Architectural thinking and modeling with the Architects' Workbench

S. Abrams
B. Bloom
P. Keyser
D. Kimelman
E. Nelson
W. Neuberger
T. Roth
I. Simmonds
S. Tang
J. Vlissides

Collecting and organizing all of the architectural information for a system is a challenge faced by information technology (IT) architects. Transforming that information into models of a viable architecture and keeping associated work products consistent and up to date is an even greater challenge. Despite this, model-centric architectural methods are not as widely adopted or as closely followed as they could be, partly due to a lack of appropriate tools. The Architects' Workbench (AWB) is a prototype tool that addresses these problems and supports the creative process of architectural thinking and modeling. This paper presents key AWB innovations and discusses how their design was motivated by insights into architectural work and feedback from IT architects. We describe the design of AWB itself as a metamodel-driven and method-based tool, and we report on experience from the use of AWB in production environments.

INTRODUCTION

IT architects are faced with a formidable information mangement challenge as they design systems to address customer needs. For example, in designing a new Web application that will serve as the single point from which a customer of a large financial institution can access all of his or her financial information, it is often necessary to integrate hundreds of legacy systems dealing with various financial instruments, dozens of databases storing account information, and myriad rules and constraints. The challenge for the IT architect is to organize and analyze initial descriptions of customer needs as well as the existing IT environment, and to design an appropriate solution. This activity often involves progressively formalizing information and building up architectural models. The resulting models and design rationale must then be incorporated into a number of overlapping work product documents for a variety of stakeholders, and these documents must be kept up to date and consistent as the system evolves.

IT architecture

To better understand this challenge, we consider IT architecture in greater detail. Many definitions of IT

architecture exist (see, for example, Reference 1), but most agree that an architecture describes the structures of an IT system—both hardware and software—and their relationships to one another. It defines components that need to be bought, built, or reused, focusing on their externally visible properties. It defines the operational infrastructure (e.g., servers, network connections, etc.) on which these components will be deployed, and it ensures that the system will meet its functional and nonfunctional requirements (to be defined in the following). IT architecture also provides a high-level breakdown of the necessary development work and documents key decisions and their underlying rationale.

There are several methods and languages employed for describing architectures, such as IEEE 1471,[2] the Rational Unified Process* (RUP*) "4+1" view of architecture,[3] and the Unified Modeling Language** (UML**[4]). In the IBM Global Services organization (IGS), an IT architecture is documented in a number of work products, often using the Architectural Description Standard (ADS[5]) for terminology and notation, all of which is defined by Global Services Method (GS Method). Regardless of the methods followed, architectures are typically implicitly connected sets of models, usually documented with combinations of diagramming, spreadsheet, and word-processing tools.

There are a number of GS Method work products that are often among the work products that an IT architect is responsible for producing. The *system context* work product shows the IT system solution as a black box that exchanges information with specified external actors—human users and other IT systems. This documents the scope of the solution. The *architecture overview diagram* illustrates key elements of a solution, such as actors, locations, components, servers, network connections, and subsystems. This work product draws elements from across the breadth of an architecture, for purposes of communicating key concepts to various external stakeholders and sponsors. The *use case model* elaborates on the information exchange between the IT system solution and the external actors. This work product describes functional requirements of the system, and is often annotated with nonfunctional requirements (i.e., requirements that pertain to system qualities or constraints, such as performance, availability, security, and standards compliance).

The *nonfunctional requirement* (NFR) work product collates nonfunctional requirements from throughout the architecture. The *component model* defines software components and, their responsibilities, interfaces, and static relationships such as dependencies, as well as the dynamics of the collaborations by which components interact to support use case scenarios. This work product documents the functional aspect of the solution. The *operational model* documents the infrastructure of the solution and the deployment of the application components onto that infrastructure at three levels: conceptual, specified, and physical. Finally, the *architectural decision* work product documents the architecturally significant decisions that were made across all aspects of the architecture, along with alternatives that were considered and the rationale for the choices that were made. This work product is critical for understanding a solution, for preserving its integrity as it undergoes maintenance and evolution, and for reusing parts of this solution in other designs.

Although GS Method defines many other work products, these are some of the key architectural ones. These work products illustrate the interconnected nature of an architecture, as there are many relationships among the elements described in each of these work products. In an ideal world, these work products consistently document different aspects of a single architecture and combine to produce a complete, coherent, and unambiguous picture of the system under design.

The balancing act

In the field, architects spend much of their time scavenging mounds of unstructured information for useful tidbits—in stark contrast with neat descriptions of well-structured work products. A path from the copious incomplete, inconsistent, and informal material gathered from meetings and existing documents to the well-structured specifications required for solution development is unclear.

Architects often start by gathering vision statements, sketches of requirements and system structures, descriptions of existing systems with which to integrate, and other informal documents. They add structure and rigor, sketching and fleshing out ideas as their understanding deepens. When they do formalize, it is often only to the extent that it helps clarify important ideas for themselves or their colleagues. Architects are constantly balancing

opposing forces, thinking fluidly but within well-defined structures.

Architects need to grasp myriad details without losing sight of the larger picture. They need to respond rapidly to high-level changes and understand their impact throughout the architecture. In addition, they must interact with a steadily increasing set of stakeholders as disciplines such as enterprise architecture, service-oriented architecture and asset-based consulting business architecture mature. They must balance the specific needs of their solution against enterprise-level standards, including technical standards ensuring coexistence and interoperability.

Time-to-market pressures can cause architects to trade precision for expediency. They try to rapidly capture their thoughts (even if incomplete and ill-formed) and then clarify them and remove inconsistencies as time permits. They juggle ideas, sketch out approaches, weigh alternatives, articulate a vision to various stakeholders, and produce the work products, usually as deadlines loom. Among other things, the transformation of the unstructured material into a coherent set of work products is a huge information management challenge. Throughout the architectural process they analyze, consider issues and trade-offs, raise and address concerns, and make decisions. For all of this they rely upon their training, drawing on their experience, methods, interactions with colleagues, documented best practices, and whatever tools are available.

The tools of the trade

Despite a number of special-purpose modeling tools (such as Rational* Software Architect or the Telelogic System Architect**), IT architects often use the same tools to create an architecture that high-school students use to complete a homework assignment, namely, standard office-presentation, spreadsheet, and word-processing tools. These tools are sometimes tailored to produce one or more of the output work products discussed earlier, but often without regard for the thought process that goes into composing them or the interrelationships among the artifacts. For example, architectural decisions are typically documented with a tool such as Microsoft Word, which has no explicit representation of the domain of architecture, while operational models are often documented with Microsoft Visio**. Not only do these tools lack semantics, but they are not oriented toward easily interacting with and consistently maintaining a large network of related and only partially complete models.

