RC23168 (W0403-169) March 30, 2004
Computer Science

IBM Research Report

An Authoring Technology for Multi-Device Web Applications

G. Banavar[1], L. Bergman[1], R. Cardone[1], V. Chevalier[1], Y. Gaeremynck[1], F. Giraud[1], S. Hirose[2], M. Hori[2], F. Kitayama[2], G. Kondoh[2], A. Kundu[3], K. Ono[2], A. Schade[4], D. Soroker[1], K. Winz[5]

[1]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[2]IBM
Shimotsuruma 1623-14
Yamato City
Kanagawa 242-8502
Japan

[3]IBM India Research Laboratory
Block-I, IIT
Hauz Khas
New Delhi, India 110016

[4] IBM Research Division
Zurich Research Laboratory
8803 Rueschlikon, Switzerland

[5]IBM
P.O. Box 12195
3039 Cornwallis Road
Research Triangle Park, NC 27709

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

An Authoring Technology for Multi-Device Web Applications

G. Banavar, L.. Bergman, R. Cardone, V. Chevalier, Y. Gaeremynck, F. Giraud, S. Hirose, M. Hori, F. Kitayama, G. Kondoh, A. Kundu, K. Ono, A. Schade, D. Soroker, K. Winz
IBM Worldwide Research and Development Labs
Contact: Guruduth Banavar, T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532
Email: banavar@us.ibm.com, Tel: +1-914-784-7755

Introduction

Pervasive computing enables information and applications to be accessed anytime, anywhere, using any device. For a user to have a satisfying experience accessing a pervasive application *using any device*, the application developer must author the application to execute on multiple devices. In the past few years, there has been a proliferation of computing devices with a wide range of capabilities; these devices include PDAs, internet-capable phones, and consumer appliances. Creating and managing applications for this diversity of devices has become a significant challenge. For example, building an interactive web application that can be accessed by multiple browser-based devices usually means writing separate applications for each device. This is problematic, since the application developer must reflect any modification in several versions of the application.

The need for application development support is particularly evident in mobile devices. These devices are being released at an accelerating rate, with any particular model having an increasingly short lifecycle. In addition, both hardware and software manufacturer's time-to-market is becoming shorter and more critical. For these reasons, development and maintenance of multi-device applications is growing prohibitively complex and costly.

Multi-device applications are also essential for enabling future types of pervasive applications [Banavar 2000]. When pervasive applications are enabled for migration between a multiplicity of heterogeneous devices, application authoring becomes far more complex. In addition to being concerned with presentation on each device, the application author must also address migration issues including user interface consistency and differing device capabilities. A technology for authoring multi-device applications is essential to manage this complexity.

One solution to this problem might be to author a multi-device application just once, and to build an intelligent infrastructure that can automatically customize that application to the context of use, based on meta-data describing the characteristics of the context of use. However, in reality, the more specialized the characteristics of a device, the less likely it is for an automatically generated application to be usable on that device. A good solution to this problem must allow for both automatic adaptation as well as developer customization of multi-device applications.

Multi-Device Authoring Technology, or MDAT, has been developed to reduce the complexity of creating interactive, form-based, web[1] applications and targeting them to multiple heterogeneous devices. The approach is for the application developer to specify the common aspects of an application within a device-independent *generic* application, and to use the MDAT tools to create customized device-specific versions of that application. MDAT enhances reuse by enabling the generic part of an application to be shared across multiple devices, while allowing device-specific extensions when necessary. MDAT also automates repetitive tasks such as writing the same application in multiple markup languages. A distinguishing characteristic of MDAT is the combination of automated generation of device-specific applications based on device characteristics and a rich set of tools for incremental manual customization of the application for specific target platforms.

In support of the above approach, the MDAT tools provide several capabilities:

1. Visual specification of the generic application flow and user interaction common to multiple devices.
2. Visual specification of incremental changes to the generic flow and user interaction for target devices, either individually and grouped by category.

[1] Web portals, consisting of a collection of portlet applications, are emerging as a new paradigm for composing and organizing groups of applications suited for particular tasks on the web. The phrase "web application" in this article refers to both conventional web applications as well as portlet applications.

3. Access to a device profile repository containing the characteristics of a large number of popular devices, including PDAs, handhelds, phones, and desktop browsers.
4. Design-time and runtime generation of device-specific web and portlet applications artifacts (Java Server Pages, or JSP, and Struts [Struts] code) for a variety of devices supporting multiple markup languages.
5. Hand-customization of generated device-specific pages and storage of the customizations so that they can be re-applied upon re-generation.
6. Conversion of existing HTML pages into generic, retargetable application fragments.

An Eclipse-based [Eclipse] toolset supporting most of the above features has been developed and has been released for public evaluation in February 2003 [Everyplace]. The design of MDAT is based on lessons learned from first-generation research prototypes.

This paper describes the application developer's perspective on the MDAT toolset. The primary contribution of this paper is an end-to-end development methodology and toolset for building interactive form-based web applications for multiple heterogeneous devices. The novel aspects of this methodology are (1) *Specialization* – the ability for the developer to incrementally modify the generic application for specific targets, (2) *Iterative Refinement* – the ability for the developer to iteratively edit generated artifacts without losing the changes, and (3) *Generalization* – the ability to convert existing HTML pages into retargetable applications. We will illustrate how this development methodology makes it easy and cost effective to build and manage multi-device applications.

