface.

The second generation tools introduced the concept of user tasks in the interface design process. The task centric or user centric design focuses on supporting the user's activities and interactions with the application in the most effective manner. Another important research question these modeling tools focused on, was how an interface development environment can let UI designers control all the aspects only when they needed to, without over burdening them. Paternò defines the third generation of tools and methodologies [5], which have particularly focused on the current challenges of supporting multi-platform, multi-context and multi-modal UI development. These tools and methodologies have been built upon the general concepts from the second generation of tools but extend them to address the multiple of contexts of use of an application in the UI development process. The third generation tools comprise of authoring environments, XML based languages and reverse engineering tools. This research also falls in the third generation of tools.

In the following sections an analysis of various tools and their approach towards model-based design is presented with more emphasis on the third generation tools and methodologies. Figure 4 represents an overview of the significant projects and the year in which the work was published. Appendix E is compiled list of model-based tools and references to these tools.

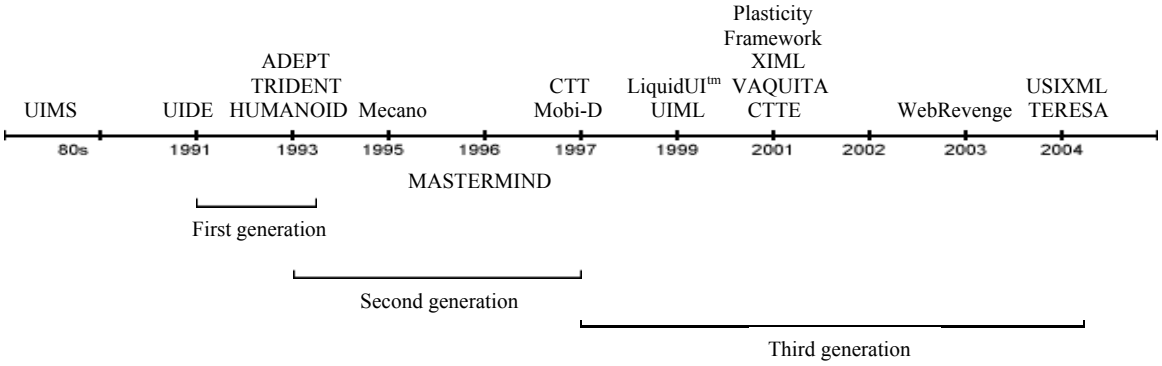


Figure 4: Evolution of model-based tools and methodologies

2.4.1 First Generation

UIDE

UIDE (User Interface Design Environment) [8, 9] is among the early model-based systems. UIDE provided a mechanism to specify a data model and a mapping from the data types to interface widgets or the presentation model. It defined a notion of pre-conditions and post-conditions on the interaction. Pre-conditions define the set of conditions, which if true, will enable interaction with the interface objects and post-conditions specify the modification to the user interface state. This in a way represents the dialog model. The main shortcoming of this tool was the automatic generation of user interfaces and not providing much control to the designer.

HUMANOID

Luo et al. [10] identified that the current UI design tools are on extremes of either completely automating design or completely relying on a designer controlling numerous aspects. They put forth a requirement for future interface design tools to strike a balance between providing control to the designer and automating the design. HUMANOID [10, 11, 12] was built with the same goal in mind. HUMANOID defines an interface model which captures an application's interface presentation, interactions on the objects and their effects. However, there are no explicit presentation, domain or dialog models. HUMANOID also provides a run time system that produces a working UI using these models. It has predefined options that control the final UI generation process but also lets the designers modify the design according to their preference. This system introduced a collaborative (or mixed initiative) development environment for model-based design where human designers are responsible for making design decisions and the system automates the implementation.

2.4.2 Second Generation

ADEPT

ADEPT (Advanced Design Environment for Prototyping Tasks) [14, 15, 16, 17] was one of the early interface designers to encourage the creation of user-centric models. The ADEPT philosophy was to begin interface design process at the most abstract level,

i.e. task analysis, and then automatically creating an abstract interface model that would best support the tasks. Using the abstract interface model it would generate a concrete interface model which can be translated into its equivalent platform dependent implementation. The design goals of the ADEPT demonstrator, the design environment, were to not force a design process on to the developers and allowing them to work at any abstraction level while keeping all the levels of models synchronized. The published literature however does not make it clear, how flexibility is introduced into the design process and what rules the automation is based on. Clearly, if the automatic translation of one model to another is not based on sound principles, it will be counter-productive for the designers.

Trident

TRIDENT (Tools foR an Interactive Development EnvironmeNT) consists of a methodology as well as a supporting environment for developing highly interactive business oriented applications [18, 19]. The main feature of TRIDENT was that it proposed a knowledge based technique for automatically generating user interfaces. Models used in this approach are task and domain models. TRIDENT makes decisions based on the information captured in these models and the predefined knowledge bases containing ergonomic rules. The system automatically makes intelligent decisions for selecting interaction styles that best support a task. It selects interaction objects based on an object-oriented entity relationship model for the domain and automatically places interaction objects on the UI. This is a very sophisticated system that provides automation, customization of guidelines on which automation is based and computerized advice on what interaction techniques to use. The system is definitely robust in what it does but it puts a burden on the designer to specify production rules, guidelines and the knowledge base upon which the automation and computerized advice is modeled. It may be more onerous to specify these rules and customize them according to the varying needs than directly implementing the UI.

MASTERMIND

MASTERMIND (Models Allowing Shared Tools and Explicit Representations Making Interfaces Natural to Develop) [13] was a follow-on work on UIDE and HUMANOID. This tool designed a representation for task models that captures not only the high level tasks but also the steps required to perform the tasks and reaction of the interface to each of these tasks. MASTERMIND also specifies an application model and a presentation model. No explicit dialog model is specified but the task model in MASTERMIND can be specified with low level details of interaction techniques, e.g. button click, menu selection etc. and effects, and the specification of actions to be performed for a particular task.

The Mecano Project and Mobi-D

The Mecano project introduced multiple levels of abstractions at which an interface is specified and a declarative language (MIMIC) to specify interface models and relationship among these models [21]. Interface models, in Mecano, included all the models defined above: user-tasks, domain, presentation, dialog and user types. It allows specifying multiple versions of interface model components to address different requirements. This specification suggested an early attempt towards solving the problem of differences in interface requirements due to variety of user base or devices the application may be targeted at.

Mobi-D (Model-based Interface Designer) is an interface development environment where designers can visualize, manage, edit and refine the interface models defined using the MIMIC language [22, 25]. Another major contribution of Mobi-D is to address the "mapping problem" [26] such that it identifies key mappings between interfaces models and proposes that developers are allowed to set the mapping for facilitating interface development.

2.4.3 Third Generation

Presenting the retrospective on the model-based interface development, Szekely [4] had identified few challenges that the model-based design community was facing then. Providing support for development of MPUIs was an important one among them.

The first and the second generation tools had shown great promise in managing the complexity of the interface design process and ensuring usability of the applications developed by using various models but they were mainly designed for standard desktop platforms and a standard set of input and output techniques. Today, computing devices come in all shapes and sizes and have different computing power. Designing user interfaces for applications that need to run on all these devices increases the variables that the designer needs to control and therefore the overall complexity of the development process. For example, screen size and resolution of the devices, operating systems and software toolkits available etc. have a significant impact on the user interface. A platform is usually defined as a combination of, hardware, operating system and software tool. MPUIs are sometimes referred to as device-independent user interfaces as well.

A variety of input modalities have been developed like speech, natural language and pen gestures. These add another level of complexity in the UI design. This category of interfaces is referred to as multi-modal user interfaces. Another important aspect affecting the usability of an interface is the context in which the interface is being used. For example, cell phones are more likely to be used for information access when the user is mobile, but stationary users may prefer using a desktop computer for the same purpose. Considering the situation in which the users are, we can imagine that they may have different requirements from the applications. A mobile user may only need access to critical information which is an immediate concern. It should not involve heavy cognitive effort as there might be other environmental factors the user need to manage while using the service. Consequently the interfaces need to be tailored for each specific context and the interface design should be able to support it. Such interfaces are referred to as multi-context interfaces or context sensitive interfaces.

Paternò [5] analyzes this paradigm in the light of Szekely's work and shows how things shaped after that. However, the paper does not clearly specify how new languages and tools have addressed the challenges mentioned above. Although his work presents a general picture of the problem, a detailed discussion is provided only for the research conducted by his group that focused on ConcurTaskTrees (CTT), a notation for defining task models and TERESA, which is a transformation-based system to provide complete development.

In the following section, an analysis on the recent technological developments in this area is presented. The main goal of the third generation tools is to provide an integrated design and authoring environment to create various models and a transformation environment that assists developers in creating user interfaces from the high level models.

Cameleon Project

Cameleon (Context Aware Modeling for Enabling and Leveraging Effective interactiON) [67] project aims to build methods and environments supporting design and development of highly usable context-sensitive interactive software systems. The Cameleon project involves some of the main research groups that are actively involved in the research of multi-target applications. Further, the research spans into the areas of plasticity of user interfaces, TERESA and USIXML. It appears largely that the research conducted under the umbrella of the Cameleon project has parallel and competing streams. Although the overall goal and problem they are trying to solve is the same.

Plasticity Framework

Thevenin & Coutaz [23] defined another terminology to identify the problem of building user interfaces that adapt to changes in the context in which they are used. "Plasticity is the capacity of a user interface to withstand variations of both the system's physical characteristics and the environment while preserving usability". The main goal was to obviate the need for complete redesign and re-implementation by providing a framework which was built upon the model based design paradigm. The novel aspect of this approach was the introduction of platform and environment model and a separation between the user-task and system task model. The framework also models the dynamic aspect of the interface where changes in platform or environment model may affect user task model. The framework defines another model called the interactor model that can be compared to a combination of a presentation and dialog model.

Calvary, Coutaz & Thevenin [24] revised the framework which is now referred to as the Cameleon Reference Framework. This framework defines various transformations that can be performed on the different levels of abstractions. Forward engineering of user

interfaces was thus defined as starting from tasks & concepts and going through a series of transformations, called reifications, to obtain more concrete interface representations. Reverse engineering is performed through a series of transformations called abstractions. Yet another type of transformation is called translation where a model in one context of use is converted into its counterpart in another context of use. Reengineering process is a combination of these transformations. This multi-directional user interface development is defined in detail by Calvary, Coutaz & Thevenin [24].

2.4.4 Forward Engineering of UIs

The main goal of the forward engineering approaches is to define formal specification formats for various models and design a set of transformations that convert one specification into another. Although, previously defined methodologies defined interface specification formats, each tool had its own proprietary format and it was aimed mainly at designing single platform UIs. Third generation tools focused on building specification formats using markup languages such that they are capable of specifying a user interface in a platform, context and modality independent way becomes important. An XML representation is inherently platform independent and provides a clean construct. XML technology has already advanced to high level of sophistication which benefits the expression of user interface design and allows for easy parsing and modification of data. Motivated from this, many XML-based languages have been proposed as the abstract interface description language. Vanderdonckt et al. [28] have summarized requirements for a powerful user interface definition language.

The remaining part of this section describes some of the significant contributions in abstract languages that have been developed for model-based design and transformation based tools that support transformations from one model to another. Web application interfaces has been an important target for all these researches.

ConcurTaskTrees

The research group of Fabio Paternò at CNUCE-C.N.R., Pisa, Italy has been involved in developing tools and methodologies for model-based interface design. An early contribution was a diagrammatic notation to represent hierarchical task models that

was primarily specified by a textual notation until that time. They developed CTT (ConcurTaskTrees) notation [39] to capture tasks of various categories, their hierarchical relationships and their temporal constraints by means of icons and operators. Tasks can be related to each other with a parent-child relationship or temporally defined siblings. CTT defines four types of tasks: abstraction, interaction, application and user. An abstraction task is a high level task defined as a combination of any of the four types of tasks. Interaction task is a task performed by a user interacting with the system using some interaction technique, for example pressing a button. Application task is a task completely executed by the application; typically in response to an interaction task. User task is a task performed entirely by the user without interacting with the system.

CTTE (ConcurTaskTrees Environment) [40] is a tool that provides a graphical editing environment to support the development of task models using CTT. CTT is an XML based notation and has widely been accepted as a method to specify task models among the model-based community. Table 1 shows the icons used to represent these tasks. Table 2 describes temporal operators defined by CTT that connect siblings in a task tree. A more detailed described can be found in [39, 40].

Table 1: Types of tasks defined by CTT and icons used to represent the task types

Type of task	Icon
Abstraction	
Interaction	
Application	
User	

Table 2: Temporal Relationships defined by CTT and the operators used to represent the same

Temporal Relationships	Operators
Optional Task	[T]
Iterative Task	T*
Choice	T1 [] T2
Order Independency	T1 |=| T2
Concurrency	T1 ||| T2
Concurrency with Information	T1 |[]| T2

Exchange	
Disabling	T1 [> T2
Suspend/Resume	T1 |> T2
Enabling	T1 >> T2
Enabling with Information Exchange	T1 []>> T2

TERESA

Paternò and Santoro proposed a complete transformation based method for building multi-platform applications [41] as a general solution to the problem tailorable to specific cases. Their approach is very weakly related to the plasticity framework. The task model is the only model they use in their approach and other steps are simply transformations into more concrete representations. The task model is annotated with information related to which platforms certain tasks are available and what object model a particular task is associated with. A more detailed discussion on how task models are annotated is given by Paternò and Santoro [42]. One can argue that on one hand putting a heavy emphasis on just one model makes the model more complicated and difficult to comprehend. On the other hand, all the information is consolidated in one model not requiring a designer to handle multiple models. TERESA (Transformation Environment for inteRactivE Systems representAtions) [43] is the tool support for this transformation based user interface design. TERESA also defines XML notations to specify the interface at each of these stages [45]. Mori et al. [44] provide a more comprehensive description summarizing their research findings including the design goals of TERESA.

Marucci et al. [46] also talk about context sensitive applications and run time adaptation of applications when migrating from one platform to another and how TERESA can be modified to support such interactions with the use of user models. A close connection to the plasticity framework can be identified here but their framework is developed independently of the plasticity framework. This approach involves adding yet another detail to the original task model of the application. Their work realizes that the changing user preferences and knowledge have an effect on the UI. The user model is dynamically adapted at run time and can dynamically change the final interface. There is no explicit mention of how TERESA connects the frontend of an application to its

underlying data and the backend. The "dialogue part" seems to be specified only in terms of changes in presentation but not any change in the backend.

XIML

The goal of XIML (eXtensible Interface Markup Language) [36] is to provide a language for effective representation and manipulation of interaction data which is similar to that of other languages. A XIML representation is a collection of different models discussed in section 2.3 which are called components and each component is a collection of elements. XIML allows attributes and relationships to be associated with the elements. The unique aspect is that it defines an abstract model and abstract properties from which any model can be derived. This is a very flexible language that allows virtually any element to be modeled and correspondingly any attribute or relationship to be associated with it. XIML is primarily designed as a universal language for user interface design that cleanly separates various models of an interface design. The goal is to provide an entire lifecycle support for user interface designing, i.e. design, operation and evaluation. Puerta and Eisenstien [36] suggest interesting uses of XIML including multi-platform development, intelligent interface management and task modeling.

USIXML

USIXML (USer Interface eXtensible Markup Language) [28] is similar in most aspects to other XML based languages in its goals. USIXML basically contains task models represented using an extended version of CTT, domain model analogous to class description in UML, an abstract and concrete user interface description that includes presentation and dialog models. However, USIXML is specifically designed for multiple contexts of use. It has an elaborate context model. USIXML is also a part of the Cameleon project and is based on the Cameleon Reference Framework [24]. USIXML was designed with the goal of creating a user interface description language that is a solution to all problems. Using USIXML one can theoretically design multi-context, multi-modal and multi-platform interfaces. USIXML defines a transformation model and mapping models, based on the theory of graph transformations, for expressing abstract concepts in the form of a graph structure and defining operations producing relevant

transformations on the graph structure [29]. The graph transformations are used to formally describe dialog states and dialog transition. USIXML also provides mechanism to connect the interface to backend application logic. USIXML is a recent contribution and the authors have not provided any concrete evidence in the published literature of the feasibility of this language for developing multi-context [28], multi-modal [30] and multi-platform. There is definitely a lack of convincing examples. However, it already comes with a wide range of tool support like ReversiXML for reverse engineering, model editors like VisiXML, GrafiXML, SketchiXML and KnowiXML and transformation tool called TransformiXML [29].

UIML

UIML (User Interface Markup Language) [31] is an XML-based language that defines a canonical representation for a user interface that can map to existing user interface languages. UIML is modeled by a meta-interface model [38] which separates out interface description specified in <*interface*> section, underlying application logic in <*logic*> section and specification of actual rendering to a particular device in <*presentation*> section for any application. <*interface*> section is composed of four main subsections. <*structure*> section refers to what interface elements the UI is comprised of. <*style*> section refers to presentation style e.g. fonts, colors etc., specified as a set of <*property*> tags on each part. <*content*> refers to text or images that go on a UI. Finally, the <*behavior*> section specifies as a set of conditions and actions performed when these conditions are met. The <*behavior*> section specifies the interactive behavior of any interface. Figure 5 shows a skeleton UIML document and its main components.

```
<?xml version="1.0" encoding="ISO-8859-1"
?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML
3.0 Draft//EN" "UIML3_0g.dtd">
<uiml>
  <head>...</head>
  <interface>
      <structure>...</structure>
      <style>...</style>
      <behavior>...</behavior>
      <content>...</content>
  </interface>
  <peers>...</peers>
  <template>...</template>
</uiml>
```

Figure 5: Skeleton UIML document

The bridge between a canonical representation in UIML and a concrete implementation is a toolkit vocabulary. "A vocabulary specifies a set of *classes* of parts, and *properties* of the classes [38]". The UIML section, <*peers*>, contains another element, <*presentation*>, which defines the vocabulary a UIML document uses. This is a specification of the legal classes for parts in the <*structure*> section that are a part of any target UI toolkit. As the toolkit vocabulary is tied to one particular programming platform, it is considered to be platform-specific vocabulary.

Ali et al. [32] identified the power of this canonical representation to address the multiple platform design and development. They created a generic vocabulary that was independent of any particular platform such that a developer may just specify the description of the user interface once and it could be rendered for different platforms. An interesting observation made by them is that creating one generic vocabulary for all the different kinds of platforms available is not feasible. It is more appropriate to divide these platforms into a set where they share certain common properties, i.e. layout features, and create a generic vocabulary that specifically handles each of these sets. They call it generic vocabulary for a particular family of devices. A family of devices is referred to a set of devices that have similar layout characteristics. A desktop family, for instance, would have similar layout characteristics but the underlying software toolkit and the operating systems may differ. A desktop family of interfaces may consist of a Java based Interface or HTML based interface on either a Windows, Mac or Linux operating system.

A user interface for an application that needs to run all these platforms is specified in generic UIML which provides an abstraction over the underlying software toolkit and platform. There can a cell phone family or PDA family of devices. It is important to emphasize on the definition of family of devices because changes in the layout features between platform significantly changes the interface, to best support the device it runs on. For example, it would be difficult to define a common abstract representation for a tiny screen phone and a desktop. Plomp et al. [35] have demonstrated the use of generic vocabulary to create graphical and speech interfaces for a common application.

UIML was not based on the concepts of model-based interface design originally. However, a UIML document has parts that easily map to some of the existing models. For example, a presentation model is defined in the <*structure*> and <*style*> sections and a dialog model is specified in the <*behavior*> section. Ali et al. [34] introduced task modeling as the first step while designing MPUIs with UIML and ensuring usability. They have devised a transformation method for the same as depicted in figure 6. They also have developed a research prototype that provides an integrated development environment TIDE (Transformation-based Integrated Development Environment) for UIML. The UI designer could specify the UI in UIML and the tool will convert it into a specific language by trail and error. LiquidUI^{TM} is the commercially available authoring environment that is based on UIML and provides rendering features for UIML into different languages.