Even when a special-purpose modeling tool is used for one aspect of an architecture, common office tools are still used for the other parts of the architecture, and similar deficiencies exist. As an example, when component models are created with Rational Software Architect, interrelationships with operational models made with Visio or architectural decisions described in Word documents are not possible.

Without a tool that understands the many facets of architecture and their interrelationships, developing an architecture is tedious and prone to lapses that ultimately result in failures of the designed system to meet its requirements. Maintaining the architecture and continuing its evolution is even more daunting. In practice, many architectural documents quickly go out of date as the system evolves, creating difficulties for those charged with maintaining or enhancing the system.

Furthermore, while developing an architecture is a journey from an unclear understanding of unstructured information to a well-structured specification, the tools typically employed do not support this transition. Some tools are well-suited for unstructured information—word processing tools, semantics-free drawing programs, or white boards, for example. Other tools are well-suited for more structured information—UML diagramming tools or formal requirements management tools, for example. However, none of these support the transformation from unstructured to structured information that is inherent in the task of developing an architecture.

A tool facilitating the practice of the art of IT architecture

Our goal was to design a tool that supports the realities of practice, while supporting the formalities of a given architectural method. To that end, we formed a team of researchers and IT architects and analyzed the working methods and thought processes of the architects. Based on this analysis, we incrementally built a tool that attempts to match not only the formalisms of the practitioners' methods, but also the realities of their creative process and working styles. We wanted a tool that would let

architects balance formalism and freedom, while helping them transform unstructured information into sufficiently formal work products. The initial result of this work is the Architects' Workbench (AWB).

After a brief overview of AWB from a user's perspective, we discuss in more detail some of its key features and its underlying design rationale. Next, we discuss the overall design and architecture of AWB, with particular emphasis on those design decisions that facilitated key developments. We then report on users' experience with AWB in field trials, where IT architects have found AWB to be highly effective in "live" use in major customer engagements. Finally, we conclude with a discussion of related efforts and future directions for AWB.

Without loss of generality, examples and method-specific discussions in the remainder of the paper will be based on GS Method and ADS. Nonetheless, these discussions apply equally to other methods and metamodels. In fact, as will be described later in the paper, the metamodel- and method-related parts of AWB have been factored out as "pluggable components," and AWB can be (and has been) used as a workbench based on any of a number of different methods and metamodels.

A BRIEF TOUR OF AWB

AWB is an Eclipse**[6]-based tool, providing several views and editors tailored to the "balancing act" described previously. ***Figure 1*** shows the AWB user interface (UI) configured for system context work—that is, sketching the boundaries of the system and its relationship to associated systems and people. In this example, the *Music Web System* is being developed. The *project view* is on the left side of the UI. The Outline pane of the project view consists of a textual hierarchy that shows the model elements important for system context work, starting with the Music Web System. The Reminders pane shows model elements requiring attention. For example, under "Actors not involved in Exchanges," we can see that actors named "Fans," "Record execs," and "Artists" have been identified but have not yet been placed in the system context as being involved in exchanges.

The middle section of the AWB UI holds editors for specific model elements. The top, rich-text editor shows "BHR meeting notes," containing the notes taken by the AWB user during a meeting with the (hypothetical) client BHR. The underlined phrases in the note are hyperlinks to model elements. The lower editor shows the Music Web System Context diagram in progress.

On the right side of the UI is the palette view. Its Prototypes pane shows a list of the kinds of elements that are likely to be needed during system context work. Users can instantiate these elements, often automatically linking them to selected elements in the model. The System Context pane of the palette view shows existing model fragments that the user may want to incorporate into the system context, such as exchanges (general categories of information flowing between the Music Web System and its users). Similarly, the Nonfunctional Requirements pane shows other model elements that are useful to have on hand.

AWB employs wizards to simplify complex tasks. For example, there is a Relator wizard, which helps to determine the kind of relationship to use when connecting model elements. There is also a Refactor wizard that facilitates splitting and merging model elements, and transforming model elements from one type to another.

With AWB, a practitioner typically begins with textual information; that is, some combination of preexisting documents and notes captured from meetings with the stakeholders. The user either types or pastes these into AWB's rich-text editors. The user can study these notes, identifying important bits of information. With the *markup and model* technique, he or she can create model elements directly from the text. In response, AWB creates bidirectional hyperlinks to assist in navigation between the notes and the model elements.

When the time comes for more detailed modeling of some aspect of the system, the user chooses an appropriate activity from the viewpoint menu. (A viewpoint is a configurable mode for displaying the panes of AWB.) This directs AWB to present the model in its current state in a form well-suited to the chosen activity. The hierarchical project view alters itself to show the relevant model elements and relationships for this activity, and the palette view presents relevant prototypical model elements, assets, and peripherally interesting portions of the model to the user. The Reminders pane in the lower