The next section gives an overview of the MDAT development approach. The sections following that describe the major aspects of the development process in the following order: device profiles, application specification and specialization, device-specific application generation, iterative refinement, and generalization.

Multi-Device Application Development Approach

Application Model

MDAT applications are called *multi-device applications.* Multi-device applications capture the structure and function of interactive form-based web applications, such as web-based banking, travel booking, registration, and shopping cart applications. A single multi-device application is associated with multiple *targets*, each of which is either an individual device or a category of devices. A multi-device application consists of a *generic application* and one or more *specialized applications*. The generic application describes those aspects of the MDAT application that are common to all targets. Each specialized application describes incremental deviations from the generic application for a particular target.

Multi-device applications are based on the Model-View-Controller pattern, also known as the JavaServer Pages "Model 2" architecture [JSP MVC]. The Model components, which encapsulate business logic and data, are common to all target devices and are defined as JavaBean components [JavaBeans].

The View components describe the presentation and user interaction within a multi-device application. A collection of presentation and interaction elements that logically belong together is known as a *dialog*. Generic dialogs, i.e., dialogs within the generic application, contain *content elements* (e.g., text and images), *form elements* (e.g., type-in fields or selection lists), *containers* for specifying dynamic iteration and grouping, and references to *data model elements* defined as JavaBean components. The elements within dialogs are represented using XHTML Basic with embedded XForms [XForms]. Specialized dialogs, i.e., dialogs within a specialized application, can also include device-specific information, such as user-specified page breaks.

The Controller component describes the control flow of an application and is represented as a flow graph. The nodes of the graph represent the dialogs described above, and the arcs, called *transitions* in MDAT, represent the navigation among dialogs. A transition can be conditional, that is, a request from a dialog might result in navigation to different dialogs depending on the result of running a JavaBean method at runtime, which processes the information in the request. Transitions may also be associated with an action, which is a JavaBean method that is invoked when the transition is taken. The generic controller, i.e., the controller within the generic application, represents control flow that is common to all targets. Specialized controllers, i.e., controllers within specialized

applications, can customize the flow for particular targets by deleting generic dialogs and transitions or by adding device-specific markup pages and transitions.

Hybrid Design-Time and Runtime Adaptation

A fundamental aspect of the MDAT approach is its support for both *design-time* and *runtime* adaptation. Design-time adaptation is the process of converting a multi-device application into multiple device-specific versions before the application is deployed to the server. Design-time adaptation includes both manual and automatic activities, as described in the next section. Design-time adaptation has several benefits:

1. The developer can view, edit, and fine-tune the generated artifacts if necessary.
2. The system does not incur any runtime overhead for adaptation.
3. The generated applications can be deployed to industry-standard servers that do not have special support for multi-device applications.

Although design-time adaptation provides significant benefits, runtime adaptation is also essential for a complete solution. Runtime adaptation is the process of converting the generic portion of a multi-device application to a device-specific version when the application is requested from the server by a client device. Runtime adaptation has the following benefits:

1. Devices that are not known at design-time, for example, new devices that are introduced in the marketplace, can be supported.
2. Adaptation based on dynamic data can only be handled at runtime.
3. The size of the deployed application archive can be significantly reduced.

The MDAT system derives all the above benefits by supporting both design-time and runtime adaptation. Design-time adaptation results in one or more device-specific application versions that can be deployed to a Web application server. When a device makes a request for the application, the runtime web application dispatcher determines if an application request can be satisfied by an existing device-specific application version. If not, a device-specific version of the application is generated on the fly and delivered to the device. Thus, runtime adaptation allows MDAT to service requests from devices that do not have a pre-defined device-specific application version.

In the rest of this paper, we focus on design-time adaptation support, since the purpose of this paper is to describe the development process and tools.

Semi-Automatic Design-Time Adaptation

In this section, we describe how MDAT developers create web applications and how these applications are adapted to run on different devices using a *semi-automatic, design-time* adaptation process. The process is semi-automatic because most aspects of the process are automated, but developers have control at key points for customizing how the application is adapted for each target.

Figure 1 identifies the three main steps involved in creating multi-device applications in MDAT. Solid arrows indicate processing steps that are automated by MDAT; dashed lines indicate steps that use manual input. We now describe the automated and manual components of each step.

In Step 1, the developer defines a generic application, which represents the common aspects of the View and Controller portions of an application across multiple target devices. The MDAT *Generic Controller Editor* is used to define the control flow among the pages in the application. The MDAT *Generic View Editor* is used to define generic dialogs, which are the logical pages of an application. Dialogs are logical in the sense that they may get realized as multiple physical web pages for particular target devices. The Controller and View editors are described in later sections.

In addition to authoring new dialogs from scratch, developers can convert existing HTML files into dialogs through the *generalization* process. The MDAT *Generalization Editor* allows developers to interactively translate standard HTML markup into MDAT's dialog representation. The dialogs created through generalization can be manipulated

using the controller and view editors just like manually created dialogs. The Generalization Editor is described in the Generalization Section below.