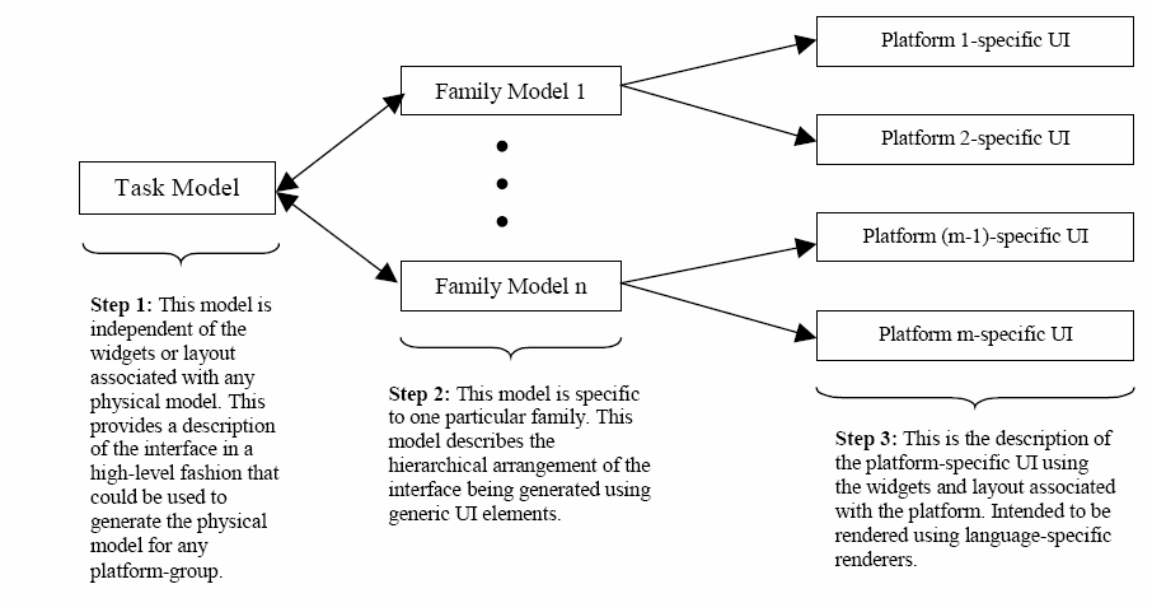


Figure 6: Overall framework for developing MPUIs with UIML and task model

A Comparison

It is probably hard to say which approach is better as there is a lack of usability studies on these approaches. Few examples have been shown to demonstrate how these different approaches are capable of producing MPUIs but there is no mention of how comfortable are developers using them. Souchon et al. [50] conducted an extensive review of the XML-compliant languages that have been developed for building user interfaces. They chose important criteria for evaluation of these languages like component models, methodology, tool support, target, level of abstraction among others. Their work provides a big picture of this latest trend in model-based methodology and covers many different XML languages for specifying UIs. Another work by Trewin et al. [48] focused on assessing how these languages support universal usability. But it is probably even more important to evaluate how UI engineers would feel about using the approaches mentioned above. There is no significant work published in this area except that by Paternò [5]. An evaluation was conducted at Motorola Italy Software Development Center with an early version of TERESA. Their study showed that it was easy to prototype using TERESA and the developer were able to do it in a short span of time but the time for making further changes to the generated prototype increased

tremendously. The time to redesign the interface almost doubled. This suggests that there is a huge scope for improvement.

2.4.5 Reverse Engineering Web Applications

A significant target of the model-based design community has been web applications. Lot of tools have been built for reverse engineering web applications with different goals. The goals differ from supporting general maintenance of websites to supporting efficient evolution or adaptation to another platform. These approaches had significant drawbacks the led us to derive a new approach for reverse engineering web applications built in Click.

WebRevenge

WebRevenge (Web Reverse Engineering) is a tool developed by Paganelli et al. [56] that reverse engineers a website into a task model through what they call a bottom-up approach to extract an abstract design from a concrete implementation. The approach followed by the tool is to utilize HTML DOM to analyze page structure and navigations that are internal or external to the website. For each building block of a website i.e. links, input fields, buttons etc., a rule is defined to build a task model specific to these building blocks [55]. Higher level task models can be generated by combining the smaller task models. The task models are represented using the CTT notation. The resulting task model can serve as an interface design for the website. There is no doubt that the automatic reconstruction of a task model for a web page based on the rules above is very useful. However, this automatic algorithm lacks the ability to capture the dynamic behavior of the website. This is a limitation inherent to dynamic websites since the behavior related information is captured in a web scripting language and pages are generated on the fly. This makes parsing difficult and as a result the reverse engineering is limited to the structural aspect of a website.

VAQUITA

VAQUITA (reVerse engineering of Applications through QUestions, Information selection, and Transformation Alternatives) [52] is a tool for reverse engineering

presentation and dialog model from a website. The presentation and dialog models recovered by VAQUITA are expressed in XIML. They introduce a notion of flexible reverse engineering which is essentially about guiding the above mentioned process based on certain heuristics. These heuristics are defined in close correspondence to the target platform and can be controlled by a developer through VAQUITA [53]. VAQUITA, like WebRevenge, also relies on only static HTML parsing and the dialog (or behavior) aspect of the website is limited to the page-and-link behavior (navigations) [54].

WARE

A parallel field of research is automatic reverse engineering of UML models from already existing web applications. The motivation behind this research is to manage the increasing complexity of the application and streamlining web application built in an ad hoc manner to support further development and testing. The WARE (Web Application Reverse Engineering) [57] approach involves both static and dynamic analysis. Static analysis is based on source code parsing whereas dynamic analysis is performed by executing the application, recording its behavior and tracing back to the source code. The WARE approach recovers class diagrams, use cases and sequence interaction diagrams by analysis of code, its execution and interaction among various components of the web application.

ReWeb

The main goal of ReWeb [58] is to improve the maintainability, usability and portability of web applications by restructuring and rewriting web application. This work is essentially focused around tidying up HTML but also goes a little farther by providing missing keyboard shortcuts, automatic organization into frames, enforcing stylesheets and eliminating deprecated tags. ReWeb uses Design Maintenance Systems Reengineering Toolkit.

2.4.6 Re-Engineering Web Applications

Bouillon et al. [53] elaborate the need for model-based reengineering of websites. Once models are reverse engineered, we can take one of the forward engineering approaches to generate different interfaces. They mention different UI re-engineering approaches based on certain context such as redocumentation, restructuring, redesigning and design recovery. Design Recovery, recovering task models for a website, is of our main concern here. It is suggested that recovering task models from HTML is impractical and does not scale well and may also require additional information. This research will focus on this aspect as the internal representation of Click provides more information than that can be recovered from plain HTML.

2.5 Click, a Tool for End-user Web Development

Click is a web-based integrated environment that contains a visual development tool, code-editing features, a preview mode, and a database management interface [63]. No installation or configuration is required on part of the end-user developer. A developer starts creating an application with a blank page or by selecting a predefined application template (such as service request form, online registration, or staff database). Next, components can be placed on the screen and suitable databases may be defined. Once the developer places components on the screen, Click generates layout and behavioral code that can be manually customized by advanced developers.

Click provides a set of pre-defined high level components that form the building blocks of an interactive web application. The high level components are divided into four major classes: Layout, Input, Output and Authentication. Layout components are simple static components like text and images. Input components are the basic form input elements like textbox, checkbox, radio button list, dropdown list and button. Output component, a data table, is a complex component that does not have an equivalent in HTML. An authentication component is a login box that contains a user name and password input field for which Click automatically implements the authentication functionality. The design rationale and selection of features is based on a study of end-users by Rode et al. [61]. Figure 7 is a screen shot of Click which shows the integrated

development environment, a data table component placed on a web page and a button component being defined through a dialog box.

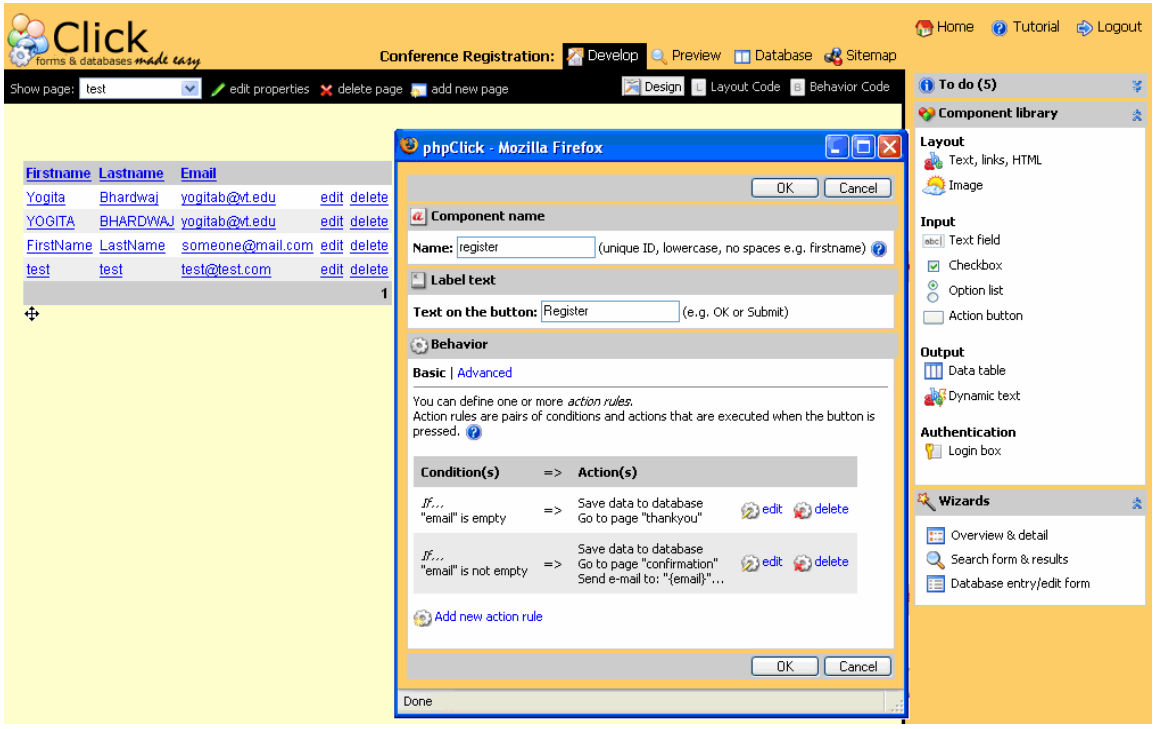


Figure 7: Click's development environment.

2.5.1 Architecture of the end-user developed web application

An application developed by an end-user using Click is based upon an extensible component framework called PRADO [64]. Each page in the application is composed of a layout specification in the form of a template file (page.tpl) and a behavior specification implemented in PHP (page.php). Click implements an event-based programming model similar to that found in ASP.NET or Java Server Faces (or programming tools for desktop applications such as Visual Basic). The behavior code for buttons and data tables is comprised of event handlers that use high-level PHP functions predefined by Click and PRADO (e.g. `sendEmail`, `goToPage`, etc.). These functions are explained later in context of behavior specifications for the components. Figure 8 shows an excerpt from the template file that contains text, a textbox, a checkbox, a radio button list component.

Each component belongs to a certain class and has few properties. Click also saves the layout and behavior specification in an internal XML files (explained later).

```
<com:Form>
<%include Pages.showOnEveryPage %>
 <com:HtmlText ID="htmltext1" X="74" Y="27" Z="52">
  <prop:Text>Welcome! Please register below.</prop:Text>
 </com:HtmlText>
 <com:RadioButtonList ID="mrms" RepeatDirection="Horizontal"
  RepeatColumns="2"" DbFieldName="data:mrms" X="73" Y="115" Z="71">
  <com:TListItem Value="Mr." >
   <prop:Text> Mr. </prop:Text>
  </com:TListItem>
  <com:TListItem Value="Ms." >
   <prop:Text> Ms. </prop:Text>
  </com:TListItem>
 </com:RadioButtonList>
 <com:InputText ID="email" X="77" Y="257" Z="82" Columns="20"
  Rows="1" TextMode="SingleLine" DbFieldName="data:email"
  InputRequired="true">
  <prop:Label> E-Mail: </prop:Label>
  <prop:ErrorMessage>Please enter a valid e-mail
  address.</prop:ErrorMessage>
  <prop:RegularExpression>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-
  .]\w+)*</prop:RegularExpression>
 </com:InputText>
 <com:Checkbox ID="sendemail" Checked="false" TextAlign="Right"
  DbFieldName="" X="73" Y="458" Z="78">
  <prop:Text>Send me an email after the registration</prop:Text>
 </com:Checkbox>
</com:Form>
```

Figure 8: A sample template file generated by Click

2.5.2 Presentation Attributes of Components

Each predefined component in Click is associated with a class that defines how this component is rendered in HTML. Table 3 shows a mapping between classes used by a Click application and an equivalent HTML widget.

Table 3: Mapping between Click/PRADO component and their equivalent HTML tags

Click/PRADO component	HTML
<com:HtmlText>	
<com:Image>	<img>
<com:InputText TextMode="SingleLine">	<input type = "text">
<com:InputText TextMode="MultiLine">	<textarea>

<com:Checkbox>	<input type = “checkbox”>
<com:Button>	<input type = “submit”>
<com:RadioButtonList>	<input type= “radio”>
<com:DropDownList>	<select><option>
<com:LoginBox>	<input type=“text”> + <input type=“password”> + <input type=“submit”>

Location

A developer places components in Click using drag-and-drop and absolute positioning. The location of the component is thus an important presentation aspect captured in Click’s abstract representation as X, Y and Z attributes. Since Click’s visual development environment places components in different layers for easy manipulation, Z value refers to the z-order of the component.

Style

Style for a Click application is mainly determined by a default CSS file. A developer can specify, through Click’s interface, default background color, font, font-size and font-color for the application.

2.5.3 Behavior of Components

The run-time behavior of components can be specified through a dialog box in Click and then Click generates the code to implement this behavior and stores the specifications in an XML file as well. There is not a complete separation between a component’s layout (presentation) and behavior. Input text validation can be considered to be the behavior of the application but it is expressed in the layout code in Click. The following are some important behavior specifications that a developer can specify.

Input text validation

Input fields on a form typically need to be validated before they are submitted. Input fields can be specified to accept any input without any restrictions or the user is required to fill some value in it. Click uses regular expression “*validators”* to check for valid email address or a valid date format using regular expression matching. Range validations checks fall into two categories. Input values can be string such that the number of characters is within certain range or it can be an integer, a number within

certain range. The developer is required to specify an error message the application will pop-up when the user input does not match the specifications. Figure 9 shows sample template code for InputText component that accepts string input between 1 and 30 characters but can accept empty input, specified through "InputRequired" attribute. It pops up the given message (<prop:ErrorMessage/>) in case the value entered is wrong.

```
<com:InputText ID="firstname" X="77" Y="156" Z="80" Columns="20"
Rows="1" TextMode="SingleLine" DbFieldName="data:firstname"
InputRequired="false" ValueType="Characters" MinValue="1"
MaxValue="30">
   <prop:Label>First name:</prop:Label>
   <prop:ErrorMessage>Please enter between 1 and 30
   characters.</prop:ErrorMessage>
</com:InputText>
```

Figure 9: A sample input text component with input constraints in Click

Database Connections

Input fields like textbox, checkbox, radio button or dropdown list can be connected to a field in some database table. This information is also captured in the abstract representation as a property "DbFieldName" whose value is of the form "table_name:field_name". While saving a value a user enters for any of these input fields the "DbFieldName" is parsed to create an appropriate SQL query.

Button Click Behavior

Button click behavior is specified through a rule-based language. Each rule contains a set of actions and may contain a condition. If a condition is specified, the actions associated with the rule are performed only when the condition is true. A condition is a Boolean expression on one of the input fields. For instance, a condition holds if a checkbox is checked. Click has a set of predefined actions that a developer may choose from. These include: save values from input fields to database, send an email, reset values entered by the user, reload the current page and go to another page of the application. The behavior rules for each button are stored in an XML file and the behavior code, in PHP, is automatically generated that can be customized by the developer. Figures 10, 11 and 12 describe three different representations of a button

component, with figure 10 describing the layout and figure 11 and figure 12 describing the behavior.

```
<com:Button ID="registerbutton" Text="Register"
OnClick="registerbutton_runActions" X="76" Y="498" Z="83"/>
```

Figure 10: Sample template code for a Button component in Click

```
<ActionRules Id="registerbutton">
 <actionRule>
   <conditions connector="and">
    <condition fieldId="sendemail" operator="checked" parameter=""
/>
   </conditions>
   <actions>
    <actionGoToPage pageId="emailthankyou" />
   </actions>
 </actionRule>
</ActionRules>
```

Figure 11: Click's XML representation of the actions specified by a developer on the button

```
function registerbutton_runActions($button, $parameter) {
    $condition1 = $this->newCondition('{sendemail}','checked');
    if ($condition1->isTrue())
    {
        $this->runAction('goToPage','emailthankyou');
    }
}
```

Figure 12: High-level behavior code for the button in PHP by Click

Data table

A DataTable component displays results of a SQL query on one of the application's database. A developer can use the query builder interface in Click to build a query or can directly specify a SQL query. Similar to the button component, Click stores these specifications in an XML file and generates the behavior code for the same. The data table can also show a link to edit or delete rows displayed. Clicking on edit link for a row would take the user to a form that displays contents of the selected row which the user can edit and save to the database. In addition, Click also allows each row in the data table to be a link to another page that may show more details about the row selected. Figure 13 shows an example of the data table component with all the links.

Firstname	Lastname	Email		
Yogita	Bhardwaj	yogitab@vt.edu	edit	delete
YOGITA	BHARDWAJ	yogitab@vt.edu	edit	delete
FirstName	LastName	someone@mail.com	edit	delete
test	test	test@test.com	edit	delete
				1

Figure 13: Sample data table component created using Click

Figure 14 shows a trimmed down version (not showing custom fonts and styles) of a data table component layout code in Click. The overall component is of class "<com:DataTable>" and the properties defined on this class apply to the entire data table. Each column in the data table is composed of a class "<com:HyperLinkColumn>" if the data rows are linked to a page that may show the details of the data row whose URL is specified as a property on this column. Otherwise the columns are of type "<com:TBoundColumn>" that just shows the text. Each column is connected to a field in a data table and it displays data contained in that field. The edit and delete columns are categorized as action columns and are specified as class "<com:ButtonColumn>" that can be either rendered as a link or as a button. The behavior code for this data table component uses PRADO database classes to execute the SQL query for performing the record fetch, edit or delete operations.

```
<com:DataTable ID="datatable1" X="18" Y="49" Z="54"
  OnSortCommand="datatable1_sort"
  OnPageCommand="datatable1_page"
  OnDeleteCommand="datatable1_deleteRecord">
  <com:HyperLinkColumn DataTextField="firstname"
  DataTextFormatString="" DataNavigateUrlField="id"
  DataNavigateUrlFormatString="index.php?page
  =page_showdata&dbTable=data&id=%d" HeaderText="Firstname"
  SortExpression="firstname"/>
  <com:HyperLinkColumn DataTextField="lastname"
  DataNavigateUrlFormatString="index.php?page
  =page_showdata&dbTable=data&id=%d" HeaderText="Lastname"
  SortExpression="lastname"/>
  <com:ButtonColumn Text="delete" CommandName="delete"
  ButtonType="LinkButton" />
</com:DataTable>
```

Figure 14: Template code for a DataTable component in Click

Authentication

Click allows a developer to create secure applications through an internal or external authentication system. External authentication is the Virginia Tech's authentication system, using LDAP, where users from Virginia Tech could use their personal identification and password for accessing the applications. Internal authentication happens against the database table, "users".

Appendix A.1 shows a sample web page created in Click and appendices A.2 and A.3 show the layout and behavior code generated by Click.

2.6 Summary

The goal of our research is to facilitate end-user and expert developer collaboration and development of MPUIs for end-user developed web applications. The survey of the literature presented in this chapter throws light on certain models and representations that will help in achieving these goals. Sitemap is an important artifact we need to automatically generate from an end-user developer application. The sitemap will be useful to both end-users and expert developers in understanding and brainstorming the web application. Task modeling is considered to be an essential step in interaction design and ConcurTaskTrees notation is used widely for representing task models. The original goal of UIML was to make UI development, specifically prototyping, easy and more accessible to UI designers who may not be expert UI programmer by providing a canonical representation. UIML by nature is a simple language with a goal of providing only a canonical representation for UIs that can be mapped to any UI toolkit. Other tools propose a general solution which broad enough to discourage designers to be able to customize these solutions for their purposes. These representations are good candidates for the purpose of this research.

Chapter 3: Reverse Engineering End-user Web Applications

The reverse engineering toolset, main contribution of this research, automatically generates abstracts representations from end-user developed web applications. There are four tools in this toolset that generate a sitemap, text documentation, a task model and the UIML representation. The algorithm for each tool is similar and begins with identifying pages and databases in the application. Then the algorithm parses each page to identify its high level components, their properties and behavior and create an appropriate representation of each component and its behavior as it is parsed. Sitemap and text documentation allows for smooth transitioning of the web application from an end-user to an expert developer. The task model and the UIML representation support development of MPUIs for web applications created by an end-user.

Section 3.1 describes in detail what a sitemap is comprised of and how it is automatically generated using Click's representation. Section 3.2 is a design description of the documentation. Generation of the task model, using the CTT notation, from a Click application is described in section 3.3. This section describes a task structure for each of the building blocks of a Click application. These task structures are connected to create the final task model of the whole application. Finally, a description of how Click's representation is mapped to an equivalent representation in generic UIML is presented in section 3.4. Appendix A shows the source code for an example application built in Click and the generated artifacts for the same. The reverse engineering algorithm has been integrated with Click and can be accessed through a link in Click's interface.

The reverse engineering toolset identifies these presentation and behavior aspects and generates the artifacts that capture these details. This process is discussed next.

3.1 Sitemap

A sitemap is a diagrammatic representation of the high level structure of a website. The reverse engineering toolset identifies the high level structure of a web application in terms of pages and navigations among these pages and depicts it in the

sitemap. In addition to this, an important aspect of web based applications is their underlying databases. Web pages may contain information that is derived from a database or they may save some information to a database. For secure applications, their authentication system plays an important role too. This structural information is also depicted in the sitemap.