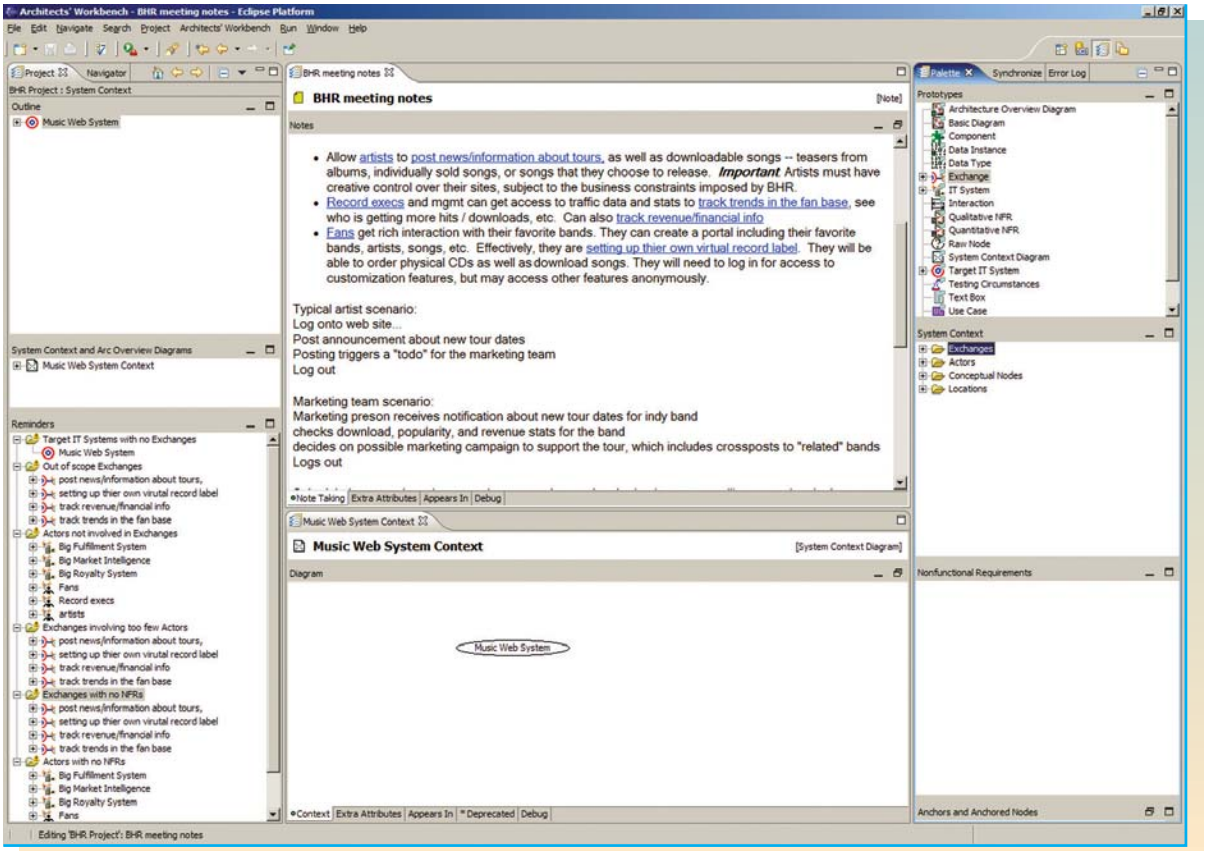

Figure 1
AWB configured for system context, showing reminders

left corner presents omissions and recommendations that are appropriate to the chosen activity, allowing the practitioner to focus on issues that are relevant to the task at hand.

AWB provides many ways of viewing and interacting with models. At any time, the architect can begin to visualize relationships among elements by using the basic diagram. As understanding of the solution evolves, the user can refactor the model, refining the elements involved and specifying the relationships among them and elaborating the details of the model in customized diagram editors (such as those for system context, component interactions, and operational modeling), while maintaining connections among the model elements and traceability back to less-structured information. The user can move between textual, tree-based, tabular, and graphical representations of the model and can generate work products to check the state of the architectural documents. The documents can often be edited in WYSIWYG ("what you see is what you get") views, and the changes are immediately reflected back to the underlying model. As the user switches among the various activities needed to complete the architecture, AWB responds with task-appropriate guidance and presentations of the model.

We have briefly outlined many of AWB's capabilities. In the following sections, we will discuss in more depth some of its key features, focusing on how they address the problems described previously.

AWB FEATURES

Several principles guided the design of many AWB features. We wanted AWB, where possible, to support multiple projections (or views) of a single, integrated model, rather than a number of distinct models that users would have to keep consistent manually. We wanted to support the fundamental thought processes used in the practice of IT

architecture, in particular helping users find order in the chaos by assisting them in organizing unstructured information. We wanted to support natural input and output mechanisms, which means not only generating work products, but also allowing users to work with diagrams, trees, tables, or forms, as appropriate. The following discussion of AWB features is grouped around these principles.

One principle is common to all features; that principle is to focus on the work, not the output. Many software tools are focused on supporting results of work, rather than the work itself. As a simple example, typical word-processing programs support font selection, layout, tables, and other typographic issues, rather than helping writers think through the development of an introduction, a hypothesis, supporting ideas, and a conclusion. Similarly, typical architectural modeling tools support the production of specific artifacts (such as class, activity, or deployment diagrams, as in the case of UML), and not the process of thinking through the ramifications of key design decisions in the architecture. Just as putting words on paper aids in thinking during the writing process, documenting architectures in diagrams assists in their design. However, tools should better support the development of works in progress and the evolution of ideas from their initial unstructured states to their more refined forms, as embodied in the final outputs. Similar design philosophies motivated the development of an earlier system in a different domain—that of music composition.[7] While the mechanisms used are not the same, the philosophies and design trade-offs discussed in Reference 7 for that work are quite similar to those of AWB.

Finding order in the chaos

AWB features aimed at bringing order to chaotic, unstructured inputs are detailed in the following subsections.

Capture before modeling

Early in the development of AWB, we recognized that capturing information is an activity in itself. It is hard to structure information while capturing it, especially if it is not clear how it should be structured. Accordingly, AWB provides facilities for capturing "raw" information. Capture is supported by the viewpoint "Note Taking" and by facilities for storing and pasting rich text. In this viewpoint, the architect can create collections of free-form rich text artifacts called Notes. Architects can use simple text formatting to add some organization while capturing thoughts and ideas in a meeting. Existing documents can also be pasted into Notes. This capability is critical. Furthermore, most model elements also have a rich text "raw notes" field, allowing raw information about each element to be stored.

Markup and model

Including rich text facilities in a modeling environment is not novel, but allowing the text to serve as input for the modeling process is. Because architects often glean key requirements and constraints and architectural opportunities from existing documents and notes taken during meetings, we developed "markup and model" capabilities, which enable an architect to select a word, phrase, or passage within a note and create a hyperlink from it to a newly created element. AWB creates the new element of the appropriate type, labels it based on the selected text, and turns the selected text into a hyperlink to the new element. AWB also creates a "refers to" relationship between the note and the new element, ensuring traceability from the new model element back to the note from which it was created. When the hyperlink is followed, an editor is opened on the newly created element. Markup and model can also be used to link a phrase in a note to an existing model element, allowing several textual references to a model element to exist within a single note, or even in multiple notes. Conceptually, the markup and model function is similar to Skuce and Lethbridge's CODE systems[8,9] although, as described later, AWB's reminders and viewpoints are activity-specific, whereas CODE's maps and notifications are not.

Progressive markup and model through semantic hyperlinks

The markup and model capability is available in all rich text fields, including text in notes as well as "raw notes" fields. When appropriate, it can create relationships other than the generic "refers to" relationship. We call this "semantic hyperlinking," and *Figure 2* illustrates it.