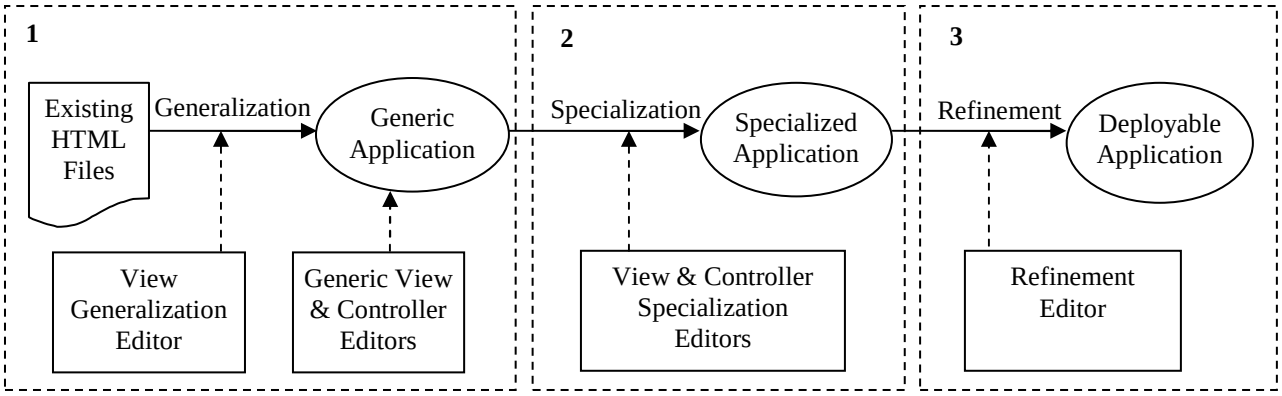

Figure 1 – Design-Time Application Generation

In Step 2, MDAT automatically translates a generic application into one or more specialized applications, which are intermediate non-deployable versions of device-specific applications. The devices targeted during translation are specified using the controller editor. For each target device, *specialized JSP* files are generated from the dialog files in the generic application. These specialized JSPs support the markup languages required by their particular devices, such as HTML or WML. The *specialization* process automatically modifies both the View and Controller portions of an application to accommodate specific devices. For example, a dialog may be split into multiple specialized JSPs for devices with small screens. In this case, MDAT will also generate the appropriate control flow for the specialized JSPs.[2]

Automated specialization, however, cannot meet the needs of all applications, as argued earlier. If the semantics of an application influence how it should be specialized, then a completely automated approach is not practical. For example, portions of an application may be removed or added on particular devices due to security or usability concerns. This type of customization requires the developer to be able to incrementally add or remove from the generic application for particular target devices. MDAT allows developers to specify incremental customizations in both the controller and view portions of application for particular devices. This process, which we refer to as *manual* specialization, is one of the key innovations in the MDAT development methodology. Developers use the *Controller and View Specialization Editors* to manually specialize application flow and content for specific devices.

In Step 3, MDAT translates specialized applications into standard web applications, which can then be deployed. Once a specialized application has been generated, developers can iteratively modify the formatting and layout of its presentation pages using the *Refinement Editor*. These post-specialization *refinements* specify customizations of an application's look and feel on specific devices. The refinement editor remembers all refinements performed by the user, which allows the system to help the developer reapply those refinements whenever the generic application is modified and the specialized application is regenerated. The Refinement Editor is described in the Iterative Refinement Section.

Overview of MDAT Tool Components

MDAT provides a comprehensive set of tools to facilitate a flexible and iterative development process. The following are the main components:

- **Controller Editor:** This visual builder helps the developer define the application flow consisting of dialogs and transitions. The targets (individual devices or categories of devices) for an application can be defined. The application flow can be manually specialized for each target. JavaBean business logic can be associated with the flow.

[2] This feature, known as *automatic page splitting*, can be turned off by users.

- **View Editor:** This visual builder helps the developer define the form and content elements within each dialog. The user interaction can be manually specialized for particular targets. Data sources for dynamic content, as well as submission of form data to the server, can be specified.
- **Refinement Editor:** This editor is used to fine-tune the generated JSPs for a particular device. These refinements are stored and can later be re-applied in case the base generic dialog is changed and the JSPs regenerated.
- **Generalization Editor:** This editor helps the developer interactively convert a static HTML page into a dialog that can be retargeted to multiple devices. This provides the flexibility for designers to use external tools for web page design and then import the pages into MDAT.
- **Device Profile Tools:** These tools help the developer define and view device profiles and device profile categories.
- **Deployment and Test Environment:** The MDAT tools are integrated into underlying IBM WebSphere Studio tools, built on the Eclipse IDE framework. The user can iteratively design, build, refine, deploy, and test an application, without throwing work away – for any or all targeted devices.

Device Profiles

MDAT's support for device profiles is central to both the automated and the manual process of application specialization. When specializing applications, MDAT users have access to a database of predefined *device profiles* that describe the detailed capabilities of individual devices. The "Targets" area within the MDAT Controller Editor shows the list of supported devices organized by manufacturer name. Developers can view device characteristics, edit existing profiles, add new device profiles, or select devices for specialization from this page.