Sitemap blocks

The sitemap shows a box for each of the pages in the application. The box is labeled with the page file name. A database forms another block which is represented by a figure that resembles a table and is labeled by the table name. Default tables, "data" and "users", will always be present in a sitemap. Virginia Tech's authentication system is represented by a circle labeled "VT Auth". The home page of the application is distinguished from other pages by having a heavy border for the node.

Navigations and Links

Depicting navigations is a crucial aspect of the sitemap. The toolset separates navigations into categories and thus assigns different styles of edges to represent these navigations. "Go to Page" actions that happen on a button click are represented using solid green colored edges. If there is a condition, which the developer has specified on this particular action, it is displayed as the label of the edge. The default label is 'Button'. If data is saved from a page to one or more databases on a button click, a dashed green edge from the page to all the databases is shown. The label for this navigation specifies on which button click the data is saved and a part of input fields saved. E.g. '"Register" saves firstname, lastname …'. On the other hand actions happening on clicking a hyperlink are depicted using a solid blue-colored edge. Authentication actions are represented using double-sided solid red-colored edges. In general, interactions with external entities like databases and authentication systems are shown with dashed gray lines. The graph is a directed graph and the direction of the edge represents the direction in which information travels.

The sitemap shows an edge from any page to itself, labeled "validations failed on 'button'". This means that if the page contains any input fields that need to be validated

once a button is clicked and the validation fails, the user is taken back to the same page. Data from any database may be displayed on a page using the DataTable component and this interaction is shown using a dashed gray edge from the database to the page. View details or edit link from a page containing data records to a page showing the record details to be viewed or edited are depicted using solid blue edges with labels as "View Details" and "Edit Record" respectively. A delete link brings the user back to the same page and the DataTable component is updated with the new set of records. This is a self link to the page depicted using solid blue edge with label as "Delete Record". A link between a secured page and the login page is a solid red colored edge labeled as "login required". This is a double sided edge that shows that the secured page may redirect to the login page if the user is not logged on and once the user is authenticated the login page redirects back to the secured page.

Figure 15 shows a sample sitemap automatically created from a web application built in Click. The sitemap shows how a database table has remained unused in the application and a set of pages are inaccessible given that a user always starts at the home page. Figure 16 shows the legend for the same. We are exploring more ways to code information other than simply using color to account for people's sensitivity towards colors.

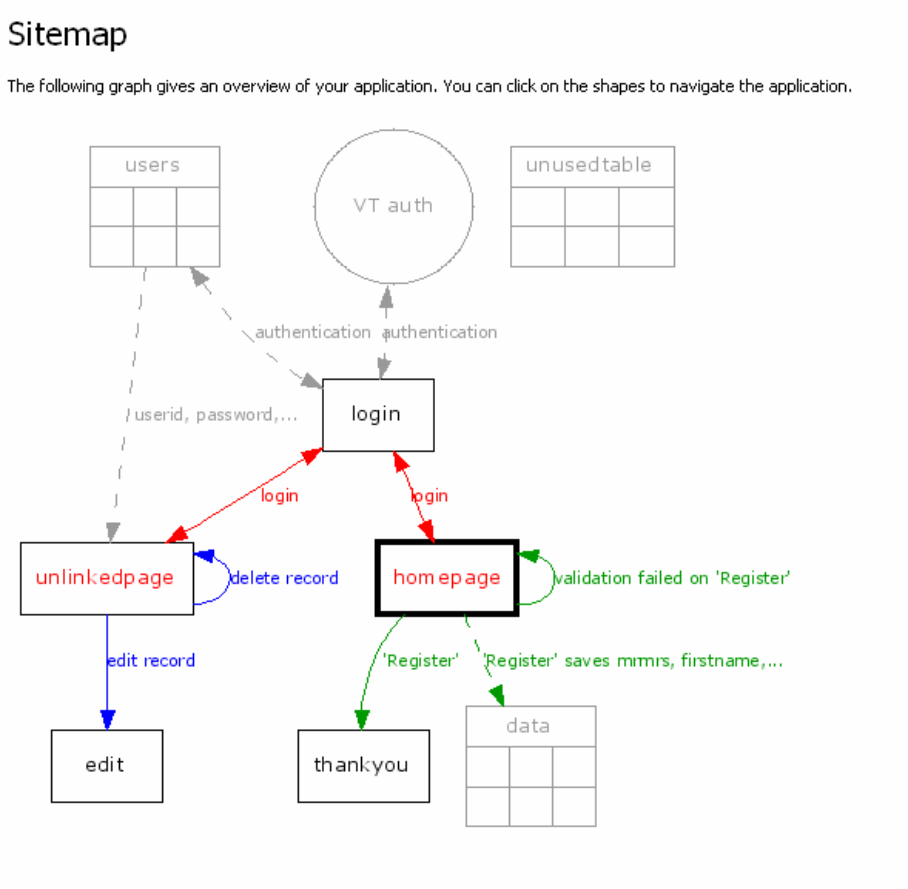


Figure 15: A sample sitemap from an application created in Click.

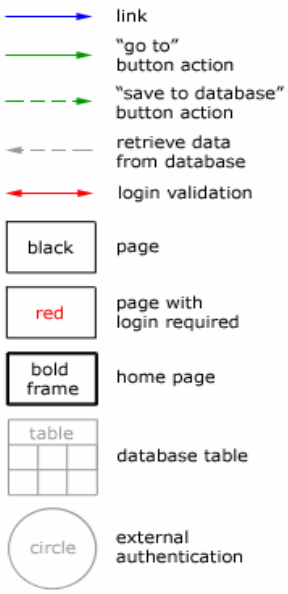


Figure 16: Legend for the sitemap

Hyperlinked Nodes

Each block in the sitemap, except "VT Auth", is a link to their corresponding entity in Click's environment. Every block representing a page is a link to the actual page in application shown in the design view of Click. The database block is also linked to the corresponding table shown in the database editing interface in Click. The application designers can use this feature to explore the design of application while keeping the big picture in perspective.

Implementation

The sitemap is an HTML page containing an image map generated using the GraphViz, graph visualization software. Input to the GraphViz dot executable is a specification of a directed graph using the DOT language [65]. The DOT language has many sophisticated features for creating diagrams, such as a variety of colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes. The sitemap makes heavy use of these features to create an elaborate illustration. The implementation uses an open source GraphViz library in PHP that provides an API to create nodes and edges and to generate image maps for the web.

Sitemap is an important artifact that provides an overview of the web application created by an end-user. This would help an expert developer understand the main components in an application and its behavior without having to read the low level source code. It serves the purpose of smoothly transitioning the end-user developed web application to an expert developer for further enhancements. Since the sitemap feature does not involve any programming technology related details, it may be useful to even non-programmers to get an overview of their application. Realizing this potential, Sitemap is presented as a tab in Click's development environment for the end-user developers. Clicking on the tab regenerates the sitemap which ensures that the sitemap is always up-to-date with the ever-evolving design. As sitemap is automatically generated from the application, at any point it may depict wrong or missing connections between pages, unreachable pages or unused databases. This would be an important piece of information for the developers to validate their design

3.2 Documentation

The second artifact produced by the reverse engineering toolset is a text documentation. Complementing the high level overview provided by a sitemap, the documentation for a web application provides finer details of functionality of the application. Documentation also serves as an aid, along with sitemap, for understanding the application or as a tool for discussion among the developers. The reverse engineering toolset generates a description in an HTML file for each page. The documentation describes the purpose of each page in the application, whether it is a form, displays database content or is simply static information page.

The toolset recognizes whether the application contains a form that a user needs to fill and what kind of values each input field is expecting. The text generated by the tool describes each component in detail. The most significant aspect of the documentation is the use of hyperlinks to connect the relevant pieces of information. Click's design allows a page to use values from the input fields from all the pages in the application not just those defined in the page itself. While expressing this feature in the documentation, these input fields appear as hyperlinks to the documentation of their respective pages where the input fields are actually defined. To situate the context even more, the URL is appended with the target of a particular input field that takes the user straight to the description of that field. Each page that has a link to another page has an equivalent link in its documentation to the documentation of the target page.

Databases are an important component of a web application developed in Click. The generated documentation also contains database schema for the databases of the application. The database schemas are generated in separate html pages but are hyperlinked from the documentation of other pages whenever the databases are referenced. For example, while describing database connections of input fields or DataTable components accessing data from a database, the reference to these databases is linked to their documentation. Authentication information is specified in the documentation of each page describing whether the page requires authentication and if so what kind of authentication is performed. It specifies which page in the application is the login page and puts a link to that page in the documentation. Figure 17 shows an excerpt

from the documentation of a page created in Click and complete documentation is shown in appendix A.5.

Evaluation Application- Page: homepage

This is the home page of the application, Evaluation Application
This page requires the user to fill up a form. Following are the form fields:

firstname: Text Input Field
Database Connection:
Table: data Field: firstname
Constraints: This is an optional field.
firstname should be a string having between 1 and 30 characters.
Filling this field incorrectly will generate the error message "Please enter between 1 and 30 characters."
…

On clicking **Register** the form is validated and if any error is found the user is prompted to correct the input. Once the form is validated following actions happen:
=> Always
Save data from current page to database. Data from the following input fields are saved to table data: mrms, firstname, lastname, email

=> If "sendemail" is checked
Go to page "thankyou". Find the documentation here

=> If "sendemail" is not checked
Go to page "thankyounoemail". Find the documentation here

Figure 17: An excerpt from the documentation of web page created in Click

3.3 Task Model

The tool created by Paganelli et al., WebRevenge [56], proposed an interesting possibility of automatically reverse engineering task models from a website based on a set of rules. Task models are represented using the CTT notation. The rules define a set of building blocks of the higher level task model of a web page based on analysis of all the important interaction elements and the components that group them. The scope of these rules is limited to analyzing simple static web pages. As Click provides an abstraction over the implementation of dynamic behavior through the template file and the behavior XML files, it is easy to parse and automatically reflect this dynamic behavior in the task model. The resulting task model, generated by the reverse engineering toolset, will be more sophisticated than that generated by WebRevenge and will more closely reflect the application behavior. The sections 3.3.1 to 3.3.5 describe a redesign of the rules as they

apply to Click and explain the enhancements to these rules for incorporating dynamic behavior. The rules for creating the initial task structure of a web page and for links internal and external to the website remain unchanged. The names associated with the tasks are carefully chosen to represent the task as closely as possible. However, the figures for the task structures in the following sections use generic names that are replaced by values specific to each application. Appendix A.6 shows how these rules are applied to a web page created in Click and how task structures for various components are connected to create a task model for the whole page.

3.3.1 Task structure for a form

A Click (or PRADO) application requires a form tag (<com:Form>) to always be present and therefore it is not a good measure of whether the web page contains a form or not. The reverse engineering toolset recognizes whether the web page contains any form input fields, like text fields, radio buttons, push buttons etc. If so, an abstract task "Compile form" is created with two abstraction sub-tasks "Fill in Form Fields" and "Select Submit Button". The subtask of "Select Submit Button" *disables* the sub-task "Fill in Form Fields". Variations are possible for this task structure such as a web page with no input fields, but having buttons, will not have subtask "Fill in Form Fields". There can be more than one buttons on the form. This scenario is explained in the context of task structure for a button in section 3.3.3. "Fill in Form Fields" is an iterative task because the user can attempt to fill the form fields and can make any number of corrections before submitting the form. Once the form is submitted the application loads another page which could be the same page depending on what happens in the backend after form submission. This is even true for the case where the only action defined on the button is to reset the form fields. This is a deviation from the form structure created by WebRevenge where the "Compile Form" task is an iterative task when a reset button is specified on the form. In that case the user may reset the form any number of times and start from scratch. Although the two scenarios defined above are exactly the same, the task model reflects Click's implementation (i.e. actual system task model). Figure 18 shows the corresponding task structure for the form.

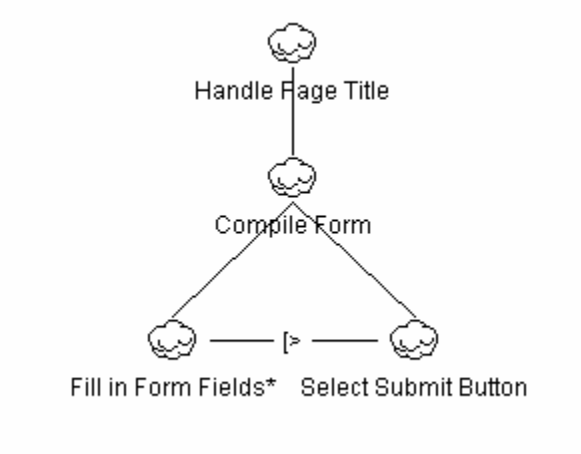


Figure 18: Task structure for a form

3.3.2 Task structure for input fields

The following section defines how task structures are created for each of the input fields a developer can create on a form. All the tasks of filling in various input fields on a form are not dependent on each other, at least as defined by Click. One can fill in any input field without having filled any other. Thus, all these task structures are temporally related to each other with "Order Independency" operator in the final task structure of the web page.

Input Text

While creating subtasks to insert into each of the form fields, we can specify if the form fields are mandatory or not and make them as optional tasks in case they can be left empty. Click specifies a property "InputRequired" on each InputText component. If this property is set to false, we make the overall task of inserting a value into this field as optional. The same concept applies for multi-line input text fields. Figure 19 shows the task structure for an optional InputText component. A mandatory input text field will have a similar task structure except that the task "Type Text Name" would not be optional.

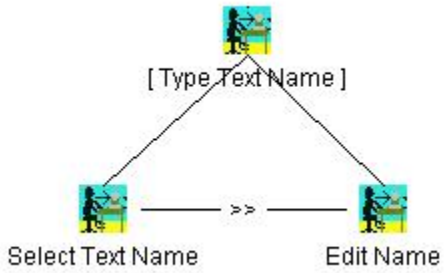


Figure 19: Task structure for an optional InputText component

Checkbox

Selecting a checkbox will be a single interactive task called "Select Checkbox checkboxid". Since a checkbox already has a default value, it is optional to select the checkbox. Paganelli et al. [55] suggest it to be an iterative task where there are multiple alternatives that can be selected, but in Click and general HTML each checkbox is an independent entity not having multiple alternatives. Click does not yet support Checkbox group.

Option List

Selecting a value from a list of options is a single task. The list of options can be rendered either as a dropdown list or a radio button list but it is a presentation aspect and does not affect the task structure. The single task will be an interactive task "Select Option name". It is possible for a developer to specify whether the list of options has a default value or not. If a default value is specified then the task of selecting an option is optional otherwise not.

3.3.3 Task structure for a button

When extending the sub-task, "Select Submit Button", in the form task structure defined in section 3.3.1, two sub-tasks for "Select Submit Button" are created. One is an interaction task, "Click Submit Button", which *enables* another abstraction sub-task, "Submit Form". The "Submit Form" task should not be merely a system task as the dynamic behavior for a button takes effect only when the form is submitted. Figure 20 shows the common task structure for each button defined in Click.

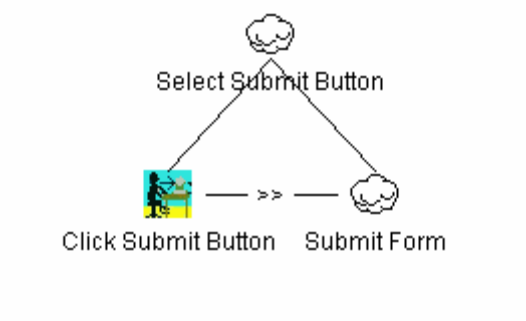


Figure 20: Task structure for a Submit Button

The application behavior, when a button is clicked, may involve form input validation which may result in correction of the form by the user or it may perform other actions defined by the developer if all the input is correct. Input validation upon submitting the form will result in a task structure as shown in figure 21. The "Submit Form" task is an abstract task which consists of an application subtask "Submit" that does the task of submitting the form data to the server side of the application. Submitting the form *enables* an abstract task called "Validation". "Validation" task is composed of an application task "Validate Input Fields" that validates the input fields and *enables* the abstract task of "Correct Form Fields" if the form fields are wrong and passes this information to the task. If validation was unsuccessful the user is requested to correct the form fields in error by displaying the incorrect fields with an error message. "Correct Form Fields" is composed of an application task, "Display Incorrect Fields", which displays the input fields that were incorrect and this task *enables* an interaction task, "Edit Incorrect Fields", which lets the user correct the fields in error. Validation task is an iterative task indicating that it continues until the user enters correct values. Overall actions performed by the application, after a button is clicked, are sub-tasks under an abstraction task, "Handle Submit", which is enabled by the "Validation" task and receives information from it.

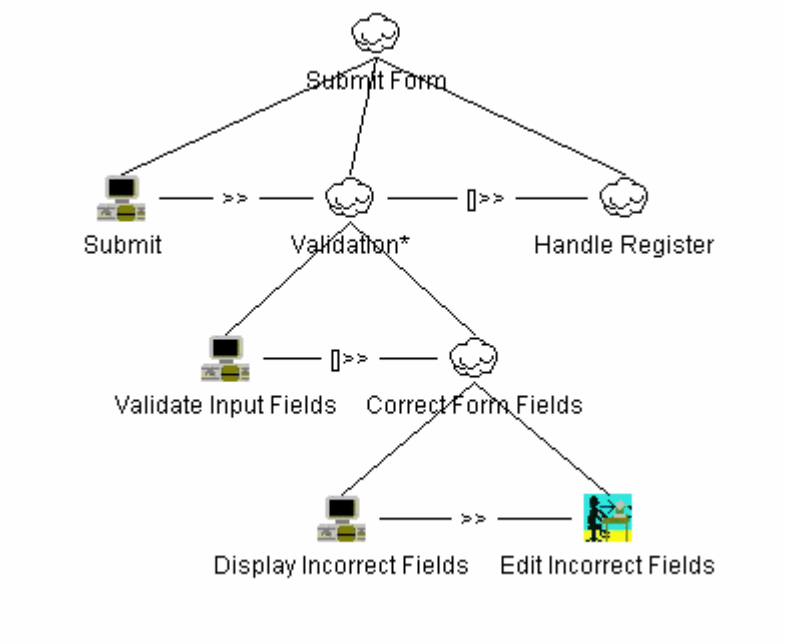


Figure 21: Task model for Input Validation and Input Correction on a Button Click

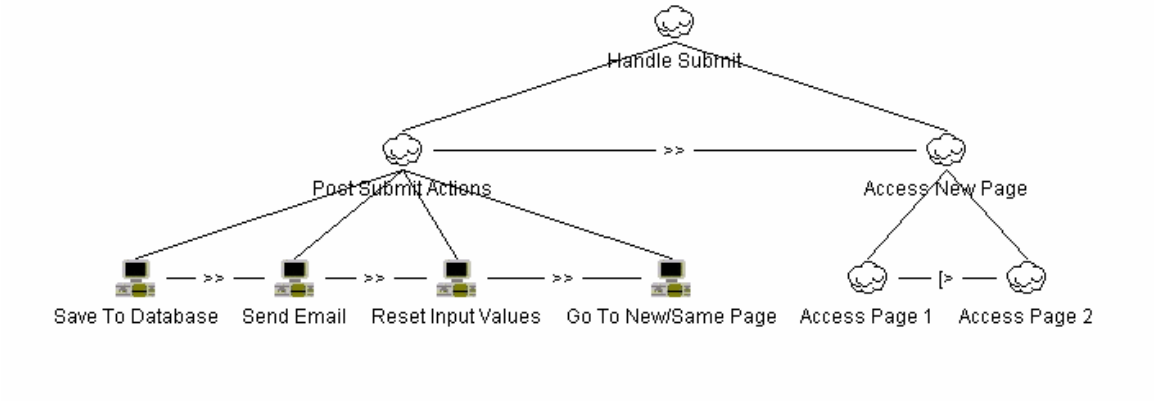


Figure 22: Task Model for Submit Button Actions

Figure 22 is an example of a button's behavior. The "Handle Submit" task is composed of another abstraction task "Post Submit Actions". "Post Submit Actions" has subtasks as application tasks that represent the actions specified by a developer for the button. These tasks are "Send Email" that sends an email as specified by the developer, "Save To Database" that saves information from the current page or the whole application to specified databases, "Reset form fields" which is analogous to a Reset button and finally go to another or same page specified as "Go to New/Same Page" application task. The order in which these application tasks are performed is predefined

by Click. If “Go to New/Same Page” action is defined then the next task in the temporal sequence to “Post Submit Actions” is the abstraction task “Access New Page”. This abstraction task is composed of all the different pages that could be accessed after a button is clicked. Each of those pages is listed as an abstraction task “Access Page Name” and is temporally related with the “disabling” operator as accessing one page disables the task of accessing another page. These abstraction tasks are high level tasks and their corresponding task model is created as the toolset progresses on to other pages in the application.

A developer may be able to put multiple buttons on a page. A task structure for each of the buttons is created as described above and all these task structures are connected with a temporal operator “disabling”. This is because a user may press any one button but it disables the selection of any other button on the page. Each of the tasks in one button’s task structure is differentiated from other button’s task structure by the button ids that are added to the task identifier.