The leftmost rectangle in the figure shows notes that were taken by an architect in a meeting with a customer. Even though the architects were thinking in terms of GS Method and concepts such as use cases, it was nevertheless effective to first capture the discussion in meeting notes along with the varied information about servers, locations, processes, people to talk to, things to do, and other

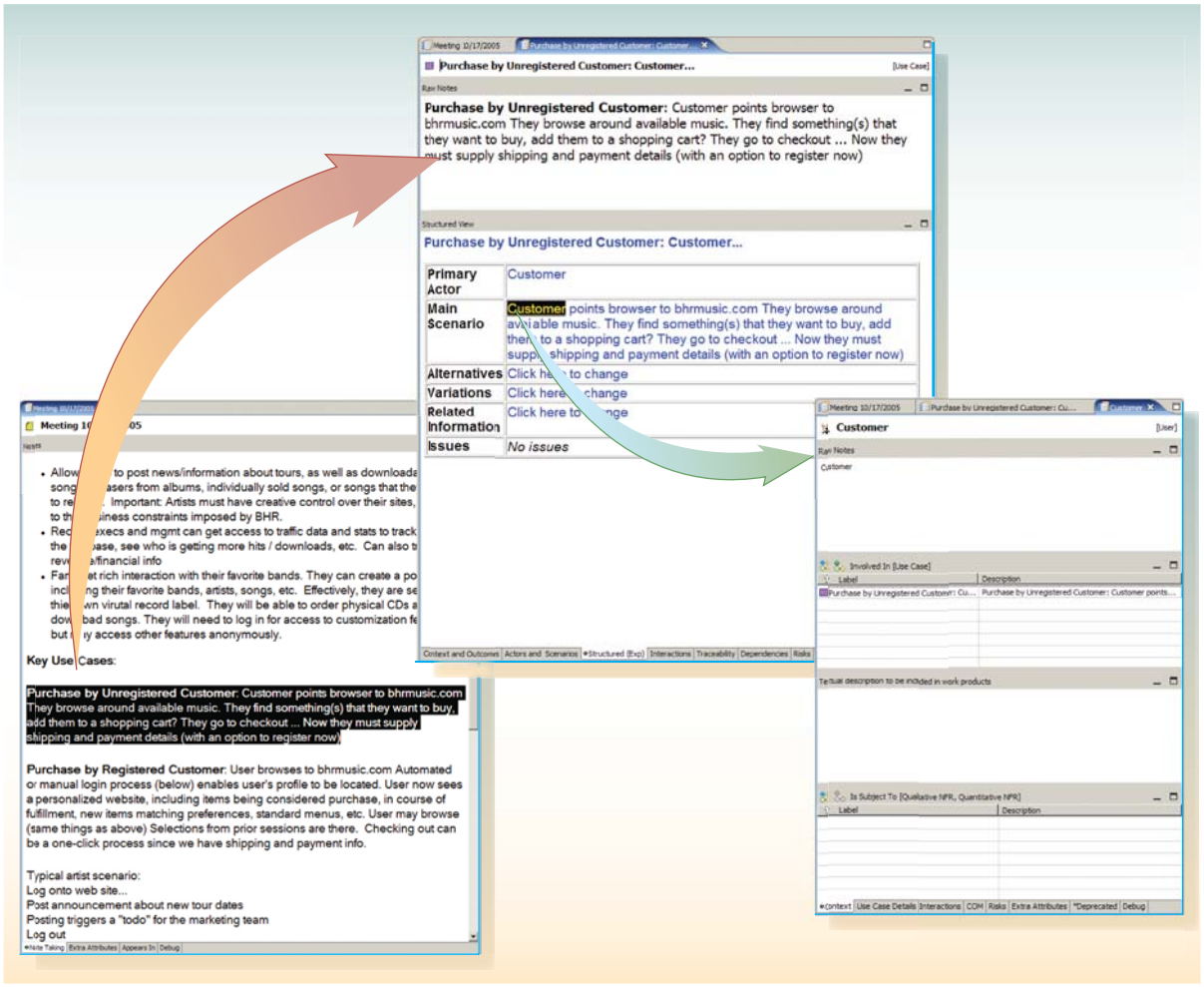

Figure 2
Progressive markup and model

issues that came up. In reviewing the notes after the meeting, the architect noticed a large passage of text that related to a use case. The architect selected the relevant text, starting with "Purchase by Unregistered Customer," and created a use case model element from it (shown in the middle rectangle). When a large passage of text such as this is selected, an elided substring is used for the label of the created model element, and the entire text is copied into the "Raw Notes" field of the new model element. In this way, a large passage of text is quickly inserted into a model element, without concern for its structure. (Structuring typically takes place later.) In this case, after looking at the new use case, the architect cut and pasted the bulk of it into the "Main Scenario" field of the use case, where it would later be refined for inclusion in the use-case work product. Noticing that the scenario mentioned a kind of actor, the architect selected the word "Customer" and created a new actor model element (shown in the rightmost rectangle). AWB consulted the metamodel, found that 'Actors' can be related to use cases with the "Involves" relationship, and automatically created this relationship and associated it with the hyperlink.

In this way, the initial simple pieces of model structure can be progressively assembled by marking up and gradually adding structure to plain text. *Figure 3* shows this fragment of the model after the architect marked up a second paragraph of the meeting notes to create another use case and actor. In some sense, parts of this work derive from semantic linking and hypertext as in the Vannevar Bush Memex[10] and Engelbart's Augment.[11] These

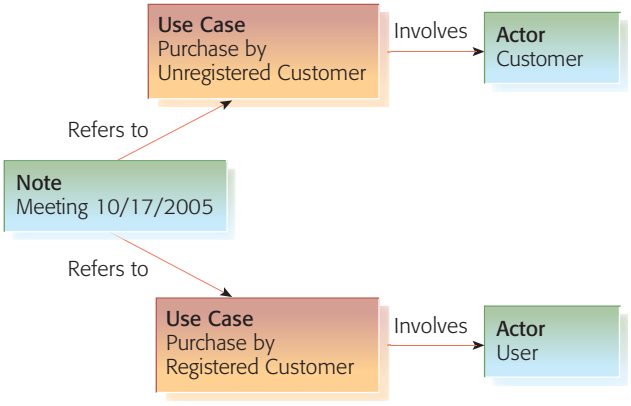

Figure 3
Model fragment

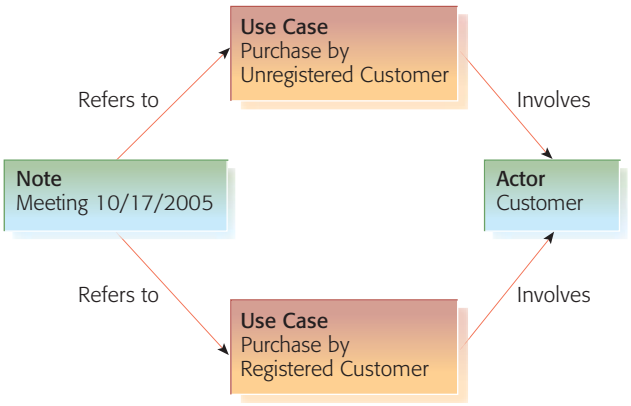

Figure 4
Model fragment after refactoring

works were precursors to modern Web and "wiki" concepts. Augment was used as a knowledge management tool by defense intelligence groups for analysis work.