Device profiles contain device properties that are organized according to the six components defined by the OMA UAProf specification [UAProf]: hardware platform, software platform, browser UserAgent, WAP characteristics, network characteristics and push characteristics. Device profiles for phones, PDAs and desktops are stored in a database of predefined profiles. Most device profiles in the database correspond to actual phones or PDAs. We have also defined profiles for desktop clients, such as Microsoft Internet Explorer and Netscape Navigator, and for a number of device emulators.

Developers can also define *profile categories* to help manage large numbers of devices. A category is a user-defined predicate over the properties contained in device profiles. When a category's predicate is applied to the profile database, the result is a set of device profiles that satisfy the category's predicate. We say that the category contains the devices represented by this set of profiles. For example, developers may choose to define a category that contains all devices that support WML markup, or a category that contains all devices with screen size less than VGA. Profile categories can also be created by refining existing categories, thus resulting in a hierarchy of categories. MDAT allows profile categories to be used as specialization targets.

Application Specification and Specialization

As described in the section on the application model, MDAT supports the easy development of the View and the Controller portions of multi-device web applications. MDAT developers typically begin by creating generic applications by specifying the control flow and the dialogs that apply to all devices. Developers then use various editors to define device-specific customizations when they are needed. In this section, we describe how to specify and specialize applications using the Controller and View Editors.

Controller Editor

As described earlier, the generic controller (i.e., the controller within the generic application) is the common basis for all devices. The specialized controller (i.e., the controller within a specialized application) is often identical or very similar to the generic flow. We now describe how the Controller Editor is used to create both the generic and specialized controllers, and to specify the associations with the View and the Model.

The Controller Editor provides a visual builder for the generic flow graph. Generic dialogs and transitions can be created, deleted, and moved around on a canvas, and their attributes can be modified through a property sheet editor. Figure 2 shows a simple banking application within the controller editor. The left screenshot shows the generic

controller flow, which starts with a Login dialog and proceeds to the BrowseAccounts dialog if login is successful. The transition from Login to BrowseAccounts is conditional upon successful authentication of login information. Similarly, there are transitions from the BrowseAccounts dialog to four other dialogs.

The Controller Editor contains a drop-down list for selecting the specialization target (highlighted in Figure 2). When a specific target is selected, the specialized controller for that target is shown, and can be manipulated. The right screenshot of Figure 2 shows the specialized controller for a Nokia target device. The nodes in a specialized controller are either specialized dialogs (inherited from the generic flow) or concrete device-specific pages. In Figure 2, "InfoTran.." is a device-specific page added for the Nokia target. In addition to adding pages and transitions to and from them, the user can also delete dialogs and transitions from a specialized flow, as shown in Figure 2. Note that a change in a specialized flow applies only to that device, whereas a change to the generic flow is reflected in all specializations.

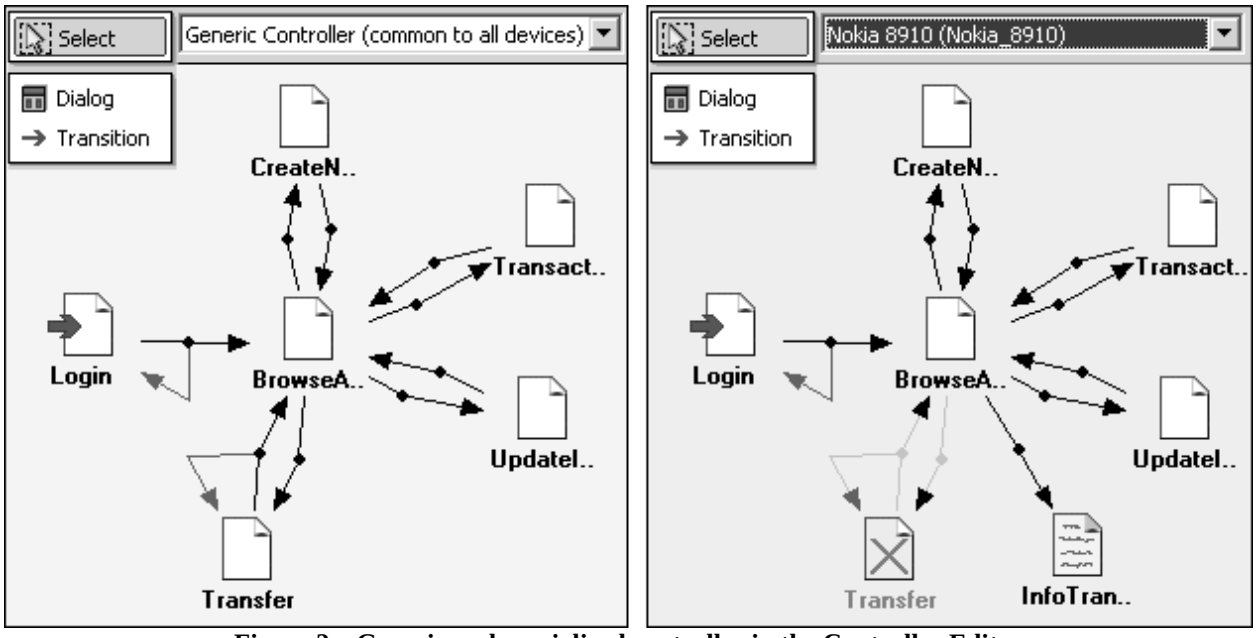

Figure 2 – Generic and specialized controller in the Controller Editor

Besides the visual builder, the Controller Editor contains separate areas for defining the list of target devices, for specifying the list of JavaBean components, and for viewing and editing the XML source representation of the controller.