3.3.4 Task structure for a data table

The data table component is essentially made up of a SQL query whose results are rendered on the page in a table layout. The interaction between the application and the data table is through this SQL query. The table is shown as a dynamic component with maximum 5 rows displayed at a time. Rest of the data rows are hidden inside the links displayed on the bottom of the component. When the table has more than 5 rows the user clicks on one of the links to view a different set of data rows. Each column in a data table also has a link to sort the columns. Each row in a DataTable in Click can be a link to the details page and can have links to edit page or for deleting the record.

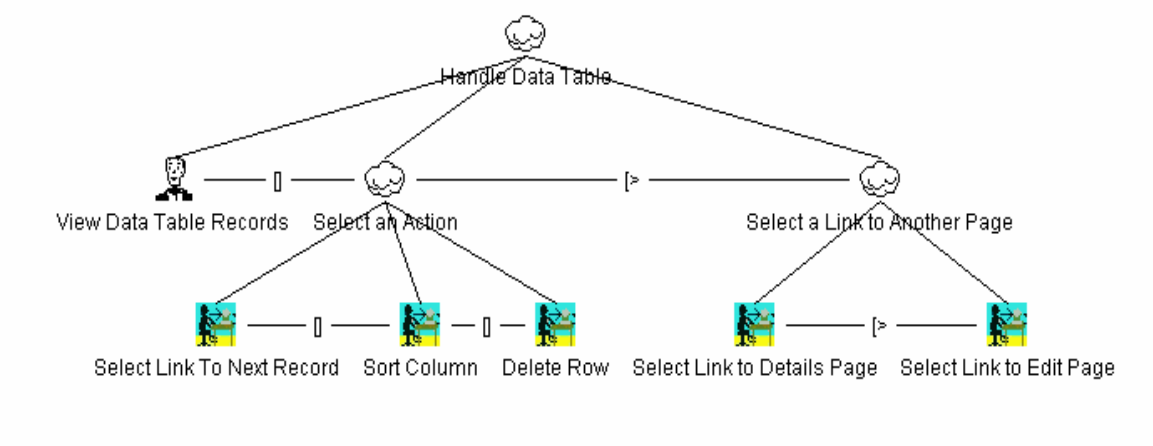


Figure 23: Task model for a data table component

The task structure described in figure 23 represents typical tasks supported by a data table. A user can view the data table records (user task "View Data Table Records") but has a *choice* of either performing the interactive tasks "Select Link to Next Record", "Sort Column" or "Delete Row". All these actions are also temporally related with the "choice" operator as once the user performs one task, the other tasks cannot be performed but they are available once the previous task is over. The other set of actions available are clicking on the links in the data table that takes the user to another page. This set of actions *disables* the tasks of viewing the data records or selecting another link on the data table. The interaction tasks available are "Select Link to Details Page" and "Select Link to Edit Page". Further action when a user performs these tasks is that of accessing the corresponding web page. This task is the high level task of accessing a page i.e. abstraction task "Access Page Name". This task is not shown in figure 23 and its creation proceeds as already discussed. A developer can define multiple tables on a page and each of these task structures are connected via the temporal operator "choice".

3.3.5 Task structure for authentication

Click allows a developer to define security constraints for each page. If a page is secured and the user is not already logged in, then the first task that the user performs is logging in to the application. The application directs the user to the login page and then once the login is complete the user is redirected to the original page. The task structure corresponding to the authentication behavior is shown in figure 24.

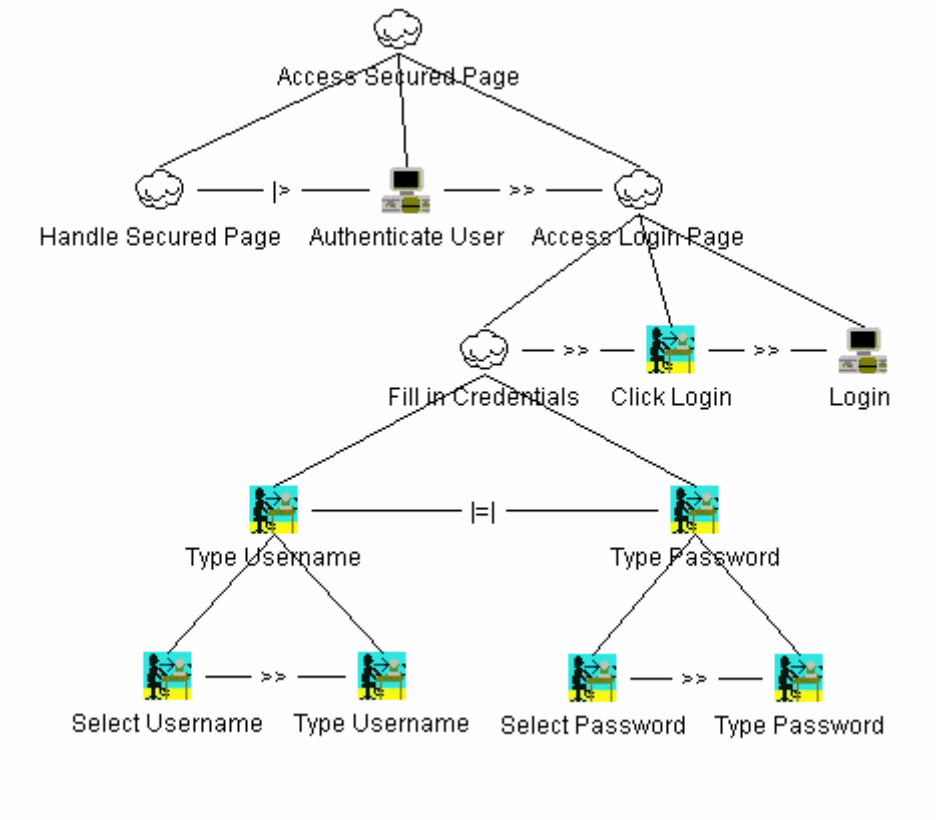


Figure 24: Task structure for accessing a login-protected page

The root task, "Access Secured Page", corresponds to an abstract task of accessing a login protected page. Detailed task of handling the secure page, "Handle Secured Page", is *suspended* until the user is authenticated (application task "Authenticate User"). These tasks are temporally related using the suspend/resume, |>, operator. "Authenticate User" task *enables* the abstraction task, "Access Login Page", as the user is redirected to the login page. On the login page, users are asked to fill in their credentials that involve typing in the user name and password followed by clicking on Login button and then initiating the actual login process. The task structure of the abstract task "Fill in Credentials" is similar to "Fill in Form Fields" task structure as the login page is essentially a form with two fields.

3.3.6 Implementation

A task model for each page is created separately as it may be more convenient in understanding than all the task models for each page combined in one huge task model. However, a combined task model for the entire application is also created such that it can

be used in automatic simulators like the one provided by the CTTE [40]. The reverse engineering toolset generates an XML representation for the task model. The CTTE generates the graphical tree using this XML representation.

3.4 UIML representation

The fourth tool in the reverse engineering toolset for Click generates an alternative representation in UIML for web applications specifying a presentation and dialog model for the same.

3.4.1 Click Application to Generic UIML

The generic UIML vocabulary specifications are provided by Harmonia Inc. [66] which also provides the authoring and rendering tool support for this vocabulary. The latest version of LiquidUI™, version 3.1, by Harmonia Inc. would support rendering of a document written using this generic UIML vocabulary specification onto the Java platform. In the future, we may expect it to render the generic UIML onto the HTML and CSS platform as well. A comparison between Click's abstract representation and a UIML document shows that structure, style, content and behavior, the main building blocks of a UIML document, form a part of the Click representation as well. However, at present Click makes no attempt to clearly separate style, text content and behavior of the application. The generic UIML created would specifically be designed for the desktop family and the language would be English. Developers would then be able to modify to customize various sections according to their needs.

The reverse engineering toolset generates a separate UIML document for each page and the pages are linked by the underlying application logic. Each page in the web application is mapped to a <*part*> of class G:TopContainer, representing the overall container for the webpage. Click has a PRADO component <*com:Form*> which is a container for all the components on the page. This is mapped to a part of class G:Area and is a sub part of G:TopContainer.

3.4.2 Presentation specifications in UIML

The reverse engineering toolset generates parts and properties for all components in a page belonging to each class predefined in Click. It adds a part to the <*structure*>

section for each component with a suitable class that matches closely the Click component's class. Each property defined on a Click component has a mapping to a similar property in UIML. This property is added to the <*style*> section of the UIML document. Click supports an opportunistic behavior and allows absolute positioning of the components thus the relationship between a pair of components cannot be inferred from their arrangement in the code. This is unlike static HTML where the order of the components in the interface specification affects the rendering. It implies that the reverse engineered UIML document may not depend on the ordering of components for layout and the location aspect plays an important role.

UIML defines a concept of <*reference*> to a constant for a text value of any property. The value of a property can be a reference to a constant defined in the <*content*> section. This gives the flexibility of changing the content section once and reflecting the change in all the properties that have a reference to that section. This is particularly useful for text properties of the components, e.g. label for input field, checkboxes or radio buttons etc. All the text values associated with any component are generated as a reference to the <*content*> section of the UIML document where the actual text is specified. The <*content*> section is given an id "English" to denote that the text is in English and a similar section can be defined with a different id for another language. This makes internationalization, a long term goal of Click, of a web application very easy. However, <*reference*> in generic UIML is not yet supported by the rendering engine in LiquidUI^{TM} Version 3.1. Appendix A.7 shows the automatically UIML document without any <*content*> section.

A default style sheet (default.css), specifying the fonts and colors, is created for each application created in Click that a developer can change using Click's interface. Background color can be specified independently for each page and this property overrides that defined in the default style sheet. UIML allows defining of a background color for the containers but not fonts that may apply to all the parts in that container. So, the background color property that applies to the current page is specified as the g:bgcolor property of the G:TopContainer. Fonts are specified as properties on parts individually, if applicable.

Table 4 shows a mapping of Click's components to an equivalent class for the <*part*> section in the <*structure*> section. Table 5 shows a mapping of the properties specified on a Click's component to an equivalent property in the <*style*> section. IDs used for each of the parts are automatically generated based the IDs for a Click component. These tables only show mappings for simple components which are specified as a single part and have a straightforward mapping. A description of other components that are specified as a combination of parts and require manipulation of properties is presented next.

Table 4: Mapping of Click's components to UIML classes in Generic Vocabulary

Click/PRADO Component	UIML Class for <part>
<com:HtmlText>	G:Text
<com:Image>	G:Image
<com:InputText TextMode="SingleLine">	G:TextField (g:textfieldtype = text)
<com:InputText TextMode="MultiLine">	G:TextBox
<com:Checkbox>	G:Button (g:buttontype = checkbox)
<com:Button>	G:Button(g:buttontype = submit)

Table 5: Mapping of the properties in Click to properties in Generic UIML

Click/PRADO Component	Properties on Click Components	Properties on UIML parts
All	X, Y	g:location
<com:HtmlText>	Text	g:text
<com:Image>	Name	g:image-src
<com:InputText TextMode="SingleLine">	Columns	g:size
<com:InputText TextMode="MultiLine">	Columns	g:columns
	Rows	g:rows
<com:Checkbox>	Text	g:text
<com:Button>	Text	g:text

Radio Button List

A RadioButtonList component in Click is composed of ItemList components that define the options in a radio button list. A RadioButtonList component is mapped to a G:ButtonGroup part in UIML and each ItemList component maps to a G:Button part, of type "radio", that is a subpart of G:ButtonGroup. ID for G:ButtonGroup is directly taken

from the ID property of RadioButtonGroup whereas for G:Button parts the id is automatically generated from the id of RadioButtonGroup. “RepeatDirection” property of RadioButtonList specifies how the ItemList components are laid out on screen and its value can be one of “Horizontal” and “Vertical”. There is no equivalent property in UIML. The location property is manipulated to achieve the desired layout. For horizontal alignment of radio buttons, Y value remains constant and X value is incremented for each radio button. For vertical alignment of radio buttons, X value remains constant and Y value is incremented for each radio button. The calculated positions are an approximation which can be changed by the developer if desired. Figure 25 shows a RadioButtonList component as specified in Click and figure 26 shows its equivalent UIML representation.

```
<com:RadioButtonList ID="mrms" RepeatDirection="Horizontal" X="73"
Y="115" Z="71">
  <com:TListItem Value="Mr." >
    <prop:Text> Mr. </prop:Text>
  </com:TListItem>
  <com:TListItem Value="Ms." >
    <prop:Text> Ms. </prop:Text>
  </com:TListItem>
</com:RadioButtonList>
```

Figure 25: Click’s representation of a group of Radio Buttons

```
<part id="mrms" class="G:ButtonGroup”>
  <part id="mrms1" class="G:Button">
      <style>
        <property name="g:buttontype">radio</property>
        <property name="g:location">73,115</property>
        <property name="g:text">
          <reference constant-name="mrms1text"/>
        </property>
      </style>
  </part>
  <part id="mrms1" class="G:Button">
      <style>
        <property name="g:buttontype">radio</property>
        <property name="g:location">80,115</property>
        <property name="g:text">
          <reference constant-name="mrms2text"/>
        </property>
      </style>
</part>
```

Figure 26: Representation of the group of Radio Button in UIML

Drop Down List

DropDownList component in Click is a variant of RadioButtonList, which is expressed with a construct exactly similar to RadioButtonList. The UIML equivalent is a class G:List. Each ItemList component maps to a value in the content property of the G:List. Figure 27 shows a RadioButtonList component as specified in Click and figure 28 shows its equivalent UIML representation.

```
<com:DropDownList ID="mrms" X="73" Y="115" Z="71">
  <com:TListItem Value="Mr." >
    <prop:Text> Mr. </prop:Text>
  </com:TListItem>
  <com:TListItem Value="Ms." >
    <prop:Text> Ms. </prop:Text>
  </com:TListItem>
</com:DropDownList>
```

Figure 27: Click's representation of a Drop Down List

```
<part id="mrms" class="G:List">
  <style>
    <property name="g:location">73,115</property>
    <property name="g:content">
      <reference constant-name="mrmsoptions"/>
    </property>
  </style>
</part>
```

Figure 28: Representation of the group of Drop Down List in UIML

Data Table

DataTable component can be represented using the G:Table class of UIML. G:Table will have all the columns corresponding to the columns in the DataTable component and two columns with buttons, edit and delete, for each row. UIML allows column type to be associated with a column in a table such that each column can belong to a different class. The generic UIML does not support hyperlinks as they are particular to HTML which led to finding alternate ways to represent various links on a DataTable.

For the same reason the rows can also not be shown as links if they are linked to a "details" page in the original application. However, it is possible to import all the styles defined on the DataTable component in the G:Table of UIML. A data table can be represented in UIML as shown in figure 29.

```
<structure>
  <part id="datatable1" class="G:Table">
  <part id="datatable1.TableColumn0" class="G:ColumnDef">
    <style>
      <property name="g:width">100</property>
      <property name="g:header">Firstname</property>
    </style>
  </part>
  <part id="datatable1.TableColumn1" class="G:ColumnDef">
    <style>
      <property name="g:width">100</property>
      <property name="g:header">Lastname</property>
    </style>
  </part>
  <part id="datatable1.TableColumn2" class="G:ColumnDef">
    <style>
      <property name="g:width">100</property>
      <property name="g:columntype">button</property>
      <property name="g:header">Delete</property>
    </style>
  </part>
  </part>
</structure>
<style>
  <property name="g:location" part-name="datatable1">18,49</property>
  <property name="g:size" part-name="datatable1">409,205</property>
</style>
```

Figure 29: Representation of a data table in UIML

Login Component

Login component is composed of a text box that accepts a user name, a text box that accepts password and a login button that authenticates the user.

```
<part id="username" class="G:TextField">
  <style>
    <property name="g:textfieldtype">text</property>
  </style>
</part>
<part id="password" class="G:TextField">
  <style>
    <property name="g:textfieldtype">password</property>
  </style>
</part>
<part id="login" class="G:Button">
  <style>
    <property name="g:buttontype">submit</property>
  </style>
</part>
```

Figure 30: Representation of a Login Box in UIML

3.4.3 Behavior specifications in UIML

UIML <behavior> section is modeled as a rule-based event/action language. The behavior is defined by a set of rules that are comprised of conditions and actions. The actions are executed only when the associated conditions are evaluated to true. An action can either change certain property of an element or fire an external script or can dynamically change the user interface visual tree. UIML allows specification of how a user interface may be modified on a particular event. Click also follows the same model.

Button Click Actions

Click allows developers to specify particular actions to be performed when a button is clicked. Also, a developer can specify constraints on these actions that are defined in terms of values of input fields. Each pair of conditions and related actions is called an action rule similar to a <*rule*> in UIML. It is possible to define multiple action rules that are executed in order. This is the direct behavior specification we can infer from a Click application. The indirect behavior specifications that are not explicitly specified in connection with the button click event by an end-user developer also are an essential part of the generated application. A developer specifies input constraints on the text fields that are part of an application. The values input by the user are validated when the form is submitted i.e. when a submit button is clicked. The user is notified through an error message in case the values entered were wrong.

A <*rule*> in behavior specification of a UIML document is created for each condition action pair (i.e. actionRule) defined in the Button component's behavior XML file of Click. Validations are always performed on any submit button click prior to any other action. The typical components of a button action rule in the UIML representation generated by the toolset would be as follows:

Condition: The <condition> section will always have one condition which corresponds to the button click event. Other conditions are based on conditions defined on the button action. Comparison of values in the text field to static values, state of a checkbox and a particular item selected in a radio button list or drop down list are other conditions that can be defined on a button click event. Figure 31 is an example of a typical condition that can defined by a developer on a button in Click.

```
<condition>
  <op name="and">
    <event part-name="buttonid" class="g:actionperformed"/>
    <op name="and">
      <op name="==">
        <property part-name="checkboxid" name="g:selected"/>
        <constant value="true"/>
      </op>
      <op name="<">
        <property part-name="textfield" name="g:text"/>
        <constant value="10"/>
      </op>
    </op>
 </op>
</condition>
```

Figure 31: Specification of a Button behavior in UIML

Action: Each of the actions possible in Click is defined in the form of function calls in the equivalent UIML document. The function calls are specifically designed to canonically represent the actions to be performed. Validation actions are specified as functions of a class "validate" and have two mandatory function arguments. The "fieldToValidate" specifies the value of the input field that need to be validated and "errorMsg" specifies the message to be displayed if the value entered is wrong. Other arguments depend on the type of validations. Figure 32 shows few validations functions that will be executed once a submit button is pressed. If one of these functions fails then the function reloads the page and displays in which field the value entered was wrong.

```
<call name="validate.isLengthWithinRange">
  <param name="fieldToValidate">
    <property part-name="name" name="g:text"/>
  </param>
  <param name="errorMsg">Enter between 1 and 50 characters.</param>
  <param name="minValue">1</param>
  <param name="maxValue">50</param>
</call>
<call name="validate.isValidEmailAddress">
  <param name="fieldToValidate">
    <property part-name="email" name="g:text"/>
  </param>
  <param name="errorMsg">Please enter a valid email address.</param>
  </call>
<call name="validate.isNumberWithinRange">
  <param name="fieldToValidate">
    <property part-name="number" name="g:text"/>
  </param>
  <param name="errorMsg">Enter a number between 1 and 50.</param>
  <param name="minValue">1</param>
  <param name="maxValue">50</param>
</call>
<call name="validate.isValidDate">
  <param name="fieldToValidate">
    <property part-name="date" name="g:text"/>
  </param>
  <param name="errorMsg">Date is of format mm/dd/yy.</param>
</call>
```

Figure 32: Validation actions on a form specified as function calls

Predefined actions are specified as functions of a class "action". These actions include, send email, save inputs to database, go to another page or reload current page. Arguments to the "sendEmail" functions are the general email attributes (To, From, Subject, Body). These values are specified by the developer in Click. A developer can also specify placeholder for the email attributes, such that the "To" address is the one specified by a user. Placeholders are replaced by the actual values input in the text fields are runtime. "Save to Database" function takes a table name and a variable number of arguments for the fields that need to be stored in that database table. "Reset input fields" action is performed by setting the text property of the input fields to empty and is not specified as an external function call. Figure 33 shows calls to "sendEmail", "loadNewPage" and "saveToDatabase" functions.

```
<call name="action.sendEmail">
  <param name="To">
      <property part-name="emailaddress" name="g:text"/>
  </param>
  <param name="From">someone@mail.com</param>
  <param name="Subject">Hello</param>
  <param name="Message">Message</param>
</call>
<call name="action.loadNewPage">
  <param name="pageName">thankyou</param>
</call>
<call name="action.saveToDatabase">
   <param name="tableName">data</param>
   <param name="title">
      <property part-name="mrms" name="g:text"/>
   </param>
   <param name="firstname">
      <property part-name="firstname" name="g:text"/>
   </param>
   <param name="email">
      <property part-name="emailaddress" name="g:text"/>
   </param>
</call>
```

Figure 33: Specification of actions performed on button click

Data Table Behavior

The contents of the DataTable component are dynamically updated every time a page is reloaded. The contents are based on a SQL query that is executed on a database table. Figure 34 shows a sample XML representation that Click uses to canonically represent a SQL query. UIML allows contents of a G:Table part to be updated dynamically either through an external XML file providing the contents or an external executable like a Java module to dynamically render the UIML document. On similar lines, the reverse engineering toolset automatically generates the external Java module with a stub to connect to an appropriate database and fetching the data records for the DataTable component by using the SQL query defined in the Click component.

```
<query type="select" dbTable="data">
  <dbFieldName>destination</dbFieldName>
  <dbFieldName>departuredate</dbFieldName>
  <filters connector="and" />
</query>
```

Figure 34: XML representation used in Click for a simple database query

The dynamic behavior on a DataTable component includes sorting a column by clicking the column header, clicking on a data row for more information and clicking on "edit" or "delete" to perform respective actions. Currently, generic UIML does not define an event of clicking on a column header which rules out the possibility of defining a sort function on clicking the column header. However, when the generic UIML is rendered to a target platform this function may be specified directly. As specified earlier, a developer can define an external executable such as a Java program to dynamically render the rows of data in the data table and the behavior specifications for these rows must be defined in the external executable. The latest LiquidUI™ Version 3.1 does not illustrate an example of the use of a push button column and how events on each cell in the column are identified. However, the generated Java module can be customized to implement this behavior as well. This Java module recommends what behavior needs to be implemented by means of comments. Appendix A.9 shows the Java module for a DataTable component defined by the SQL query as in figure 34.