High-level sketching

AWB users were also interested in using simple diagrams to visualize relationships among elements, even before determining which type of relationships were appropriate. Basic diagrams in AWB address this need, allowing a kind of high-level sketching without the complications of semantics. Model elements can be placed on the drawing canvas, and generic relationships between them can be established, selectively hidden, or displayed individually or by relationship type. Basic diagrams have been used to produce architecture overview diagrams, but they are also useful thinking tools, when coupled with the refactoring and refinement operations that allow gradual structuring of models on the canvas.

Refactoring

To further assist in model refinement, AWB provides refactoring wizards to merge or split elements. In our example, the architect may realize that the inevitably imprecise language of informal meeting notes suggests that "Customer" and "User" are independent actors, when in fact they refer to the same thing. As such, AWB is asked to merge the "Customer" and "User" elements and, with a little help from the merge wizard, transforms this part of the model into that shown in ***Figure 4***. Similarly, the architect may choose to split a model element, and the ensuing wizard helps allocate the original element's relations among the resulting elements.

In addition, AWB lets users change the type of an element. This is a significant capability, as misunderstandings in categorizing meeting notes are inevitable. The Change Type wizard assists the architect in dealing with attributes and relationships as the model element is transformed. This transformation may result in errors as the ramifications of the type change propagate throughout a model. For example, a system may be misidentified as an actor because the system's acronym sounded like a person's name. When this is realized, much information about this system may have already been captured. The change type refactoring allows the architect to focus on the ramifications of the misunderstanding while keeping the bulk of the captured information intact. This is an important feature, which preserves the flow of architectural work.

One model, multiple viewpoints

As discussed, AWB encourages opportunistic identification of many different kinds of elements, facilitating architects' workflow. To avoid consistency problems, AWB keeps all of these elements in one model. To avoid information overload, subsets of that model are presented when appropriate, using viewpoints to support specific activities. When alternative views of the model are required, AWB generates them.

Generating results from a single model

One benefit of the single-model approach is that each element referred to in the architecture is

represented only once. Although different facets of an element may appear in different contexts or work products, this does not require multiple instances of that element. As an example, an actor can initiate a use case, be located in a given location, and have associated nonfunctional requirements without requiring multiple actors in the model for each of these aspects. Each of the work products referencing that actor extracts appropriate information from the model. With many of the architectural tools typically employed, this is not simple to implement.

One example of this is the automatic filtering of the operational modeling diagrams. In many cases, architects model a system of systems—a set of applications deployed using a common infrastructure and sharing some services. It is often necessary to separately visualize the operational model, focusing only on those nodes that are relevant for a particular application. Our operational diagrams can automatically filter themselves based on a number of criteria (which system is being focused on, which physical environment is being modeled, etc.), saving the user from having to explicitly maintain a set of diagrams. From one model, a family of diagrams can be generated by specifying different filtering parameters.

Template-driven work products

In response to complaints from architects that prior tools led them to focus on presentation rather than content, we decided to reverse that in AWB. Thus, AWB deliberately provided the modeling facilities outlined earlier together with a document generator based on Extensible Markup Language (XML) and Hypertext Markup Language (HTML). This document generator creates work products—documents that combine boilerplate text, diagrams, and textual descriptions of the model elements. In this way, users focus on the model, and AWB generates the documents.

Because the precise structure, layout, and style of work products may vary, AWB's document generation engine is driven by user-defined templates. By creating a new template or by modifying a predefined sample template, architects can produce a custom system context, operational model, component model, or other work product that meets their customer's needs and documentation standards. Within a template, as described below, instructions for gathering text and diagrams from the model (using the AWB query language Quetzal, described later) are combined with the HTML constructs needed to format query results into paragraphs, sections, and tables of a document, respecting defined style constraints. For example, a fragment of an operational-model work product may include all of the conceptual nodes within a particular location and create an HTML level-3 section labeled with the value of the node's name attribute, followed by the node's rich text description and tables of its deployment units and its connections to other nodes.

For this scheme to be viable, AWB includes textual information in its models. By combining textual and semantic information in one model, maintaining the set of related architectural documents is greatly simplified, even as the model evolves. Users simply regenerate the documents as needed.

Editable work products

Unfortunately, the generated documents can look quite different from most of the views and editors that AWB presents for interacting with the model. At times, this was inconvenient and even confusing for practitioners when, for example, they tried to find the source in the model for a text description of the work product. Furthermore, for many practitioners and for some kinds of modeling, forms, templates, and tables are very natural input mechanisms. We therefore began to explore what it would mean to allow the generated work products to be used as input devices as well as output artifacts.

To address this, we adopted a "modeling by forms" approach. Using this approach, with a little extra work a template designer can create a template that generates a document in which practically all model-derived parts are editable when viewed in an embedded browser-based editor. Precisely the same engine is used to generate content either for a printable work product or for this document-centric editing. The template includes markups that are instructions to the editor to make certain fields editable and includes enough embedded information to allow it to convert the document changes into model changes. By breaking down large document templates into parts that cover particular model elements (such as the section of an operational-model work product that pertains to a specific location), AWB can use a series of WYSIWYG editors whose layout and contents match sections of the documents that AWB generates and which are under the control of architects. For example,

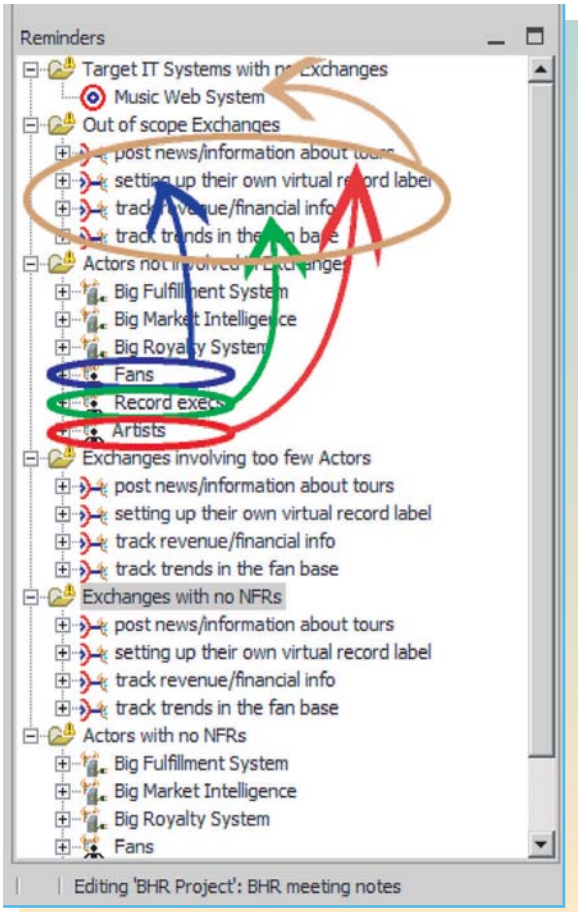

Figure 5
Resolving reminders in AWB

architects who are more comfortable entering the details of a use case in a form-based text editor can do so, while avoiding any "round-tripping" problems that can arise when going between external documents and AWB models.