View Editor

As with the Controller, the View has generic and specialized aspects. We now describe how the View Editor is used to create both generic and device-specific content, and to specify Model associations.

The View Editor is a visual builder for assembling the generic interaction elements in a dialog. Wizards and a property editor are used for specifying elements' attributes. Elements may have numerous attributes, as defined in the XHTML and XForms specs [XForms]. Figure 3 shows two screenshots that illustrates our current implementation. On the left, we see a generic dialog for updating personal information, which contains a number of elements. The visualizations of these elements are not intended to show the actual appearance on any device. Rather, this is an abstract representation that is converted into concrete presentations for particular target devices.

As with the controller editor, the view editor contains a drop-down list for selecting particular targets for specialization. The right screenshot shows the specialized dialog for a Nokia target device. The currently implemented operations allowed for specializing a dialog are limited. Developers can add device-specific page breaks, which force a dialog to be split during view translation. Also, developers can add a specialized submit

button for navigating to device-specific pages in the controller graph. Several other specialized operations, e.g., adding and removing elements and changing properties, are being implemented currently.

Besides the visual builder, the View Editor contains separate areas for associating dialog elements with JavaBean components and for viewing and editing the source representation of dialogs. Each dialog element that admits dynamic values can be bound to JavaBean properties, so that the runtime value of the property is used to populate the element.

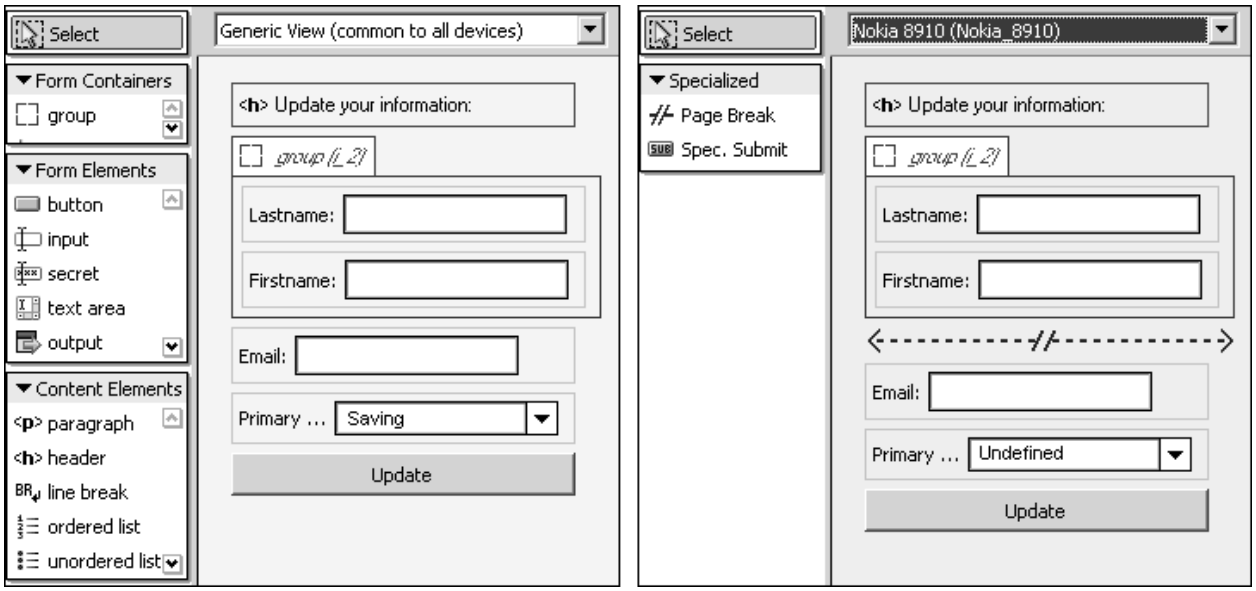

Figure 3 – Generic and specialized dialogs in the View Editor

Device-Specific Application Generation

The MDAT system implements a specialization framework that corresponds to the automated portion of Step 2 in Figure 1. The specialization framework translates generic applications into specialized applications. This framework is constructed as a pipeline of individual stages that can be easily added or removed. Each stage performs a specific portion of the translation from dialogs to specialized JSPs. For example, the HTML markup language pipeline includes a stage that specializes images and another stage that performs automatic page splitting. Each target markup language has its own pipeline.

One of the design goals of MDAT is extensibility: As new devices are introduced to the market, MDAT needs to easily support these devices and any new markup languages that they require. Thus, the requirements for MDAT extensibility include the ability (1) to define new devices, (2) to support new markup language capabilities during application definition (3) to support new markup languages during specialization, and (4) to support different controller runtime frameworks (such as Struts). The pipelines mentioned above, along with other specialization configuration, are defined in an XML configuration file. Extensibility is achieved by allowing Eclipse plug-ins that extend MDAT to modify the specializer's configuration.