Authentication Behavior

Authentication is the first action that always happens when a secured page is loaded. The subroutine checks whether any user is currently logged in and if not then it prompts with a login page. The user then logs in and the authentication is performed.

```
<rule>
  <condition>
    <op name="and">
      <event part-name="mainpage" class="g:opened"/>
      <op name="==">
        <call name="authentication.isUserLoggedOn"/>
        <constant value="false">
      </op>
    </op>
  </condition>
  <action>
      <call name="Authentication.authenticateUser">
        <param name="authenticationType">LDAP_VT</param>
        <param name="authenticationType">database</param>
      <call>
  </action>
<rule>
```

Figure 35: Behavior specification for authenticating a user on a secured page

Figure 35 shows that when the first time a secured page is opened and the application does not find any user logged in, it prompts a login page to ask for credentials of the user. There are several ways a developer can implement the login page, i.e. through an externally created dialog container window or by restructuring the current UI to replace with the login box. This toolset creates a UIML document for each page separately and lets the developer implement the functionality to load the page. This happens for the login page as well.

3.4.4 <peers> section

The <*presentation*> element specifies that the vocabulary used for the UIML representation is "Generic_1.2_Harmonia_1.0" which comes with LiquidUI™ Version 3.1. The <*logic*> section is intentionally left blank to let the developer specify a suitable implementation for the <*behavior*> section. This makes the generated UIML representation platform independent capable of being mapped to other platforms in future as well. Appendix A.7 shows an automatically generated UIML document from a web page defined in Click. The UIML renderer in LiquidUI™ Version 3.1 currently renders the vocabulary "Generic_1.2_Harmonia_1.0" in Java and Appendix A.8 shows the Java interface generated from the generic UIML.

3.5 Summary

In this chapter a description of the reverse engineering toolset and various artifacts it creates was presented. The toolset, integrated with Click, automatically creates four artifacts for a web application created using Click. These artifacts are a sitemap, text documentation, a task model and a UIML representation. The rules followed for the creation of each artifact were discussed in detail. An end-user or an expert developer can choose to create these artifacts anytime through Click's interface and these artifacts will reflect the application's current structure and behavior. Each of these artifacts plays a significant role in facilitating end-user and expert developer collaboration in implementing advanced functionality in an end-user developed web application. Sitemap provides a high level overview complemented by the documentation to provide a detailed

description of the web application created by an end user. The task model is the overview of the interaction between the web application and the user of that application. These artifacts will help an expert developer easily take over the web application for further maintenance and development. Task model and UIML representation can be used to generate MPUIs by following the forward engineering approach suggested by Ali et al. [34].

Chapter 4: Formative Evaluation

4.1 Purpose of Evaluation

The main aim of this evaluation was to perform a usability study with few web application developers using the artifacts generated by the reverse engineering toolset. It was important to find out if the expert developers can understand the functionality of a web application developed by an end-user developer using Click. The developers should also be able to make enhancements to the application using these artifacts. The assumption was that the end-user developer and the expert web developer may not interact in some realistic setting so that the generated artifacts are the only source of information. This evaluation did not intend to compare development using the generated artifacts with development in Click's environment. The design goals for both the approaches are clearly separate and are intended to be complementary.

4.2 Participants

The target audience for this research is web application developers who have a fair knowledge of client side and server side scripting languages and know how web based database centric applications function. The participants recruited reflected this group. The participants were graduate students from the Department of Computer Science at Virginia Tech and were recruited by contacting through graduate student's listserv. The evaluation was performed with five participants, 3 males and 2 females.

4.3 Limitations

One of the major limitations of this evaluation was that the model-based approach is still not widely accepted by professional UI developers. Hierarchical task analysis is a well known practice in the HCI community. But, the CTT notation, used widely in the model-based methodologies and in this research to represent the task model, is not yet a mainstream technique. Similarly for UIML, which is a relatively recent technology, it was hard to find developers who had some experience with this language. To overcome this limitation a short tutorial on both the technologies was provided at the beginning of the evaluation.

4.4 Evaluation Protocol

Participants were recruited after the experiment was approved by the Institutional Review Board (Refer to Appendix B for the IRB approval letter). Recruited participants were provided with an Informed Consent Form that detailed the experiment procedures and the participant's responsibilities. All prospective participants were asked to read the Informed Consent and sign only if they are in agreement with it. Participants had the choice to leave at any time during the experiment. After the Informed Consent Form was signed by the participants, an evaluation pre-questionnaire was given for assessing their experience with web application development (Refer to Appendix C.1 for evaluation pre-questionnaire).

4.4.1 Training

Participants were given a brief overview of the whole evaluation that explained what they will be doing. Each participant was given a short tutorial of CTT and UIML. The tutorial on CTT explained the graphical notations. Next, a short tutorial on UIML was given explaining the main concepts of the language. Participants were given enough time before performing the tasks so that they felt reasonably comfortable with these technologies. The training took an average of 20 minutes. Refer to appendices D.1 and D.2 for tutorials used in the evaluation.

4.4.2 Evaluation Application

The test application was a simple conference registration system. The homepage of the application asks for name, email address and a checkbox to select whether the registration application sends an email to the user or not. On clicking the "Register" button, all this information is saved to a database. If the checkbox is selected a confirmation email is sent and a thank you page with this message is shown; otherwise just a different thank you page is shown. This application has the basic functionality sufficient to test the understanding of various concepts while it purposely avoids more complex features of Click. This was required in order to avoid complicating the generated artifacts and overwhelming the participants as they learnt new languages and representations on the spot. Using more complex features would have meant more

elaborate and time-consuming training for the participants. In addition to this, the UIML code generated by the reverse engineering toolset is based on similar constructs for all the components as defined in the UIML specifications [38]. These include creating parts, defining their properties and defining behavior through external function calls. The developers should be able to understand the program as they become familiar with UIML.

4.4.3 Procedure

Participants were given a printed copy of the sitemap, the task model, the documentation for every application page, the generic UIML document for each page and the training material for reference. All the documents were also made available on the computer. This was specifically required for the sitemap as the sitemap is an interactive feature and it allows developers to navigate through the web applications' components. Participants were then given the task sheet consisting of all the tasks they were required to perform (Refer Appendix C.2 for the task sheet.).

Participants were encouraged to think-aloud and all the critical incidents and comments were recorded. Participants were asked to mark areas of importance in these documents for each of the tasks using task numbers as well as they were encouraged to write what they consider relevant to performing the tasks. Also tasks were given on a sheet of paper where space was provided for them to write. Participants were asked to perform the tasks based on the diagrams and documentation provided to them. However, they were allowed to interact with the application in Click to find out at what point they absolutely needed to look into the application's code and why. After the tasks were performed, a general satisfaction questionnaire was presented with the purpose of finding how the developer felt about the whole development process (Refer Appendix C.3 for the satisfaction questionnaire). Each experiment lasted about one hour and tasks were not timed.

4.5 Evaluation Results

The participants were experienced web developers having web application development experience between 1-5 years. All the participants were experienced with at

least one server side language, i.e. Perl, ASP, JSP and Java Servlets. All the participants had low experience with PHP and they rated themselves above 6 or above in their experience with Java programming. This gives an idea about the multitude of platforms people deal with and that there is a need for some standard. In summary, the participants were able to understand the artifacts and perform all the tasks correctly although it took them some time to get familiarized with the notations used. All participants found it useful to start with some high level information i.e. sitemap. They commented that the sitemap shows flow of information between various entities of the web application. All the participants had their own understanding of what the application should do based on their experiences. For some, it did not match with the actual functionality of the application. They were able to identify areas of improvement in the application just by looking at the sitemap and the task model. Following sections describes each of the tasks in detail and how participants performed on those tasks. Table 6 shows a summary of the representations used by each participant to perform the tasks. For task list refer to Appendix C.2.

Table 6: Representations used by the participants to accomplish tasks

	Participant #1	Participant #2	Participant #3	Participant #4	Participant #5
Task 1	Sitemap + task model	Sitemap + task model	Sitemap + UIML	Sitemap	Sitemap + task model
Task 2	Task model	Task model	UIML	Documentation + Sitemap	Sitemap
Task 3	Task model	Documentation + Sitemap	UIML	Documentation	Sitemap
Task 4	UIML	UIML	UIML	UIML	UIML
Task 5	UIML	UIML	UIML	UIML	UIML
Task 6	UIML + task model	UIML	UIML	UIML + task model	UIML

4.5.1 Design Comprehension Tasks

Task 1

Task 1 asked them to explain what an application does in as much detail as they can. For this task, all the participants felt that sitemap gave them a jumpstart but they needed more information to clarify some aspects about what happens on each page before navigation is made between different entities. 3 out of 5 participants then turned to task models for more detailed information. These participants were of the opinion that they would rather not look at the code, i.e. UIML representations, until they can find relevant information in diagrams. Participant #5 tried to understand the sitemap and the relevant pages and entities it linked to for getting as much detailed understanding of the application. Participant #3 jumped straight into the UIML code, instead of looking into the task model or documentation, to carefully understand the notation and understand the program flow.

Task 2

Task 2 asked the participants to find out what actions the "Register" button performs. The participants spent a considerable amount of time in task 1 trying to understand all the different aspects of the application from the sitemap and task models. This made performing task 2 very simple. They were able to clearly define what happens once a button on the web page is clicked. However, 3 out of 5 participants could not define in detail what input validations are performed on each of the input fields from the sitemap and task models but said they would expect it to be in the code, i.e. UIML representation. Participant #3 identified that all the actions on the button to be in the <behavior> section of UIML and could even find out what input validations are performed, in full detail. Participant #4 read the documentation in order to find out the details about input validations.

Task 3

For task 3, which asked about the data being stored in the database, 3 out of 5 people wanted to look at the database schema to answer this question. Sitemap did give an overview that the data is being stored in a particular database and the database schema

was available by clicking on the database shapes. However, 2 of these participants did not notice that the shapes on the sitemap were clickable despite being mentioned on the caption. Only one of the participants was easily able to find out the database schema by clicking on database icon on the sitemap. Participant #4 was able to find out the database schema automatically generated as a part of the documentation and linked through the documentation pages. However, he commented that the documentation was not an easy read and can be organized in a better way but couldn't give any suggestions for improvement.

4.5.2 Modification Tasks

Task 4, 5 and 6 were modification tasks. These tasks were very simple for the participants. The reason was that once the participants were explained the concept of separation of concerns in UIML they could very easily apply that to perform the tasks. For each of the tasks, all the participants were able to locate the correct relevant sections where they would make changes. For making the actual change they indicated that they would be learning by example and try to find similar feature already implemented and replicate that. Task 4 and 5 were simple changes to the UIML document. They understood the concept of the <peers> section in UIML and how an implementation can be externally provided to the function calls used in the behavior section. 2 participants mentioned that the function calls that were defined in the generic UIML could serve as good design specifications for them to implement these functions. One of the participants wanted to see some canonical representation for the implementation part also that may specify at a high level what the function does.

Task 6 involved adding new functionality to the interface. This task also was correctly performed by all the participants. They were able to add new parts and define new functionality in the behavior section of the UIML document. Two of the participants also mentioned that they would need to change the task model as well because the new functionality would affect the user tasks. But they said they may not be able to do so, at that moment, as they were not very comfortable with CTT yet.

4.6 General comments and satisfaction ratings

Table 7 summarizes the ratings given by the participants on how easy the generated representations were to understand, 5 being "very easy" and 1 being "very difficult". Table 8 shows the ratings given by the participants on how useful the generated representations were in understanding and modifying the web application, 5 being "very useful" and 1 being "not at all useful". A details description of results and comments made by the participants are presented next.

Table 7: Satisfaction ratings on how easy it was to understand the reverse engineered representation

	Participant #1	Participant #2	Participant #3	Participant #4	Participant #5
Sitemap	5	5	5	5	4
Documentation	5	5	-	3	4
Task model	3	5	-	3	3
UIML	3	5	5	4	3

Table 8: Satisfaction ratings on how useful the reverse engineered representations were in understanding and modifying the web application

	Participant #1	Participant #2	Participant #3	Participant #4	Participant #5
Sitemap	4	5	3	5	4
Documentation	2	5	-	5	3
Task model	5	5	-	4	4
UIML	4	5	5	4	5

Participants rated the sitemap as the easiest to understand and very helpful in understanding the behavior of the application. Documentation was rated easy to understand by three participants but they mentioned that they did not actually read it and that it was not very useful. Participant #3 did not use the documentation and task models in the experiment and therefore did not rate them. Participant #4 suggested that the documentation was difficult to read but was useful for understanding the application. Task models were rated very easy to understand by only one participant. Other

participants found the generated task model somewhat easy to understand but only after they had an overview of the application from the sitemap. They said that the original notation is complex and they may need more time to understand it.

The UIML representation was also rated very easy to understand and useful in understanding the application with the exception of participant #1 and #5 who rated it to be difficult to understand. However, all of them mentioned that they would need more practice and more detailed specifications of UIML to be able to implement more functionality. Three of the participants did not refer to the UIML document until they had to perform the modification tasks. Participant #1 said that the task model has the right level of detail for him to understand what the application does. In general, all the participants appreciated the UIML approach to UI programming and that separation of concerns allows them to only focus on aspects relevant to their tasks. Two of the participants were interested in learning more features as they thought that this technology would be very useful for their work. They considered programming in UIML better than traditional UI programming. They mentioned that traditional programming was too flat and is overwhelming when first designing a UI. Although creating a UI in UIML may be overwhelming as well when developing without any authoring support, it makes future changes easier. Only exception to this was participant #3 who did not consider that programming in UIML would be better than in Java or HTML and PHP. Participant #5 said that if she has to only program in Java it may not be that useful as Java also has some notion of separation between layout and behavior related to event specifications. But for the web it will be very useful as instead of learning many technologies required for creating a web application like HTML, DHTML, CSS, JavaScript, PHP etc. she would learn UIML and use Java for the application logic.

Participants indicated some areas of improvement for this work. An integrated environment where models and diagrams are logically connected to each other was recommended. Participants wanted to click on the relevant pieces of information in the sitemap and the task models to find the information related to it. This model was somewhat adopted in the sitemap and documentation where each piece of information in the documentation is connected to its details. Participant #3 wanted to see all the information related to a single part in the UIML representation at one place rather than

separated in different sections. This was mainly to make it easier locate all the information about one part. This is also possible in UIML but it becomes difficult to cleanly separate the different sections and have multiple such sections in one document. Another interesting suggestion was to present the UIML as a color coded document. Harmonia provides an IDE, UIML Development Tool as a part of LiquidUI^{TM} suite, which can be used for this purpose.

Chapter 5: Conclusions and Future Work

5.1 Conclusions

This research proposed a toolset to reverse engineer web applications developed by end-user non-programmers using Click. The goal of this toolset was to create representations that the expert developers can use to understand the web applications developed by end-users and assist them in enhancing their applications beyond the capabilities of Click. The motivation behind this research was to bridge the technical gap between end-user developers and their expert counterparts during the process of transitioning the development work. Another important motivation was to support development of MPUIs for the web applications developed by end-users. Based on review of the literature, certain representations were chosen for this purpose. A web application design is expressed by means of a sitemap, text documentation, task model and an alternate representation using generic UIML. The high level sitemap shows main entities of the application and the navigations between them. Documentation for the web application is a textual representation of the features implemented by an end-user. A web application is automatically reverse engineered into the model-based framework by creating a task model represented using CTT notation, a presentation model and a dialog model using UIML.

The high level design is captured in a sitemap, text documentation and task models that the expert developers can use to understand the interface design. The design of the sitemap was based on a study of non-programmers [2] so it would be useful for not only expert developers but also the end-users. They can use the sitemap to analyze their product, make it publicly available and even discuss it with other stakeholders. Text documentation is also expressed in end-users language and can also serve as a tool for discussion between the end-users and expert developers. This is just an added benefit of this research and we did not evaluate the use of sitemap with end-user developers as the target audience was expert web developers. The developers experienced with task modeling can use task models to evaluate the usability of the web applications too.

Ali et al. [34] described in detail the use of task models and generic UIML for building MPUIs. The reverse engineering toolset creates task model which can be used to create MPUIs by following their approach. The toolset also creates generic UIML for the desktop family which can be directly rendered on to multiple desktop platforms using LiquidUI™ (when the future versions of this tool will support this). Automatic forward engineering from these models may not be very successful in producing good quality interfaces. Designers would have to intervene and customize these models for their requirements. This work, therefore, did not attempt to provide a single click solution to automatically port a web application to multiple platforms. There is no alternative for the human expertise and design decisions. There are various heuristics involved in developing multi-platform web applications and research has shown that it is not advisable to provide automatic solutions as the results are generally not satisfactory. The generated generic UIML document would provide a good start for the developers that can be customized for other platforms by adding or editing properties. Similarly, the functional specifications for the behavior section are only recommendations that were acceptable to the participants in the evaluation. The specification is also flexible and can be changed as per the preferences of any developer group.

The UIML representation is reverse engineered from the web platform has a flavor of the web interface. It was important to capture as much detail as possible while reverse engineering it in order to avoid loss of details upon abstraction. The generic UIML for the desktop family however makes it easy capture all the properties defined for the web application. The most significant issue that can be identified is the usage of absolute positioning which can make it difficult to port on other platforms. However, a developer has the flexibility to change the properties to tailor a UI according particular requirements. Appendix A.8 shows the sample web application converted to a Java interface using the generic UIML without modifying any properties. This was possible because the two platforms were related to the desktop family. UIML also makes internationalization of web interfaces generated by Click possible.

A usability evaluation with web developers showed that the generated artifacts helped the developers easily understand a web application developed in Click. They were also able to make modifications to the web application that wouldn't have been possible

for an end-user using Click. We would expect the development to become much easier as the developers become familiar with the language and its capabilities and limitations. Documentation for the web application received poor reviews during the usability evaluation. This was mainly because the developers preferred diagrams as opposed to reading text. For finding information that is at a lower level of detail than in a task model and the sitemap the developers looked at the UIML code which provided more information.

5.2 Future Work

There is a lot of scope for future research in this area. A requirement came from the usability evaluation to have all the models and UIML representations logically connected to each other. There is gradual increase in the level of details in the models and developers would find it useful if the high level models are somehow connected to the low level details corresponding to a particular feature. The sitemap is already linked to the actual entities in the web application and it will be possible to link it to the documentation as well. The graphical notation for CTT, provided by the CTTE environment, may be difficult to be linked it in an online environment. However, we can use GraphViz to produce a comparable representation as provided by CTTE and make it interactive. We can also explore the possibility of giving a front end to UIML renderer through web by using an API for dynamically creating renderers provided by Harmonia. A developer may edit the UIML code for each page as they would edit Click's code in Click's environment, possibly using a color coded editor. They can then use the rendering option for creating UIs. A more comprehensive solution, if desired by developers, could be to enhance Click to also generate the backend code in Java, which would be a very trivial task.

By reverse engineering a web application into the model-based framework we can benefit from the research going on in this area. Auto help generation from only task models is an interesting possibility studied by Pangoli and Paternò [60]. As task models are automatically recovered from an end-user developed web application, these can be used to automatically generate help for the web applications also. This will certainly be more useful than the documentation currently generated not only for developers but also

for the users of the application. Next avenue for model-based UI design is context-aware UIs and web applications are an important target. Developers of web applications will certainly be faced with a problem of seamlessly migrating their applications over different platforms and across different contexts of use. Modifying task models, expressed in CTT notation, for multiple contexts of use has been studied under the Cameleon Project. Future versions of UIML may also explore this possibility.