Activity-centric modeling

The behavior of AWB is responsive to the user's current activity through the viewpoint mechanism. The following subsections detail this mechanism and its interaction with the reminders feature.

Viewpoints: Focusing on one activity

A user tells AWB the activity he or she is engaged in by selecting a viewpoint from the project view. The choice of a viewpoint determines several aspects of AWB's behavior. First, it defines the contents of the project view, presenting a relevant subset of the model in a hierarchical form. For example, the system context viewpoint lists one or more systems, together with the exchanges and actors that represent, at the highest level, each system's interactions with its environment, as well as model elements representing system context diagrams. Second, the viewpoint determines the contents of the palette view, offering prototypes and existing model content suitable for assembly during the selected activity. In the system context viewpoint, this includes target IT systems, exchanges, actors, and system context diagrams. The changing project and palette views are another example of the "single-model" principle, although driven by activity. Selection of viewpoint also determines which reminders are present, providing activity-specific progress guidance, similar in principle to the guidance provided by the WayPointer modeling product.[12]

Defined by a domain expert and stored as part of the metamodel, the viewpoint specification essentially consists of definitions of each of the panes in these project and palette views. Each pane is specified as a set of queries that are run on the model and a set of hierarchical structures in which to organize the query results. The query language used is Quetzal, which was specifically designed for dealing with AWB's modeling constructs and is discussed in more detail later.

Editors containing activity-specific pages are also responsive to the selected viewpoint. For example, exchange model elements are relevant as part of a system context model, and they can also be used during use case modeling to group use cases. The Use Cases page of the Exchange editor shows additional information about how the exchange has been refined into use cases. When an editor is opened, it is automatically set to show the relevant page, based on the currently selected viewpoint. By ordering and placing at hand those resources and elements that are needed, a viewpoint helps the architect focus on an activity.

Handling loose ends: Reminders and Quick Fixes

Populating models by marking up documents or creating informal diagrams can quickly lead to fairly large models. Typically, users create many disassociated parts of models, along with the occasional redundancy. This parallels the thought process involved in architectural work, as architects often work with fragmentary information, which is full of loose ends too numerous to track manually. AWB

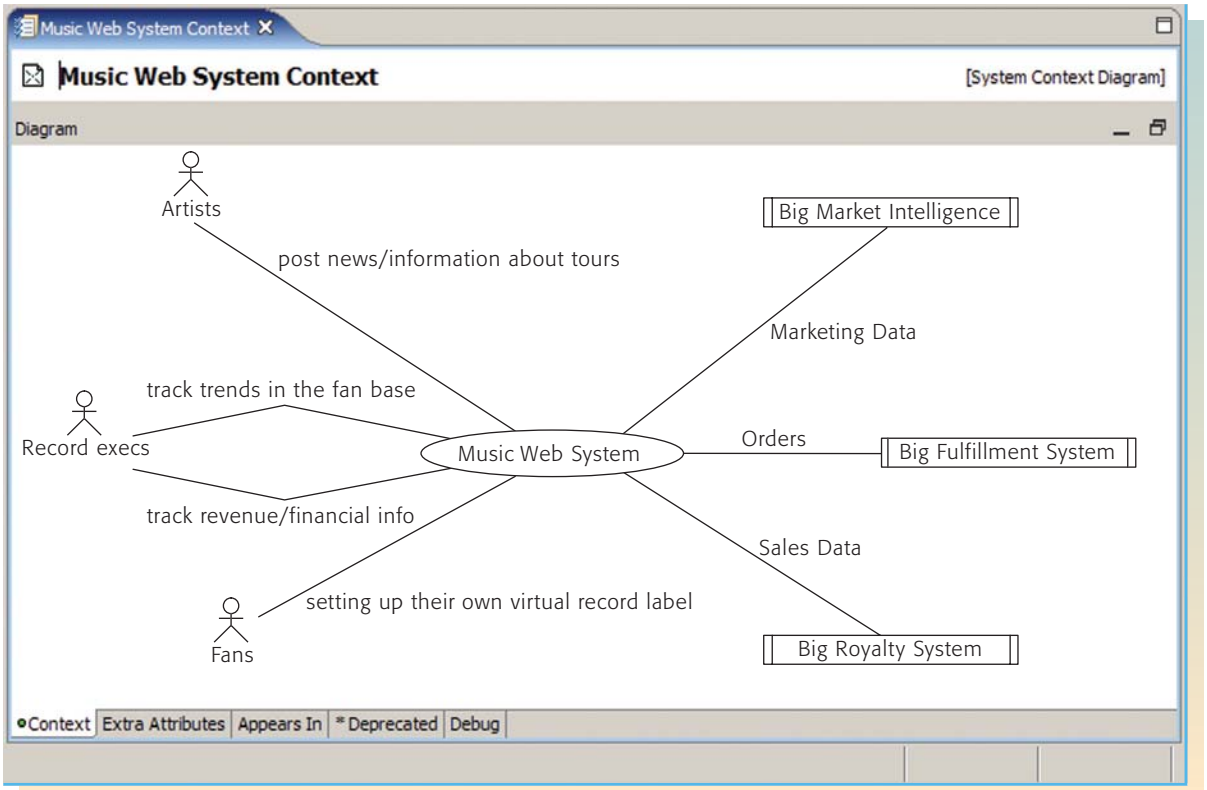

Figure 6
System context with reminders resolved

handles loose ends with *reminders*, shown in the bottom-most pane of the project view in most viewpoints. Typically, reminders show elements that have been identified but not yet properly incorporated into the model, which suggests that there is more work to be done. In the implementation, reminders are also queries run against the model and can therefore illustrate anything computable by Quetzal. AWB only presents reminders when the focus is on an activity for which they are relevant, as determined by the viewpoint.

In the example shown in Figure 1, after gleaning system context elements from a note, the architect is reminded (in the lower left pane) about existing elements that may be relevant to the assembly of the system context. With the series of drag-and-drop gestures shown in ***Figure 5***, reminders are resolved and disappear, and the architect rapidly assembles the system context diagram as shown in ***Figure 6***.

An architect may resolve each reminder at any time and in any way that he or she chooses, or an architect may decide to ignore it completely. AWB does not insist that reminders be handled and will, in fact, tolerate many kinds of errors and inconsistencies in the spirit of Reference 13. An out-of-scope exchange may be deleted, brought into scope through dragging and dropping, merged with another exchange, or even refactored to become a use case for later consideration during a more detailed phase of modeling.