View Translator. The first stage in specializing a generic dialog is the translation of the dialog into the markup language required by the target device. To implement this stage in the specializer pipeline, MDAT provides an extensible view translation framework. A *view translator* conforms to a framework interface and provides the translation needed to a particular markup language. Support for a new markup language can be provided by dynamically plugging a view translator into the framework at runtime.

Controller Translator. The Controller Translator is responsible for generating a device-specific controller from the generic application. The generated controller code uses the Struts framework [Struts] to manage control flow in both web applications and portal applications. The Controller Translator generates the required action subclasses,

form bean definitions (created by scanning the generated JSPs), and configuration files. Each transition is associated with a single action subclass, which may invoke a JavaBean method to determine which branch is taken or to call underlying business logic.. The generated configuration files include mappings between the action subclasses and transitions, as well as form bean definitions.

Iterative Refinement

After JSPs are generated through specialization, the developer can optionally apply formatting, layout and other stylistic refinements to achieve a customized presentation for a particular device. These refinement operations can include font changes, image resizing, removal of unnecessary elements, and insertion of header and footer images. As shown in Figure 1, MDAT provides a Refinement Editor that allows the developer to selectively handcraft or "tweak" pages generated for particular devices.

When an application is modified, either by changing the generic view specification or a specialized view specification, a new set of device-specific JSPs is automatically generated for each target device. It is important that handcrafted changes previously created using the Refinement Editor are not lost in this process. This is referred to in the literature as the "round-trip" problem [Medvivovic 1999]. To preserve refinements to generated pages, the Refinement Editor maintains a change history that is used to reapply refinements whenever JSPs are regenerated.

The Refinement Editor supports typical WYSIWYG editing functions, including adding static content, removing content, and changing the properties and placement of elements. The Editor records these editing operations as they are performed. When editing is completed, the refined page is not actually saved, instead a refinement command is created from the edit history. Subsequently, when a newly-generated version of the same page is reopened in the Refinement Editor, the refinement commands are re-applied. The developer can then further refine the page.

The history function of the Refinement Editor is based on the technique of programming by example, where the developer only needs to work with examples (i.e., the initial and refined views), and a refinement command can replicate changes automatically [Hori 2002]. A key feature of this history function is generalization of the refinement commands [Ono 2002], which allows commands to be reapplied to a page that has changed since it was last refined.

Generalization

Although a model-based approach appears to be an effective solution to the problem of multi-device application authoring, it poses some problems for the end user. First, the application developer must learn a new set of tools in order to create the application model. Second, the developer is forced to think about the application in terms of abstract entities. This may be problematic, since most designers are accustomed to working with concrete UI elements and layouts. Third, the approach only works for newly created applications; legacy applications must be completely re-implemented.

We have addressed these problems by developing a facility for extracting the View component of an application model from existing web applications, a process we call *generalization.* Within the MDAT system, we provide support for generalization of HTML pages. Generalization is a two step process. The first step is a completely automated process of extracting a candidate model from an HTML page. In the second step, the application developer views the candidate model and edits it. We will discuss both of these steps in more detail.

An application developer begins generalization by selecting an HTML page, or a portion of a page via rubber-band selection. An automated inference engine is invoked, which extracts a candidate abstract model. The extracted model consists of user interaction elements,sub-elements of these elements such as captions and hints, and groupings.

Figure 4 shows an example of such an extracted model The "Yes" and "No" radio buttons have been grouped together into a single choice list interaction element and the caption "e-mail Access?" assigned to it. Similarly, the input fields "First", "Middle", and "Last" have been grouped together into a phrase, and a caption assigned to the group.