The reverse engineering toolset can be considered to be one step in the direction of providing support for natural development MPUIs for web based applications. End-users create web applications in an environment that uses simple ways to express relevant concepts and hides technical complexities that impede in empowering end-users. The reverse engineering toolset automatically transforms these applications into a task model and UIML representation. The forward engineering approach suggested by Ali et al. [34] can be used for building MPUIs using task models and UIML representations while ensuring their usability across different platforms. What remains to be investigated is how successful UIML is, in achieving its goal of having even non-programmers end-users, the target audience of Click, be able to design interfaces.

References

1. Nardi, B., Miller J., *An ethnographic study of distributed problem solving in spreadsheet development*, Proceedings CSCW'90. 7--10 October, Los Angeles, CA. Pp. 197--208.
2. Newman, M. W., Landay, J. A., Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice., in Designing Interactive Systems, DIS 2000, New York City, August 2000
3. Myers, B., Hudson, S., Pausch, R. *Past, Present, Future of User Interface Tools*. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.
4. Szekely, P., *Retrospective and Challenges for Model-Based Interface Development*. 2nd International Workshop on Computer-Aided Design of User Interfaces, Namur, Namur University Press.
5. Paternò, F., *Model-based tools for pervasive usability*. Interacting with Computers, 2004, pp:1–25.
6. Moore M., *Representation Issues for Reengineering Interactive Systems*, ACM Computing Surveys, 1996, 28(4), 199−es.
7. Myers, B.A. and Rosson, M.B., *Survey on user interface programming*, in proceedings of CHI '92, 195-202.
8. Foley, J., Sukaviriya, N., *History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Development*, in F. Paternò (ed.) Interactive Systems: Design, Specification, Verification, pp. 3-14 Springer Verlag, 1994.
9. Foley, J.D., Kim, W.C., Kovacevic, S., Murray, K.: *UIDE - An Intelligent User Interface Design Environment*. In Sullivan J.W., Tyler S.W. (eds.): Intelligent User Interfaces. New York: ACM Press 1991 (pp. 339-384).
10. Luo, P., Szekely, P., Neches, R., *Management of interface design in HUMANOID,* in Proceedings of InterCHI'93, ACM Press, New York , April 1993, pp.107-114.
11. Szekely, P., Luo, P., Neches, R, *Facilitating the Exploration of Interface Design Alternatives: The Humanoid Model of Interface Design*, in Bauersfeld P., Bennett J., Lynch G. (eds.): Proceedings of CHI'92. New York: ACM Press 1992 (pp. 507- 514).

12. Szekely, P., Luo, P., Neches, R., *Beyond Interface Builders: Model-Based Interface Tools*, in Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proceedings of INTERCHI'93. New York: ACM Press 1993 (pp. 383-390).
13. Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., Salcher, E., *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*, in Engineering for Human-Computer Interaction, Chapman & Hall, London, UK, 1996, pp. 120-150.
14. Wilson, S., Johnson, P., Kelly, C., Cunningham, J., Markopoulos, P., *Beyond Hacking: A Model-based Approach to User Interface Design*. Proceedings HCI'93. pp.40-48, Cambridge University Press, 1993.
15. Markopoulos, P., Pycock, J., Wilson, S., Johnson, P., *Adept - A task based design environment*, in Proceedings of the 25th Hawaii International Conference on System Sciences, IEEE Computer Society Press, 1992, pp. 587-596.
16. Wilson, S., Johnson, P., *Empowering users in a task-based approach to design, Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, in Proceedings of the conference on Designing interactive systems: processes, practices, methods, & techniques, ACM Press, New York, 1995, pp. 25-31.
17. Johnson, P., Wilson, S., Markopoulos, P., Pycock, J., *ADEPT-Advanced Design Environment for Prototyping with Task Models*, in Proc. of InterCHI'93, Amsterdam, 24-29 April 1993, ACM Press, New York, pp. 56.
18. Vanderdonckt, J., *Automatic Generation of a User Interface for Highly Interactive Business-Oriented Applications*, in Plaisant C. (ed.): Companion Proceedings of CHI'94. New York: ACM Press 1994 (pp. 41 & 123-124).
19. Vanderdonckt, J., *Knowledge-Based Systems for Automated User Interface Generation: the TRIDENT Expierence*, Technical Report RP-95-010. Namur: Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique 1995. Available at http://www.info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-010.
20. Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.

21. Puerta, A.R., *The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development*, In Proc. of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI'96, Namur, 5-7 June 1996, Presses Universitaires de Namur, Namur, 1996, pp. 19-36.
22. Puerta, A.R., Maulsby, D., *MOBI-D: A Model-Based Development Environment for User Centered Design*, in proceedings of CHI'97, ACM Press, Atlanta, March 1997, pp. 4-5.
23. Thevenin, D., Coutaz, J., *Plasticity of User Interfaces: Framework and Research Agenda,* in Proceedings of Interact99, Edinburgh, A. Sasse & C. Johnson Eds, IFIP IOS Press Publ., 1999, 110–117
24. Calvary, G., Coutaz, J., Thevenin, D., *A Unifying Reference Framework for the Development of Plastic User Interfaces,* Proceedings of Engineering HCI, May 11-13, 2001, Toronto, Canada.
25. Puerta, A.R., *A Model-Based Interface Development Environment*. IEEE Software, 1997, 40-47.
26. Puerta, A.R., Eisenstein, J., *Towards a General Computational Framework for Model-Based Interface Development Systems*. IUI99: International Conference on Intelligent User Interfaces, pp.171-178, ACM Press, January 1999
27. Sukaviriya, P. N., Kovacevic, S., Foley, J. D., Myers, B. A., Olsen Jr., D. R., Schneider-Hufschmidt, M., *Model-Based User Interfaces: What Are They and Why Should We Care?*, in Proceedings UIST'94, November 1994, pp133-135.
28. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D., *USIXML: A User Interface Description Language for Context-Sensitive User Interface,* in Developing User Interfaces with XML: Advances on User Interface Description Languages, a Satellite Workshop of Advanced Visual Interfaces. 2004. Gallipoli, Italy, 55-62
29. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., *UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence*, in Proceedings of Workshop on Device Independent Web Engineering DIWE'04 (Munich, 26-27 July 2004), M. Lauff (Ed.), Munich, 2004

30. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., *UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces*, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004)
31. Abrams, M., Phanouriou, C., *UIML: An XML Language for Building Device-Independent User Interfaces*. in XML'99. 1999. Philadelphia.
32. Ali, M.F., Abrams, M., *Simplifying construction of multi-platfrom user interfaces using UIML*, in European Conference UIML. 2001. Paris: Harmonia & Aristote.
33. Ali, M.F., Pérez-Quiñones, M.A., Abrams, M., Shell, E., *Building Multi-Platform User Interfaces with UIML,* in CADUI. 2002. France.
34. Ali, M.F., Pérez-Quiñones, M.A., *Using task models to generate multi-platform user interfaces while ensuring usability,* in CHI 2002 extended abstracts on Human factors in computing systems, Minneapolis, Minnesota, USA, ACM Press, pp. 670-671.
35. Plomp, C., Mayora-Ibarra, O., *A generic widget vocabulary for the generation of graphical and speech-driven user interfaces*, International Journal of Speech Technology, 2002, 39-47.
36. Puerta, A., Eisenstein, J., *XIML: A Universal Language for User Interfaces*. 2001, RedWhale Software.
37. World Wide Web Consortium, *XForms - The Next Generation of Web Forms,* http://www.w3.org/MarkUp/Forms
38. *UIML3.0 Draft specification*, http://www.uiml.org/specs/uiml3/DraftSpec.htm
39. Paternò, F., Mancini, C., Meniconi, S., *ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models,* in Interact. 1997. Sydney: Chapman&Hall, 362-369
40. Mori, G., Paternò, F., Santoro, C., *CTTE: Support for Developing and Analyzing Task Models for Interactive System Design,* in IEEE Transactions on Software Engineering August 2002, 797-813.
41. Paternò, F., Santoro, C., *One Model, Many Interfaces,* in proceedings of CADUI 2002, France.
42. Paternò, F., Santoro, C., *A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms,* in Interacting with Computers 15 2003, 349-366.

43. Mori, G., Paternò, F., Santoro, C., *Tool Support for Designing Nomadic Applications*. in proceedings of IUI 2003 Miami, Florida.
44. Mori, G., Paternò, F., Santoro, C., *Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions,* in IEEE Transactions on Software Engineering 2004, 507-520.
45. Berti, S., Correani, F., Paternò, F., Santoro, C., *The TERESA XML Language for the Description of Interactive Systems at Multiple Abstraction Levels,* in Proceedings Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, May 2004, pp.103-110
46. Marucci, L., Paternò, F., Santoro, C., *Supporting Interactions with Heterogeneous Platforms through User and Task Models***,** in Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces, H. Javahery and A. Seffah (eds.), pp.217-238, Wiley and Sons, 2003.
47. Bandelloni, R., Paternò, F., *Migratory User Interfaces able to Adapt to Various Interaction Platforms,* International Journal of Human – Computer Studies, 60, pp. 621-639. Elsevier, 2004.
48. Trewin, S., Zimmermann, G., Vanderheiden, G., *Abstract User Interface Representations: How well do they Support Universal Access?,* in CUU. 2003. Vancouver, British Columbia, Canada, 77-84
49. Benyon, D. R., *Domain Models in User Interface Design,* In Benyon, D. R. and Palanque, P. Critical Issues in User Interface Systems Engineering, 1996, Springer-Verlag.
50. Souchon, N., Vanderdonckt, J., *A Review of XML-Compliant User Interface Description Languages*, in *DSV-IS*. 2003. Berlin: Springer-Verlag, 377-391
51. Bouillon, L., Vanderdonckt, J., Chow, K.C., *Flexible re-engineering of web sites.* in proceedings of the 9th international conference on Intelligent User Interface, 2004, Funchal, Madeira, Portugal, 132-139
52. Vanderdonckt, J., Bouillon, L., Souchon, N., *Flexible Reverse Engineering of Web Pages with VAQUISTA*, Proceedings of the IEEE 8th Working Conference on Reverse Engineering. Stuttgart, October 2-5, 2001. IEEE Press, pp. 241-248.

53. Bouillon, L., Vanderdonckt, J., Eisenstein, J., *Model-Based Approaches to Reengineering Web Pages*, in Proceedings of 1st International Workshop on Task Models and Diagrams for user interface design TAMODIA 2002
54. Bouillon, L., Vanderdonckt, J., Souchon, N., *Recovering Alternative Presentation Models of a Web Page with VAQUITA*, in CADUI. 2002. France.
55. Paganelli, L., Paternò, F., *Automatic Reconstruction of the Underlying Interaction Design of Web Applications*, Proceedings of SEKE 2002, Ischia, Italy, July 2002.
56. Paganelli, L., Paternò, F., *A Tool for Creating Design Models from Web Site Code*, International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), pp. 169-189, 2003.
57. Di Lucca, G.A., Di Penta, M., Antoniol, G., Casazza, G., *An approach for Reverse Engineering of Web-Based Applications*, 8th Working Conference on Reverse Engineering WCRE2001, Stuttgart, Allemagne, 5-7 October, IEEE Press, Los Alamitos, 2001
58. Ricca, F., Tonella, P., *Web site analysis: Structure and evolution*. In Proceedings of the International Conference on Software Maintenance, pages 76-86, San Jose, California, USA, 2000
59. Berti, S., Paternò, F., Santoro, C., *Natural Development of Ubiquitous Interfaces,* Communications of the ACM, September 2004, pp.63-64, ACM Press.
60. Pangoli, S., Paternò, F., *Automatic Generation of Task-oriented Help*, Proceedings UIST'95, pp. 181-187, ACM Press, 1995.
61. Rode, J., M. B. Rosson, *Programming at Runtime: Requirements & Paradigms for Nonprogrammer Web Application Development*, IEEE VL/HCC 2003. Auckland, NZ.
62. Rode, J., Bhardwaj, Y., Pérez-Quiñones, M. A., Rosson, M. B., Howarth, J., *Click: Component-based Lightweight Internet-application Construction Kit*, 2005, http://phpclick.sourceforge.net
63. Rode, J., Bhardwaj, Y., Pérez-Quiñones, M. A., Rosson, M. B., Howarth, J., As Easy as "Click": End-User Web Engineering. International Conference on Web Engineering 2005. Sydney, Australia. July 27-29
64. Xue, Q., *The PRADO Framework*, 2005, http://www.xisc.com
65. AT&T., *Graphviz - Graph Visualization Software*, 2005. http://www.graphviz.org/

66. Harmonia Inc., *Liquid UI*, 2005, http://www.harmonia.com

67. CAMELEON Project Homepage, http://giove.cnuce.cnr.it/cameleon.html

Appendix A: Sample Application

This section shows a sample application built in Click which was used in the formative evaluation as well. The original code for the application, using the PRADO framework, as well as the generated artifacts is presented. As the reverse engineering toolset generates the artifacts for each page, this appendix shows an example of the homepage of the application.

A.1 Homepage of a Registration Application

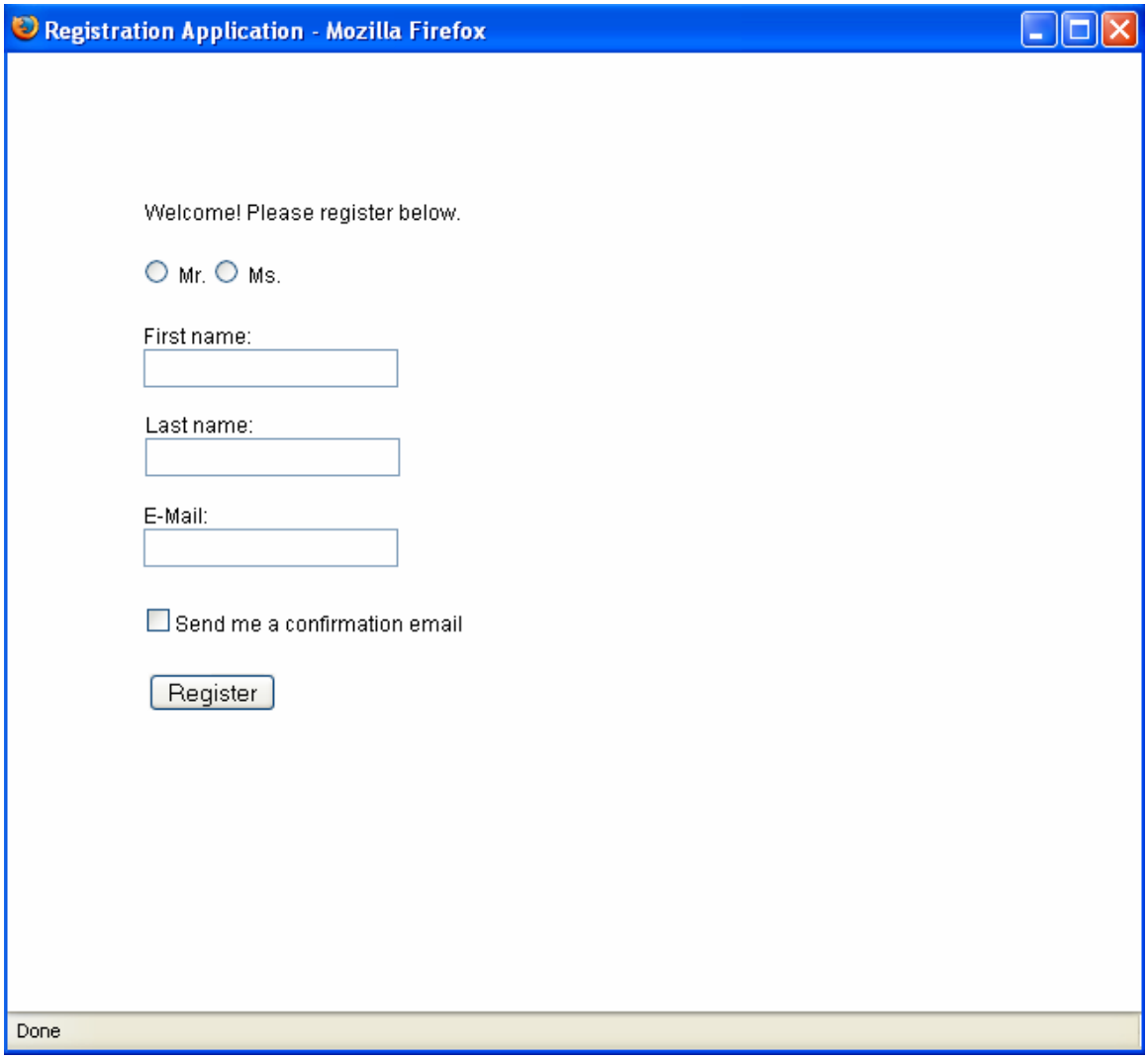


Figure 36: Home page of the "Registration Application" built in Click by an end-user

A.2 Layout code for the homepage

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Registration Application</title>
  <link rel="stylesheet" type="text/css" href="styles/default.css">
</head>
<body>
<com:Form>
<com:HtmlText ID="htmltext1" X="78" Y="83" Z="78">
  <prop:Text>Welcome! Please register below.</prop:Text>
</com:HtmlText>
<com:RadioButtonList ID="mrms" RepeatDirection="Horizontal"
RepeatColumns="2"" DbFieldName="data:mrms" X="73" Y="115" Z="71">
  <com:TListItem Value="Mr." >
    <prop:Text> Mr. </prop:Text>
  </com:TListItem>
  <com:TListItem Value="Ms." >
    <prop:Text> Ms. </prop:Text>
  </com:TListItem>
</com:RadioButtonList>
<com:InputText ID="firstname" X="77" Y="154" Z="70" Columns="20"
Rows="1" TextMode="SingleLine" DbFieldName="data:firstname"
InputRequired="false">
  <prop:Label>First name:</prop:Label>
</com:InputText>
<com:InputText ID="lastname" X="78" Y="205" Z="69" Columns="20"
Rows="1" TextMode="SingleLine" DbFieldName="data:lastname"
InputRequired="true" ValueType="Characters" MinValue="1"
MaxValue="50">
  <prop:Label> Last name:</prop:Label>
  <prop:ErrorMessage>Please enter between 1 and 50
characters.</prop:ErrorMessage>
</com:InputText>
<com:InputText ID="email" X="77" Y="257" Z="68" Columns="20" Rows="1"
TextMode="SingleLine" DbFieldName="data:email" InputRequired="true">
  <prop:Label> E-Mail:</prop:Label>
  <prop:ErrorMessage>Please enter a valid e-mail
address.</prop:ErrorMessage>
  <prop:RegularExpression>\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*</prop:RegularExpression>
</com:InputText>
<com:Checkbox ID="sendemail" Checked="false" TextAlign="Right"
DbFieldName="" X="75" Y="315" Z="79">
  <prop:Text>Send me a confirmation email</prop:Text>
</com:Checkbox>
<com:Button ID="registerbutton" Text="Register" EncodeText="false"
OnClick="registerbutton_runActions" X="80" Y="355" Z="80"/>
</com:Form>
</body>
</html>
```

Figure 37: Layout code generated by Click for the homepage using the PRADO framework.

A.3 Behavior Logic in Click

```
function registerbutton_runActions($button, $parameter) {
  $this->runAction('saveToDatabase','homepage');
  $condition1 = $this->newCondition('{sendemail}','checked');
  if ($condition1->isTrue())
  {
    $this->runAction
('sendEmail','yogitab@vt.edu','{email}','Registration
Confirmation','Thank you for your registration.');
    $this->runAction('goToPage','thankyou');
  }
  $condition2 = $this->newCondition('{sendemail}','unchecked');
  if ($condition2->isTrue())
  {
    $this->runAction('goToPage','thankyounoemail');
  }
}
```

Figure 38: Behavior code generated by Click in PHP for "Register" button on homepage