Viewpoints have evolved over time to include sufficient information to ensure that architects need not constantly switch viewpoints to look for the elements that they need in their work. Reminders act as a propellant that helps push practitioners through the design process. A common pattern is that work done in one viewpoint generates reminders to be addressed in other viewpoints.

As an example, consider an architect who has been exploring how to support a set of use cases by mapping out component interaction diagrams for representative scenarios. As each scenario is de-

constructed and responsibility is allocated to representative components, new components, new operations, and new intercomponent dependencies are introduced. After designing sequences for a number of scenarios and accumulating many operations and dependencies, the architect steps back and consolidates the component model. In doing so, the architect needs to view the model with components at center stage rather than use cases or scenarios. Switching to the component modeling viewpoint, the architect sees a hierarchy with components decomposed into interfaces, and interfaces into operations. In this viewpoint, reminders indicate that new operations have been introduced and need to be grouped into interfaces. Moreover, from this viewpoint the architect is better able to note redundant operations introduced to support different scenarios and to merge them.

Certain inconsistencies are more serious—outright violations of the underlying metamodel, for example. When this happens, the offending model elements are flagged with error indicators wherever they appear, and AWB provides Eclipse Quick Fixes to assist the user in clearing things up. These error indicators are visible in all viewpoints.

DESIGN AND IMPLEMENTATION OF AWB

In this section, we discuss the layered architecture of AWB, the use of models and metamodels, and the generation of work products.

AWB's layered architecture

AWB is built as a collection of plug-ins to the Eclipse platform. Eclipse provides much to build on: a platform-independent graphical user interface toolkit and application framework, a convenient and customizable way to manage panes, menu items, and controls, a powerful model of resources and builders, easy access to versioning and source control systems, a generic facility for error reporting and Quick Fixes, and so on. On top of Eclipse, AWB adds a core set of modeling facilities, views and editors for displaying and editing models, a query engine, a model refactoring engine, and search and document-generation facilities. These facilities, however, are mostly generic; that is, they are not particularly specialized to the domain of IT architecture, but rather to the task of building modeling tools which generally function in the way that we have described thus far. We call this core the "xWB" platform.

Above this level, a metamodel customizes xWB to the domain of IT architecture. The metamodel includes a specification of the allowable types of model elements, their attributes, and their relationships to each other. It specifies the viewpoints for that domain and the contents and layout of various editors. To help architects avoid the difficulty of designing on a "blank sheet," a metamodel can include starting points for a model. These starting points can include typical modeling constructs or predefined work-product templates, for example. The AWB metamodel has a starting point that includes templates for all of the key work products described previously. The metamodel consists of an XML schema for the key type definitions, with some supporting XML files to define relation type hierarchies, allowable relations, viewpoints (including their reminders), and editor behaviors.

Because richer domain-specific behavior is often needed, xWB defines extension points. This allows plug-ins to AWB to provide richer kinds of behavior, usually attached to specific kinds of model elements or metamodels. Diagram editors are examples of this, ranging from a simple diagram that shows nodes as boxes and relations as lines to styles of diagrams that are specific to system context and operational modeling.

In this way, AWB was designed with a "shearing layers"[14] approach, putting the most basic and common alterations at the simplest layers. Viewpoint definitions were one example of changes that needed to be simple and quick.

The first level of customization, then, is in those parts of the tool that are driven by model data. Work-product templates, some editor definitions, and user-defined reminders are at this level and are very rapidly customizable, even by fairly naïve users.

The second level of customization is at the metamodel level, particularly the peripheral aspects of the metamodel. This is where viewpoints, for instance, are defined and can be customized very quickly, but this requires understanding the more complicated viewpoint-specification language.

The third level of customization is at the heart of the metamodel itself. Metamodels have a more intricate structure than templates and queries, and, accord-

ingly, fewer people are able to customize them. Metamodel customization enables minor tweaks to the metamodel, for differing practices, or those made to accommodate major changes, such as entirely different domains. Examples of the latter include retargeting the workbench at requirements tracking, or even managing an antique-glass collection, as one author did; in effect, varying the "x" in xWB. It allows the creation of new types of nodes, editors, viewpoints, and so on. This is a mechanically simple task, but requires detailed analysis and understanding of the domain. At this level, a subject matter expert can quickly iterate and experiment with changes to the metamodel, as we did with our users in designing AWB.

A fourth level of customization involves Eclipse plug-in extensions. Their use enables the addition of a fairly arbitrary variety of gadgetry at a relatively high cost of programming. Our customized diagrams are examples of this. Alterations here required a more typical coding and debugging cycle before they could be released to our users.

Model and metamodel

The AWB internal representation of a model is simple enough; it is a directed graph of nodes, which are the model elements of interest (e.g., users, nonfunctional requirements, and use cases). Nodes are connected by relationships (e.g., a user initiates a use case). Both nodes and relationships have properties of scalar data (e.g., a user has a name).

Nodes and relationships also have types from a type hierarchy; for example, a nonfunctional requirement node may be qualitative or quantitative; the relationship "uses" is more specific than the relationship "depends on." The typing is advisory rather than strict; for example, a particular user might be given a nickname property as well as a name, or initiate a nonfunctional requirement, even if users ordinarily do not. This is a particularly powerful feature of AWB. Especially when refactoring is used, the model can be put into unusual states. Reminders and Quick Fixes then help users return the model to a normative state, as appropriate.

A metamodel describes the node and relationship type hierarchies, properties, editors, viewpoints, and so on. The most elaborate metamodels naturally concern IT architecture and requirements, but, as described previously, the core xWB platform is not tied to those topics; we also have metamodels for expressing metamodels themselves. (Indeed, a "metamodeler's workbench" built on xWB is the primary editor for our metamodels.) The metamodel is consulted by many parts of the workbench. The Relator wizard, for example, uses the metamodel to present a set of relationships to a user when relating two model elements. The markup and model facilities query the metamodel to see if semantic hyperlinking can be employed.

Work-product generation and queries

AWB can generate documents based on templates. Templates are written in a hybrid of HTML, Quetzal, and AWB-manipulation commands. For example, to create an alphabetical, numbered list of the labels of all user nodes, the template would include the following:

```
<ol>
  <for-each-node type="user" sorted-by="label">
       <li> <label/> </li>
  </for-each-node>
</ol>
```

The template language is reasonably simple, and it has been used extensively. After spending some time on an XQuery-based implementation of these features, we opted to develop a language more suited to AWB models.[15] Quetzal is a moderate-size language with strong resemblances to OCL2 and XQuery, designed with the relatively naïve user in mind. For example, to tell if some other node has the same label as node N, the following query can be used:

```
all.node.exists(it != N and it.label = N.label)
```

or, more verbosely,

```
FOR othernode IN all.node
WHEN othernode != N
SOMETIMES othernode.label = N.label
```

Substantial queries can be written to customize documents and behavior. Other aspects of the system are controlled by Quetzal queries or scripts. Viewpoints—especially the contents of Reminder panes—are defined in Quetzal. As mentioned, the user-defined reminders are written in Quetzal. Quetzal can also be used as a scripting language through a Command facility, which allows new

commands to be added to appropriate context menus in AWB (these menus launch Quetzal scripts).