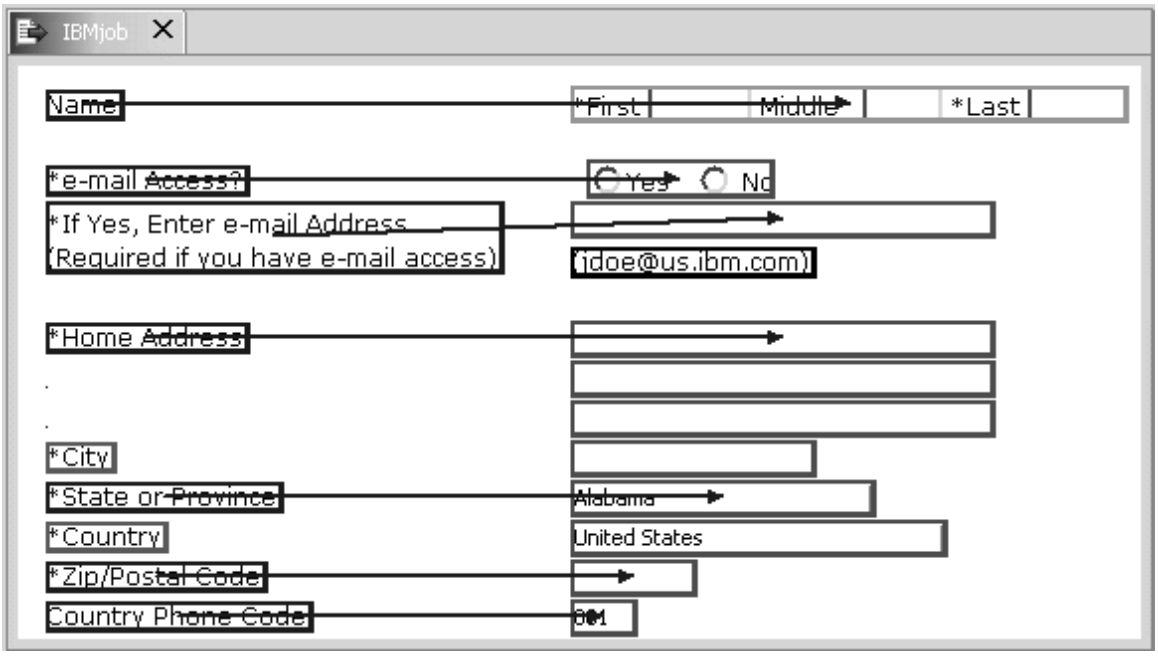

Figure 4. Example of a generic model extracted from an HTML page

The automated generalization process uses a variety of information extracted from the HTML page. In addition to structural information from the document object model (DOM), we also use geometric layout information from a rendering of the page, style information (such as bold or italic fonts) contained either directly in the page or in cascading style sheets (CSS), as well as some content information from strings (such as enclosing parentheses).

Note that the inference process must resolve ambiguities. In the fragment of HTML displayed in Figure 5, the caption of the user interaction element I_1 may be either S_1 or S_2. Our inference engine performs a spatially localized search using a set of heuristics in order to resolve these ambiguities. Even with a sophisticated set of inference techniques, however, the automated extraction is not guaranteed to perform all assignments correctly. This can be seen in Figure 4 where the strings "City" and "Country" have not been correctly identified as captions. For this reason, we provide an interactive visualization and editing interface.

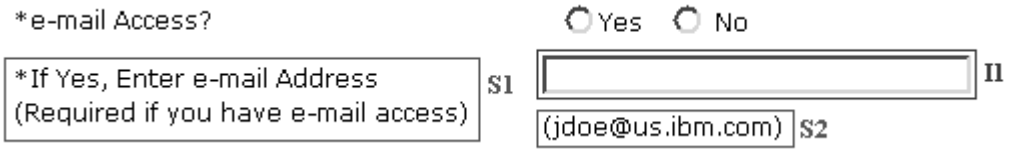

Figure 5. Fragment of HTML Page

Our design goals for the inference engine were to perform the extraction quickly, and to minimize the amount of manual editing required. Most pages are extracted in under a second, meeting our goal for rapid extraction. We performed a study to evaluate the accuracy of the extraction process. This study, described in more detail in [Gaeremynck 2003], compared the model automatically extracted from 48 HTML application pages (i.e., forms) gathered from the web against an "ideal" model manually constructed for each page. Figure 6 shows the results: 40% of the pages had no errors in the automated extraction; 77% had fewer than 10% errors. Our conclusion is that the automated extraction does a reasonable job of minimizing the amount of manual editing required.

The generalization interface (Figure 4) displays extracted user interaction elements and their properties using colored rectangles. Properties are indicated by arrows from the property value (i.e., string) to the associated interactor. Groupings such as phrases are indicated by enclosing rectangles, with a level-of-detail display that shows the contents of the group only when the mouse cursor hovers over the group. Interactive controls are provided, which permit the deletion of model elements, creation of new model elements, and reassignment of properties by

visually manipulating the rectangles and arrows in the display. This interface allows the developer to quickly evaluate and modify the candidate model. Once manual editing is complete, the developer saves the model, making it available, as a generic dialog, to the rest of the MDAT tool set.

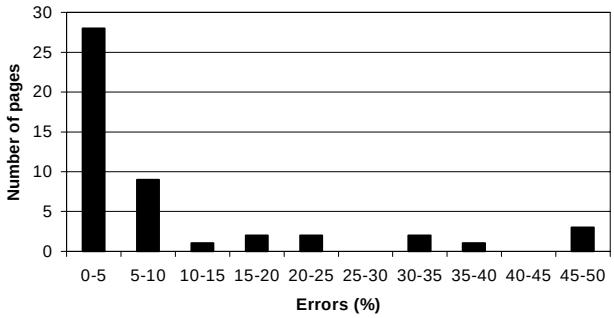

Figure 6. Histogram of evaluation results, showing the number of pages versus the percent errors on a page.

Conclusions

In this paper, we have described a solution for multi-device web and portlet application development, called MDAT. MDAT enables a developer to build a single *generic* application common to multiple devices, and specify the incremental differences to particular target devices. The generic application can be specified using a set of visual builder tools, or partially derived from existing applications using a rule-based extraction tool. MDAT then generates device-specific artifacts for the chosen targets, based on device capabilities. If necessary, the developer can modify the generated artifacts to further handcraft them for individual target devices. These modifications are stored by the tool so that they can be re-applied if the application is re-generated. In addition to design-time generation, MDAT also supports runtime generation, allowing dynamic adaptation to new device types.