```
<ButtonActionRules>
 <ActionRules Id="registerbutton">
  <actionRule>
   <actions>
    <actionSaveToDatabase saveToAllPages="false" />
   </actions>
  </actionRule>
  <actionRule>
   <conditions connector="and">
    <condition fieldId="sendemail" operator="checked" parameter=""/>
   </conditions>
   <actions>
    <actionGoToPage pageId="thankyou" />
   </actions>
  </actionRule>
  <actionRule>
   <conditions connector="and">
    <condition fieldId="sendemail" operator="unchecked"
    parameter=""/>
   </conditions>
   <actions>
    <actionGoToPage pageId="thankyounoemail" />
   </actions>
  </actionRule>
 </ActionRules>
</ButtonActionRules>
```

Figure 39: XML representation stored by Click for "Register" button actions

A.4 Sitemap for the application

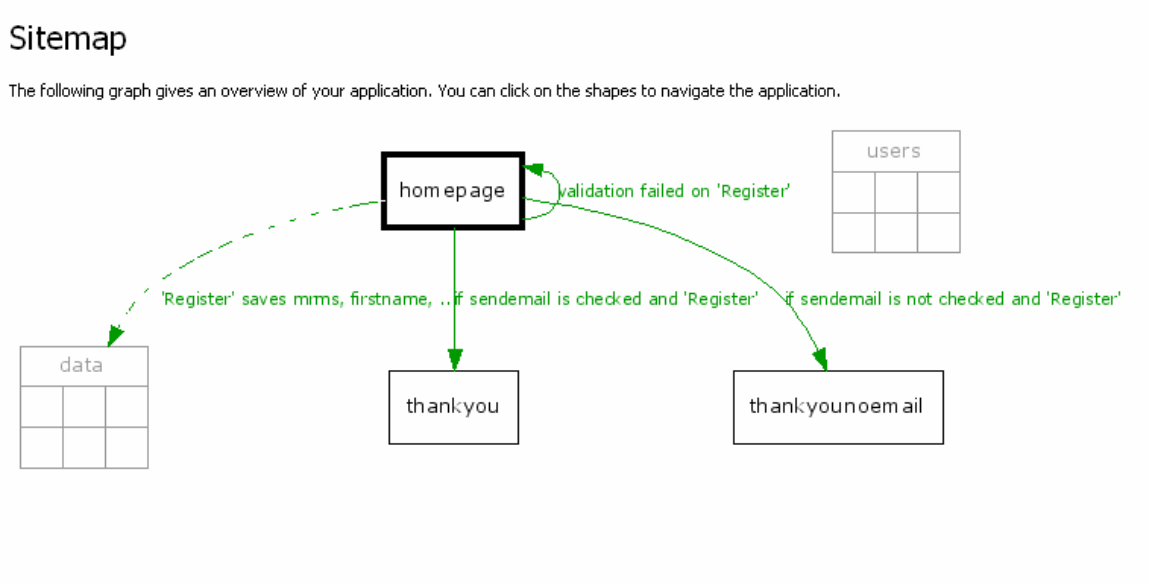


Figure 40: Sitemap for the whole "Registration Application"

A.5 Documentation for the Home Page

Evaluation Application- Page: homepage

This is the home page of the application, Evaluation Application

This page requires the user to fill up a form. Following are the form fields:

firstname: Text Input Field
Database Connection:
Table: data Field: firstname
Constraints: This is an optional field.
firstname should be a string having between 1 and 30 characters.
Filling this field incorrectly will generate the error message "Please enter between 1 and 30 characters."

lastname: Text Input Field
Database Connection:
Table: data Field: lastname
Constraints: This field is a mandatory field
lastname should be a string having between 1 and 50 characters.
Filling this field incorrectly will generate the error message "Please enter between 1 and 50 characters."

email: Text Input Field
Database Connection:
Table: data Field: email
Constraints: This field is a mandatory field

email is an email address.
Filling this field incorrectly will generate the error message "Please enter a valid e-mail address."

sendemail: Checkbox
Database Connection:
This field is not linked to any database.

mrms: Radio Button
Database Connection:
Table: data Field: mrms
Options (Text : Value):
Mr.:Mr.
Ms.:Ms.

On clicking **Register** the form is validated and if any error is found the user is prompted to correct the input. Once the form is validated following actions happen:
=> Always
Save data from current page to database Data from the following input fields are saved to table data: mrms, firstname, lastname, email

=> If "sendemail" is checked
Go to page "thankyou". Find the documentation here

=> If "sendemail" is not checked
Go to page "thankyounoemail". Find the documentation here

A.6 Task Model for the home page

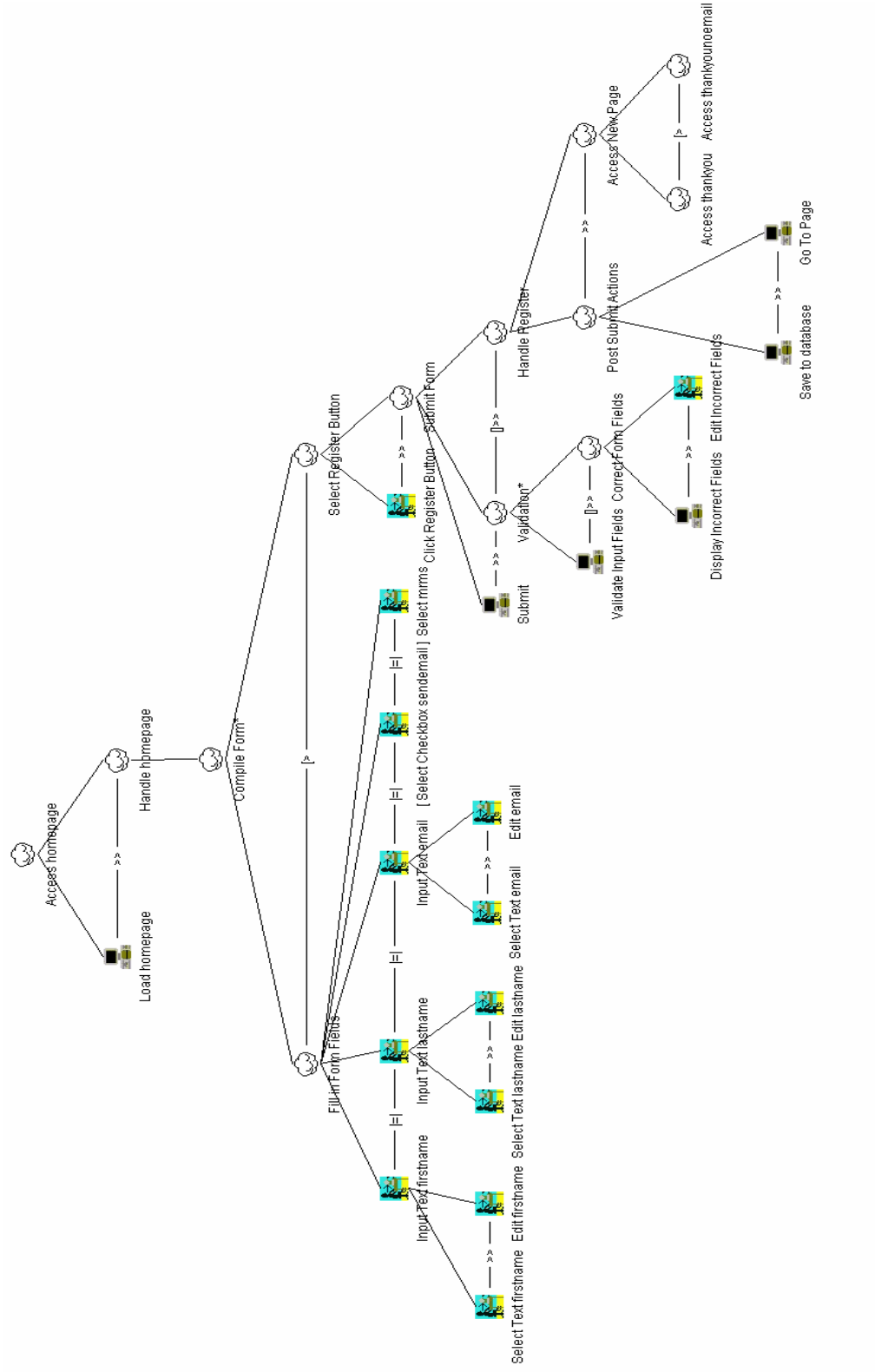


Figure 41: Task model for homepage of the "Registration Application"

A.7 Generic UIML Representation for Home Page

```
<?xml version="1.0"  encoding="ISO-8859-1" ?>
<uiml>
 <peers>
  <logic/>
  <presentation id="base" base="Generic_1.2_Harmonia_1.0"/>
 </peers>
 <interface>
  <structure>
   <part id="mainwindow" class="G:TopContainer">
    <part id="form" class="G:Area">
     <part id="titlelabel" class="G:Text"/>
     <part id="mrms" class="G:ButtonGroup">
      <part id="mrms_mr" class="G:Button"/>
      <part id="mrms_ms" class="G:Button"/>
     </part>
     <part id="firstnamelabel" class="G:Text"/>
     <part id="lastnamelabel" class="G:Text"/>
     <part id="emaillabel" class="G:Text"/>
     <part id="firstname" class="G:TextField"/>
     <part id="lastname" class="G:TextField"/>
     <part id="email" class="G:TextField"/>
     <part id="sendemail" class="G:Button"/>
     <part id="registerbutton" class="G:Button"/>
    </part>
   </part>
  </structure>
  <style>
   <property name="g:layout" part-name="mainwindow">null</property>
   <property name="g:location" part-name="mainwindow">108,36</property>
   <property name="g:size" part-name="mainwindow">601,505</property>
   <property name="g:title" part-name="mainwindow">Registration
   Application</property>
   <property name="g:layout" part-name="form">null</property>
   <property name="g:location" part-name="form">4,3</property>
   <property name="g:size" part-name="form">587,464</property>
   <property name="g:text" part-name="titlelabel">Welcome! Please
   register below.</property>
   <property name="g:location" part-name="titlelabel">78,83</property>
   <property name="g:size" part-name="titlelabel">411,27</property>
   <property name="g:location" part-name="mrms">4,3</property>
   <property name="g:size" part-name="mrms">73,115</property>
   <property name="g:buttontype" part-name="mrms_mr">radio</property>
   <property name="g:text" part-name="mrms_mr">Mr.</property>
   <property name="g:location" part-name="mrms_mr">73,115</property>
   <property name="g:size" part-name="mrms_mr">45,22</property>
   <property name="g:buttontype" part-name="mrms_ms">radio</property>
   <property name="g:text" part-name="mrms_ms">Ms.</property>
   <property name="g:location" part-name="mrms_ms">112,115</property>
   <property name="g:size" part-name="mrms_ms">53,22</property>
   <property name="g:text" part-name="firstnamelabel">First Name:
   </property>
   <property name="g:location" part-name="firstnamelabel">78,135
   </property>
   <property name="g:size" part-name="firstnamelabel">66,22</property>
```

```
 <property name="g:text" part-name="lastnamelabel">Last Name:
 </property>
 <property name="g:location" part-name="lastnamelabel">78,185
 </property>
 <property name="g:size" part-name="lastnamelabel">82,20</property>
 <property name="g:text" part-name="emaillabel">E-Mail:</property>
 <property name="g:location" part-name="emaillabel">78,240</property>
 <property name="g:size" part-name="emaillabel">82,18</property>
 <property name="g:location" part-name="firstname">78,154</property>
 <property name="g:size" part-name="firstname">108,20</property>
 <property name="g:location" part-name="lastname">78,205</property>
 <property name="g:size" part-name="lastname">108,20</property>
 <property name="g:location" part-name="email">78,257</property>
 <property name="g:size" part-name="email">108,20</property>
 <property name="g:buttontype" part-name="sendemail">
 checkbox</property>
 <property name="g:text" part-name="sendemail">Send me a confirmation
 email</property>
 <property name="g:location" part-name="sendemail">78,315</property>
 <property name="g:size" part-name="sendemail">219,18</property>
 <property name="g:text" part-name="registerbutton">Register
 </property>
 <property name="g:location" part-name="registerbutton">80,355
 </property>
 <property name="g:size" part-name="registerbutton">108,23</property>
</style>
<behavior>
<rule>
  <condition>
   <event part-name="registerbutton" class="g:actionperformed"/>
  </condition>
  <action>
   <call name="validate.isWithinRange">
    <param name="fieldToValidate">
      <property part-name="firstname" name=" g:text"/>
    </param>
    <param name="minValue">1</param>
    <param name="maxValue">30</param>
    <param name="errorMsg">Please enter between 1 and 30 characters.
    </param>
   </call>
   <call name="validate.isWithinRange">
    <param name="fieldToValidate">
      <property part-name="lastname" name=" g:text"/>
    </param>
    <param name="minValue">1</param>
    <param name="maxValue">50</param>
    <param name="errorMsg">Please enter between 1 and 50 characters.
    </param>
   </call>
   <call name="validate.isValidEmailAddress">
    <param name="fieldToValidate">
      <property part-name="email" name=" g:text"/>
    </param>
    <param name="errorMsg">Please enter a valid email address.
    </param>
   </call>
```

```
    </action>
   </rule>
   <rule>
    <condition>
      <op name="and">
       <event part-name="registerbutton" class="g:actionperformed"/>
       <op name="==">
        <property part-name="sendemail" name="g:selected"/>
        <constant value="true"/>
       </op>
      </op>
     </condition>
     <action>
      <when-true>
       <call name="action.loadNewPage">
        <param name="pageName">thankyou</param>
       </call>
      </when-true>
      <when-false>
       <call name="action.loadNewPage">
        <param name="pageName">thankyounoemail</param>
       </call>
      </when-false>
     </action>
    </rule>
  </behavior>
 </interface>
</uiml>
```

A.8 Generic UIML Rendered In Java

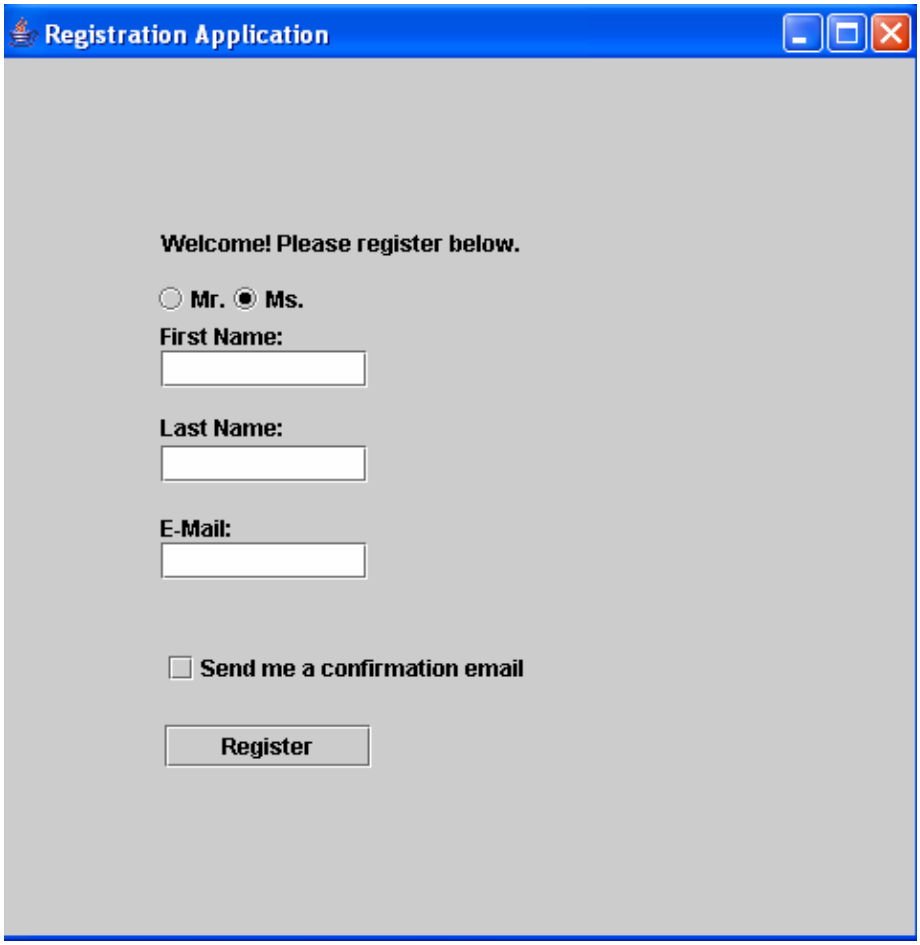


Figure 42: Generic UIML for the homepage of Registration Application rendered in Java by UIML browser

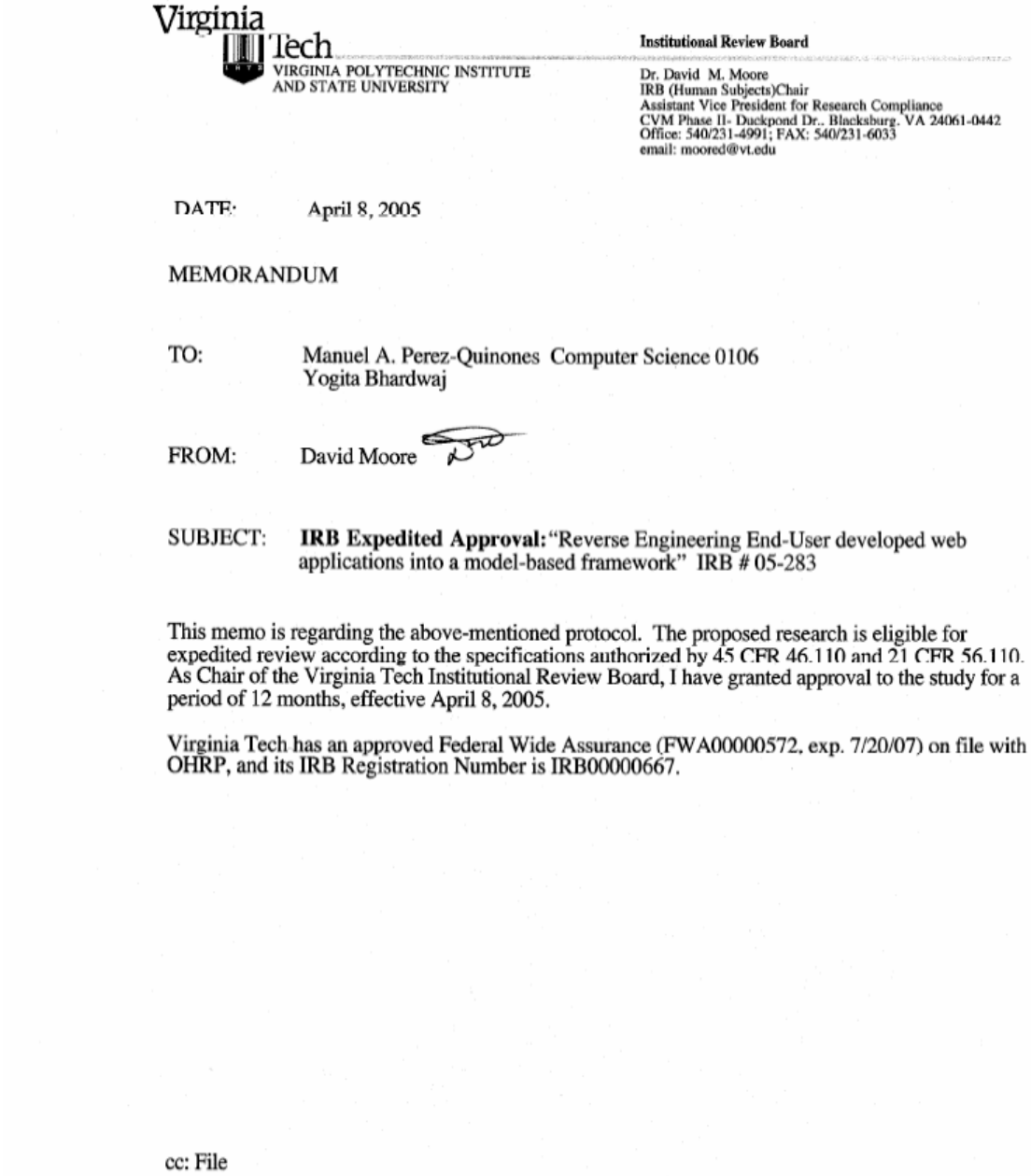
A.9 Java Module to render a generic G:Table as Java Swing JTable

```
/**
 * A program to invoke UIML renderer on file
 * GenericDataTableExample.uiml.
 * This is an automatically generated file which needs customized
 * according to any particular requirements.
*/

import com.harmonia.renderer.Renderer;
import javax.swing.JTable;
import javax.swing.table.*;
import java.sql.*;
```

```
public class GenericDataTableExample {
    static String uimlFileName = "GenericDataTableExample.uiml";
    static Object[][] tableData;

    static void fetchDataFromDB() {
        Connection connection = null;
        // CLICK: Please create the database connection here
        Statement select = con.createStatement();
        ResultSet result = select.executeQuery ("SELECT destination,
        departuredate FROM data");
        int i = 0;
        while(result.next()) { // process results one row at a time
            String destination = result.getString(1);
            String departuredate = result.getString(2);
            tableData[i][0] = destination;
            tableData[i][1] = departureDate;
            i++;
          }
        select.close();
        connection.close();
    }

    // Create a TableModel for G:Table using data from the database
    //
    static TableModel dataModel = new AbstractTableModel() {
        public int getColumnCount() { return tableData[0].length; }
        public int getRowCount() { return tableData.length;}
        public Object getValueAt(int row, int col) {
            if (row>=getRowCount() || col>=getColumnCount()) return
            null;
            return tableData[row][col];
        }
    };

    public static void main(String[] args) {
        Renderer r = null;
        r = new Renderer();
        boolean ok = r.renderUIML(
                          uimlFileName,
                          null,  //interfaceName
                          null,  //structureName
                          null,  //contentName
                          null,  //styleName
                          null,  //behaviorName
                          null,  //peerName
                          false, //-Dtime     option
                          false, //-Dnorender option
                          false, //-Delements option
                          false, //-Dverbose option
                          false, //-Dtrace    option
                          false  //-validate  option
                         );

        if (ok) {
            fetchDataFromDB();
            JTable table = (JTable) r.getPartByName("SampleTable");
            if (table!=null)
```