MDAT is the second generation of this technology based on lessons learnt from previous research prototypes, including PIMA [Banavar 2003]. The main advances in this second generation are: (1) The combination of design-time and runtime adaptation, (2) A rich notion of device profiles and device categories, (3) A solution to the "round-trip" problem, for modifying generated artifacts while not losing these modifications upon regeneration, and (4) An integrated, end-to-end development methodology, built as an industrial-strength implementation on the Eclipse IDE platform [Eclipse], and incorporating many industry standards such as Struts, JSPs, XForms, etc. The toolset is currently available for public evaluation as part of the IBM Everyplace Toolkit for WebSphere Studio [Everyplace].

The MDAT application model builds on the concepts in model-based UI development [Szekely 1993, Eisenstein 2001]. The key distinguishing features of MDAT are the notions of specialization, iterative refinement, and generalization, all targeted specifically to multi-device web application development. Specialization supports addition and removal of elements of the view or controller descriptions of web applications. Iterative refinement with round-trip support for presentation artifacts builds on software engineering notions of capturing changes to generated code for inclusion into the original model [Medvivovic 1999]. Generalization of HTML into a generic model is based on concepts from reverse engineering of user interfaces [Bouillon 2002, Vander Zanden 1990]. Another well-known end-to-end system for multi-device authoring of web applications is the Microsoft ASP .NET Mobile Controls (formerly known as the Microsoft Mobile Internet Toolkit). This technology does not support the notion of design-time adaptation, nor does it support specialization, iterative refinement, or generalization. This solution is based on runtime interpretation of a device-independent application.

Future releases of the MDAT technology will strengthen and extend the toolset in several areas. We are currently in the process of performing an in-depth field analysis to refine the usability and utility of the MDAT system. We plan to add rapid application development tools throughout the toolset, including the ability to specify business logic and data at a higher level. We also seek to be able to generalize not only from HTML pages, but from entire pre-existing web applications, including both the view and the controller. Another major area of extension is the support for non-GUI modalities such as Voice and Ink.

References

[Banavar 2000] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: An Application Model for Pervasive Computing. In Proceedings of the Sixth annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pages 266--274, August 2000.

[Banavar 2003] Banavar, G., Bergman, L., Gaeremynck, Y., Soroker, D., Sussman, J., Tooling and Systems support for authoring multi-device applications. To appear in Journal of Systems and Software, Elsevier Publications, 2003.

[Bouillon 2002] Bouillon, L., Vonderdonckt, J., Souchon, N., *Recovering Alternative Presentation Models of a Web Page with VAQUITA*, Proceedings of CADUI '02, pp. 311-322.

[Eclipse] http://www.eclipse.org. Home page for the Eclipse open, extensible, IDE platform.

[Eisenstein 2001] Eisenstein, J., Vanderdonckt, J., Puerta, A., 2001. Applying model-based techniques to the development of UIs for mobile computers.In Proceedings of IUI 2001, Intelligent User Interfaces, Santa Fe, New Mexico, USA, January 14–17, 2001.

[Everyplace] http://www-3.ibm.com/software/pervasive/products/mobile_apps/everyplace_toolkit.shtml. Web URL for IBM Everyplace Toolkit download, which includes beta release of MDAT.

[Hori 2002] Hori, M., Ono, K., Koyanagi, T., and Abe, M.: Annotation by transformation for the automatic generation of content customization metadata. In F. Mattern and M. Naghshineh (Eds.) *Pervasive Computing, First International Conference, Pervasive 2002*, Lecture Notes in Computer Science 2414, pp. 267-281, Zurich, Switzerland (2002).

[Gaeremynck 2003] Gaeremynck, Y., Bergman, L. D., Lau, T. MORE for less: Model recovery from visual interfaces for multi-device application design. Proceedings of IUI 2003 – International Conference on Intelligent User Interfaces, Miami, Florida, January 2003.

[JavaBeans] *JavaBeans: Developing Component Software in Java*. Elliote Rusty Harold. ISBN 0764580523. IDG Books.

[JSP MVC] Govind Seshadri. Understanding JavaServer Pages Model 2 Architecture. JavaWorld Magazine. December 1999. Available from http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[Medvivovic 1999] Medvivovic, N., Egyed, A., Rosenblum, D.S., 1999. Round-trip software engineering using UML: from architecture to design and back. In: Proceedings of the 2nd Workshop on Object-Oriented Reengineering (WOOR), September 1999, pp. 1–8.

[Ono 2002] Ono, K., Koyanagi, T., Abe, M. and Hori, M.: XSLT Stylesheet Generation by Example with WYSIWYG Editing. *Proceedings of the International Symposium on Applications and the Internet (SAINT 2002)*, pp. 150-159 (2002).

[Struts] The Apache Struts Web Application Framework. http://jakarta.apache.org/struts/

[Szekely 1993] Szekely, P., Luo, P., Neches, R., 1993. Beyond interface builders: model-based interface tools. In: Proceedings of ACM INTER-CHI_93 Conference on Human Factors in Computing Systems 1993, pp. 383–390.

[UAProf] Wireless Application Group User Agent Profile Specification. November 1999. Available at http://www.wapforum.org/what/technical/SPEC-UAProf-19991110.pdf

[Vander Zanden 1990] Vander Zanden, B., Myers, B.A., 1990. Automatic Look-and-Feel Independent Dialog Creation for Graphical User Interfaces CHI 90.

[XForms] W3C. XForms – The Next Generation of Web Forms. http://www.w3.org/MarkUp/Forms/