```
                table.setModel(dataModel);
            else
                System.err.println(uimlFileName +
                                   " did not contain a part named
                                   'SampleTable'.");
            table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            ListSelectionModel rowSM = table.getSelectionModel();
            rowSM.addListSelectionListener(new ListSelectionListener() {
                public void valueChanged(ListSelectionEvent e) {
                    if (e.getValueIsAdjusting()) return;
                    ListSelectionModel lsm =
                    (ListSelectionModel)e.getSource();
                    if (!lsm.isSelectionEmpty()) {
                        //CLICK: Add code to display the details of the
                        //selected record in a window.
                    }
            });
        }
    }
}
```

Appendix B: IRB Approval for Evaluation



Virginia Tech
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

Institutional Review Board

Dr. David M. Moore
IRB (Human Subjects)Chair
Assistant Vice President for Research Compliance
CVM Phase II- Duckpond Dr., Blacksburg, VA 24061-0442
Office: 540/231-4991; FAX: 540/231-6033
email: moored@vt.edu

DATE: April 8, 2005

MEMORANDUM

TO: Manuel A. Perez-Quinones Computer Science 0106
Yogita Bhardwaj

FROM: David Moore

SUBJECT: **IRB Expedited Approval:** "Reverse Engineering End-User developed web applications into a model-based framework" IRB # 05-283

This memo is regarding the above-mentioned protocol. The proposed research is eligible for expedited review according to the specifications authorized by 45 CFR 46.110 and 21 CFR 56.110. As Chair of the Virginia Tech Institutional Review Board, I have granted approval to the study for a period of 12 months, effective April 8, 2005.

Virginia Tech has an approved Federal Wide Assurance (FWA00000572, exp. 7/20/07) on file with OHRP, and its IRB Registration Number is IRB00000667.

cc: File

Figure 43: Scanned IRB Approval for Formative Evaluation

Appendix C: Evaluation Task Sheets and Questionnaire

C.1 Evaluation Pre-Questionnaire

Name:

Level of Education:

Academic Major:

No. of years of experience with web development

Web Technologies/Languages known:

Rate your expertise in following language (1-lowest, 10-highest)

HTML:

Java:

PHP:

ASP:

Javascript:

Other (Please Specify)

C.2 Tasks Sheet

Task 1

Write a brief summary of what the application does? Explain in terms of flow information and control of the application in as much detail as you can understand.

Task 2

What actions does the "Register" button perform? Briefly mention what artifact you are using to answer the question.

Task 3

What data is stored in the database, if any, and when?

Following tasks ask you to make modifications to the application. Please explain, in as much detail as you can, what changes will you make and indicate on the document particular area where you will make the changes. You may write the code if you want.

Task 4

What input validations are performed on "Last Name"? Please change this to "Last Name should not have any spaces"?

Task 5

Change the text for the page "homepage" from English to another language your choice. If you do not know any other language, you can simply point out what changes would you do.

Task 6

How would you add the following functionality to the application?

Have a text box on the home page that asks the users to enter their department's mail code. For reference a "Lookup" button is added and clicking this button will popup a window that contains mail codes for all the departments. The mail code entered by the user will also need to be saved to the database.

C.3 Satisfaction Questionnaire

Please rate the models and documentation in terms of how easy were they to understand? Why? (1 = very difficult, 5 = very easy)

Sitemap

Documentation

Task Models

UIML Representations

Please rate the models and documentation in their usefulness in understanding the application? You can write any comments if you have. (1 = not at all useful, 5 = very useful)

Sitemap

Documentation

Task Models

UIML Representations

Please rate the models and documentation in their usefulness in modifying the application? You can write any comments if you have. (1 = not at all useful, 5 = very useful)

Sitemap

Documentation

Task Models

UIML Representations

What information do you think was missing from the documents?

What modifications would you suggest, if any?

How you think the having separate sections in a UIML document, structure, content, style, behavior, was helpful in comparison the traditional programming, like in PHP or Java.

- having a separate section for all the text in the document
- having layout and behavior separate
- Do you think using UIML you will be able prototype rapidly than PHP and HTML for any enhancements that you may have to do to the web application? Why?

Appendix D: Evaluation Tutorials

D.1 Task Model Notation (CTT)

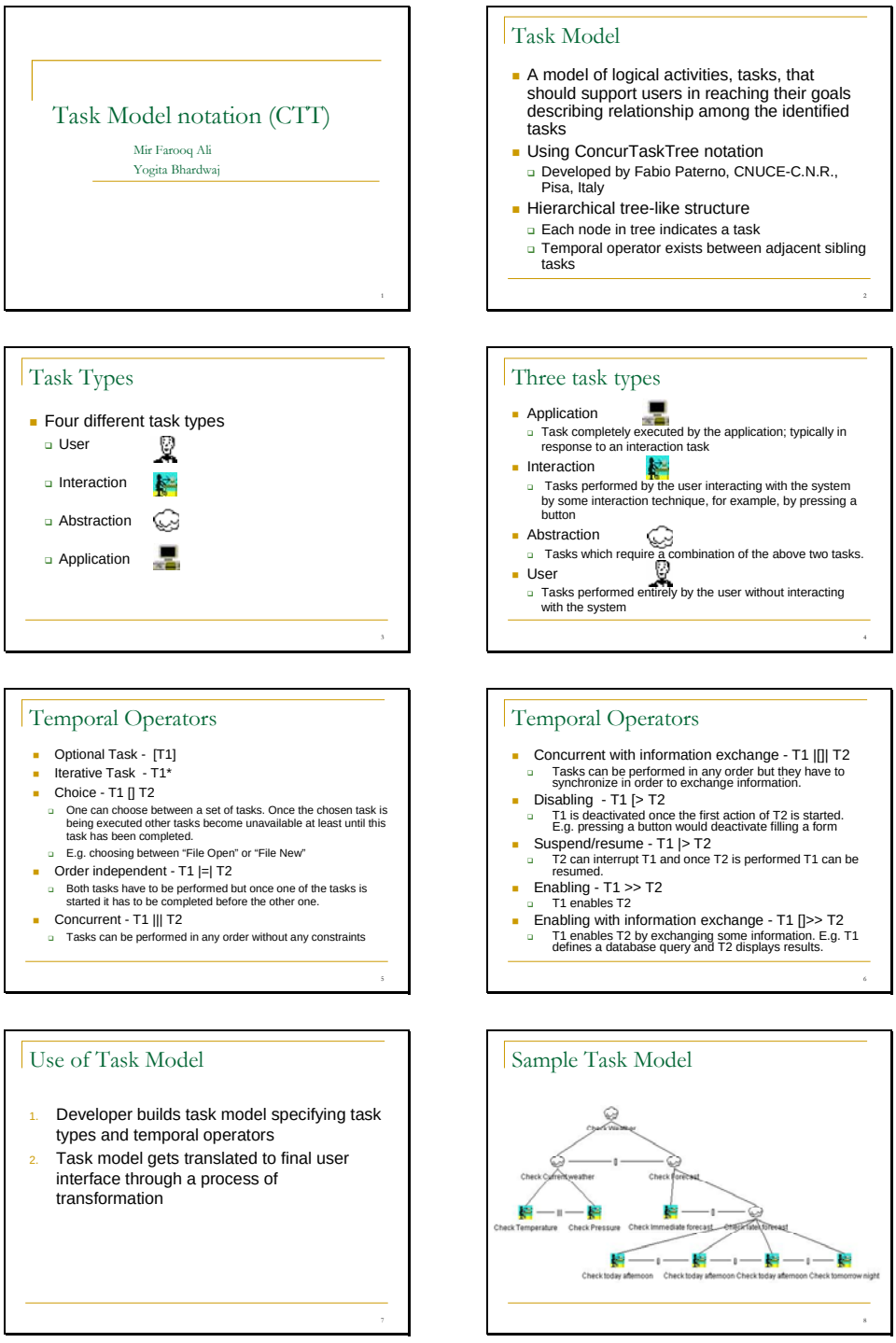


Figure 44: Slides for the tutorial on ConcurTaskTrees used in evaluation

D.2 Building User Interfaces using UIML

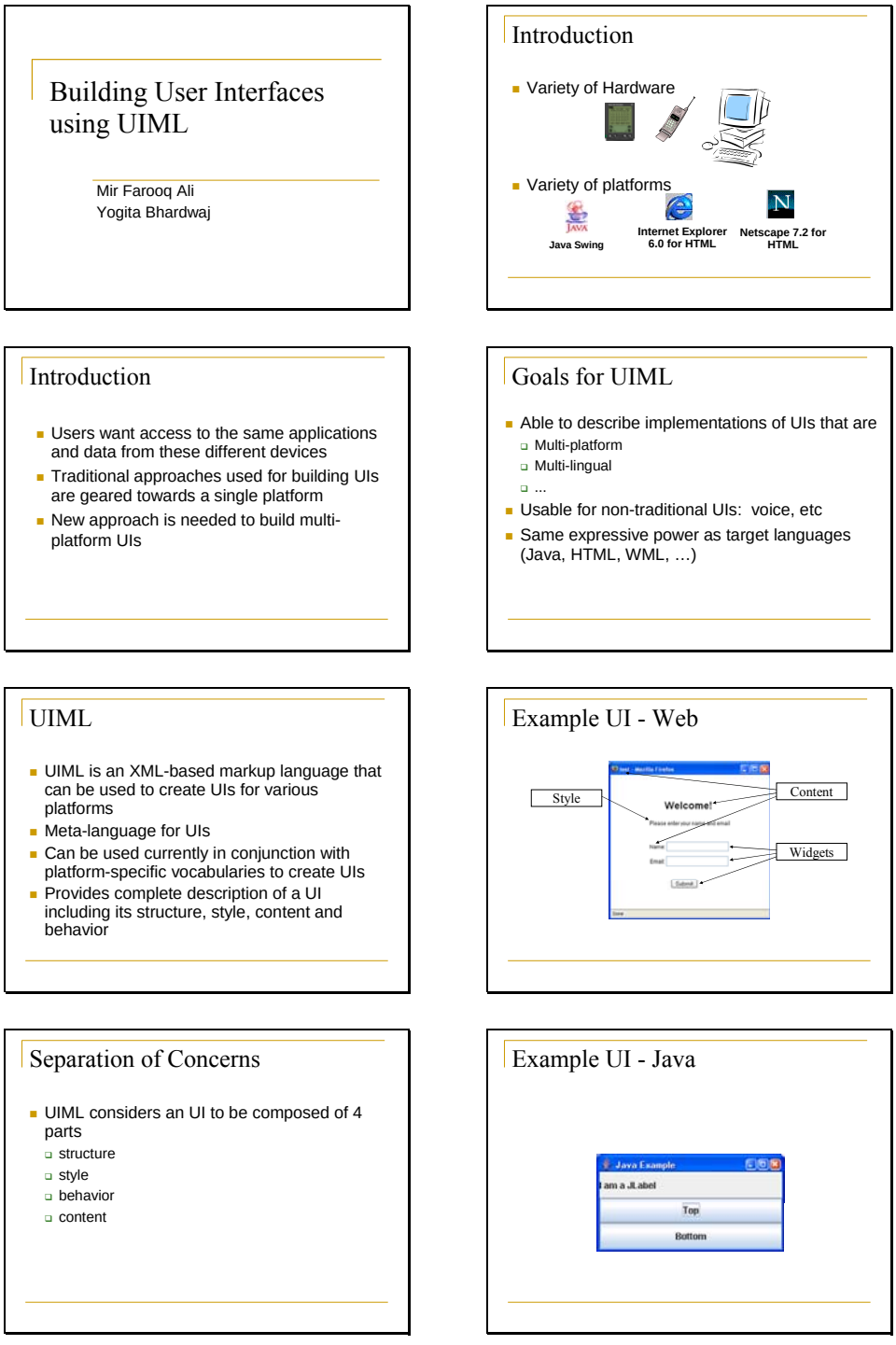


Figure 45: Slides 1-8 for the tutorial on UIML used in evaluation

Java code - widgets

```
public class SecondAppl extends JFrame {
public SecondAppl(String lab) {
    ...
    JPanel jp = new JPanel();
    add(jp);
    final JLabel label1 = new JLabel("I am a
  JLabel");
    jp.add(label1);
    JButton topButton = new JButton("Top");
    jp.add(topButton);
    JButton bottomButton = new JButton("Bottom");
    jp.add(bottomButton);
    ...
}
```

Java code – behavior (Events)

```
topButton.addActionListener(
   new ActionListener() {
     public void actionPerformed(ActionEvent e) {
     label1.setText("Top Button Pressed");
   }
});

bottomButton.addActionListener(
   new ActionListener() {
     public void actionPerformed(ActionEvent e) {
     label1.setText("Bottom Button Pressed");
   }
});
```

UIML

- Skeleton of UIML document

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0
Draft//EN" "UIML2_0g.dtd">
<uiml>
  <head>...</head>
  <interface>
      <structure>...</structure>
      <style>...</style>
      <behavior>...</behavior>
      <content>...</content>
  </interface>
  <peers>...</peers>
  <template>...</template>
</uiml>
```

UIML <structure>

- Structure section indicates the hierarchical relationship between various 'parts'

```
<structure>
  <part id="BigFrame" class="JFrame">
    <part id="panel" class="JPanel">
      <part id="label1" class="JLabel"/>
      <part id="topButton" class="JButton"/>
      <part id="bottomButton" class="JButton"/>
    </part>
  </part>
</structure>
```

Widgets

UIML <style>

- Style section describes "look and feel" of UI using properties for individual parts

```
<style>
  <property part-name="BigFrame" name="title">UIML
  Example</property>
  <property part-name="BigFrame"
  name="bgcolor">blue</property>
  <property part-name="label1" name="text">
      <reference constant-name="MyLabelText"/>
  </property>
</style>
```

Reference to Content

UIML <content>

- This section describes text, images or sounds used in the application
- Text property in the style section can be specified as a constant value or as a reference to a constant in the <content> section

E.g.

```
<content id = "English">
 <constant id="MyLabelText">I am a JLabel</constant>
</content>
```

Language Specific

UIML <behavior>

- Behavior section defines what happens when you perform any action on the parts

```
<behavior>
 <rule>
  <condition>
   <event class="actionperformed"
          part-name="topButton"/>
  </condition>
  <action>
   <property part-name="label1" name="text">Top button
   pressed.</property>
  </action>
 </rule>
</behavior>
```

UIML <behavior> contd.

- More Conditions and Actions

```
<rule>
  <condition>
   <op name="and">
     <event part-name="buttonid" class="actionperformed"/>
     <op name="equal">
       <property name="value" part-name="city"/>
       <constant value="Los Angeles"/>
     </op>
   </op>
  </condition>
  <action>
    <call name="action.function">
      <param name="param1">paramvalue</param>
    </call>
  </action>
</rule>
```

maps to Class action::function(int param1)

Figure 46: Slides 9-16 for the tutorial on UIML used in evaluation

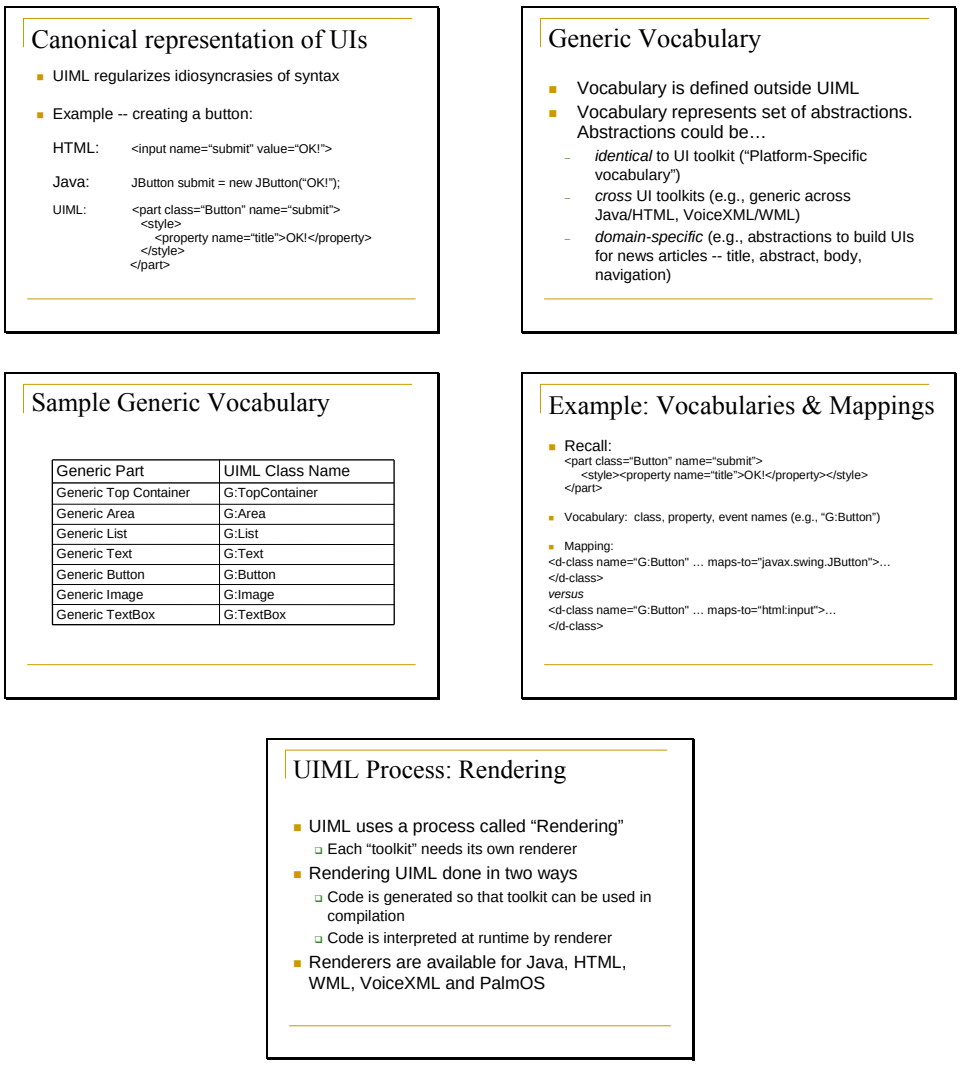


Figure 47: Slides 17-21 for the tutorial on UIML used in evaluation

Appendix E: List of references to model-based tools

Table 9: List of Model-based tools and methodologies and references to these tools

Tools and Methodologies	Year	References
UIDE – User Interface Design Environment	1991	[8], [9]
HUMANOID - High-level UIMS for Manufacturing Applications Needing Organized Iterative Development *or it can also stand for* Human Understanding and Machine Automation are Needed for Optimizing Interface Development	1993	[10], [11], [12]
ADEPT – Design Environment for Prototyping Tasks	1993	[14], [15], [16], [17]
TRIDENT - Tools foR an Interactive Development EnvironmeNT	1993	[18], [19]
Mecano	1995	[21]
MASTERMIND - Models Allowing Shared Tools and Explicit Representations Making Interfaces Natural to Develop	1996	[13]
Mobi-D – Model based interface designer	1997	[22], [25], [26]
CTT - ConcurTaskTrees	1997	[39]
UIML – User Interface Markup Language	1999	[31], [32], [33], [34]
LiquidUI™	1999	[66]
Plasticity Framework	2001	[23], [24]
XIML – eXtensible Interface Markup Language	2001	[36]
VAQUITA - reVerse engineering of Applications through QUestions, Information selection, and Transformation Alternatives	2001	[51], [52], [53], [54]
CTTE – ConcurTaskTrees Environment	2001	[40]
WebRevenge – Web Reverse Engineering	2003	[55], [56]
USIXML- USer Interface eXtensible Markup Language	2004	[28], [29], [30]
TERESA - Transformation Environment for inteRactivE Systems representAtions	2004	[43], [44], [45]

Vita

Yogita Bhardwaj

Education

Master of Science, Computer Science, at Virginia Tech, July 2005. (GPA-3.7/4.0)

Bachelor of Information Science, Kalindi College, University of Delhi, 2001 (GPA - 3.6/4.0)

Research

Graduate Research Assistant, Department of Computer Science, Virginia Tech, Jan 2004– May 2004, Aug 2004 - May 2005

- Part of the development team for Click (Component-based Lightweight Internet-application Construction Kit)

Work Experience

Software Design Engineer/Test Intern, Windows Client Platform, Microsoft, May 2004 – Jul 2004

- Part of the Media Integration Layer, Avalon SDK team.

Member of Technical Staff , Cadence Design Systems (I) Pvt Ltd, NOIDA, India, Jan 2001 – Jul 2003

- Part of the Printed Circuit Board and Analog Simulation Team.

Intern, HCL Infosystems Limited, NOIDA, India , Aug 1999 – Oct 1999

- Company Intranet web master

Awards

First Prize in Inter College Web Portal Development Contest, organized by Department of Computer Science, University of Delhi.

Merit-cum-means scholarship for undergraduate studies from 1997-2001 by IndusInd Foundation.

Shri M.P. Mathrani Memorial Award for the year 1997

K.C. Mahindra Educational Trust Scholarship for the year 1994 and 1996

Certificate of Merit in Math Olympiad 1995.