The document generation facility, coupled with an embedded, wrappered browser (i.e., one that has a programming interface to allow it to be used as a component in Eclipse), is used to create editable work products. These editable work products are essentially "work products with a twist," viewed by our embedded browser. The document generation facility provides a twist (i.e., a clever device) in that it inserts additional HTML tags which include attributes ignored by the browser but crucial to AWB. These attributes identify the source in the model of the various document parts. Depending on where in the work product the user clicks, AWB can selectively place the browser into edit mode, track the user's changes, and push them back into the appropriate places in the model. Another twist is that for certain kinds of queries, such as iterations over lists of related model elements, additional markup includes embedded Quetzal commands telling AWB how to handle the addition and removal of elements of that list. This lets AWB augment the browser's context menu with commands to add and remove elements of the list, and then AWB can push the appropriate changes into the model. For example, looking at the use-case work product, the architect can add new or existing users to the list of actors involved in the use case, and AWB will create or destroy the appropriate relationships and elements in the model.

EXPERIENCE

The goal of the AWB project, as stated previously, was to design a tool that supports the realities of field-level practice, while supporting the formalities of a given architecture method. To validate our work and to drive its refinement, we launched a program of field trials in which IT architects used AWB as part of "live" customer projects.

There have been three major phases of our field trial program. First, IT architecture work was done with AWB in 2003 by an IT architect who was a member of the AWB team. His feedback was incorporated into the tool design immediately. Second, there was an initial beta program in 2004, where AWB was used more broadly within the IBM IT architecture community, mostly for requirements gathering, proposal response, and early solution design. Third, a pilot program was established in 2005 and is still ongoing. As part of this program, IT architects are using AWB on both the functional and operational aspects of solutions, as part of end-to-end architecture work on major customer projects.

In the following sections, we detail some of the experience, feedback, and lessons learned through our field trial programs. (Some of the feedback has already been presented earlier in this paper as part of the AWB design rationale.) In brief, feedback has been extremely positive, and to our great delight, the features of AWB are indeed proving to be highly effective in support of the creative process and work styles of practicing IT architects.

Initial requirements gathering

A number of architects have used AWB "live" during customer meetings, with their laptop display projected in place of a whiteboard. They used AWB to document brainstorming discussions, capturing initial thinking about the vision, scope, interfaces, and primary requirements for a system. The architects reported that the ability to take notes, mark them up "on the fly," form basic models, instantly generate system context diagrams, and update it all dynamically proved to be invaluable in keeping discussions focused and on track. They reported that the stakeholders were comfortable that all parties involved understood the scope of the solutions and the interfaces involved. Customers were impressed and very enthusiastic about the style of work that the new tool permitted.

The architects also reported that in follow-on work, linking proved very useful. If pieces of information or requirements seemed conflicting or seemed to require excessive design to satisfy, the architect could get back to the source of the information and perhaps request clarification or restatement of the requirement.

Proposal analysis

A consulting architect used AWB to conduct a quality assessment review of a 49-page proposal prepared in response to a 79-page customer Request for Proposal (RFP). He used AWB to capture requirements from the RFP, including the current IT environment, functional requirements, nonfunctional requirements, architectural guidelines, and required deliverables. In addition, he highlighted requirements that he suspected would be challenging to satisfy. He then used AWB to capture the

essence of the proposal, including use cases, components, nodes, and architectural decisions. He highlighted areas of concern with respect to technical viability.

The captured architectural information then served as the basis for his assessment of the quality of the proposal, including whether all components to be developed were included in the estimates, whether all COTS (commercial off-the-shelf) components were included in the price calculation, and whether all of the deliverables requested by the customer were included in the estimates. The architect used AWB to present his findings during a workshop with the proposal team.

He reported that AWB helped him quickly gain insight into both the RFP and the response. He made extensive use of the ability to link multiple text passages to a single model element as a way to isolate inconsistencies and to tie together a number of different references, often using different names, to the same system component.

Early stages of solution design

A consultant, working as part of a consortium preparing a proposal for a large government portal project, used AWB for the solution design phase. AWB was used "live" to capture the various architectural aspects emerging from workshop discussions. AWB allowed the consultant to quickly capture technological and use case requirements from all the project consortium partners (18 partners in total from different industries, with different backgrounds, and with different views of the term "architecture"). The consultant used AWB mainly to consolidate his notes and draft early versions of the architecture overview, actors, use cases, external collaborations, and system context. The consultant reported that AWB greatly eased his understanding of the solution and allowed him to "play" with different options to realize the architecture. He further reported that AWB proved extremely successful in bringing together all of the participants and bringing them all up to the same level of understanding.

Infrastructure redesign

In one pilot engagement, AWB was used for a large project involving infrastructure redesign. The customer was moving from a "thick client" approach to a "thin client" approach and was taking that situation as an opportunity to completely redo the architecture for 95 percent of the infrastructure of the system. The system consisted of hundreds of servers supporting 11,000 end users. The team using AWB reported that the architecture was developed in far less time and with far less effort than it would have taken using traditional tools, a savings of approximately 25 percent. The architects are now handing the architecture over to the development design authority—that is, the group that will own the architecture through the rest of the life cycle—and AWB is to become the repository of the architecture and the tool for its manipulation for the long term. One of the project principals remarked, "AWB is now truly a viable tool for generating and manipulating large complex architectures for real engagements."

Re-hosting and server consolidation

In another pilot engagement, IBM Global Services was contracted to move a customer's data center to an IBM hosting center. The existing customer system was a complex three-tiered system, with complex firewall rules and stringent high-security and high-availability requirements. As part of the re-hosting effort, the application and infrastructure architects used AWB to consolidate 38 Web applications, which allowed the number of servers for those Web applications to be reduced from 86 to 36. The consolidation work involved documenting functional application components and their dependencies, mapping them to conceptual server nodes, and realizing those nodes with the physical server nodes that became the actual servers. The architects additionally had to handle mapping of middleware components based on application requirements. As part of this work, they developed their own custom "server build" work product template, listing machine configurations, middleware requirements, and deployed application components. This work product was input into Excel** "build sheets," which were handed over to the server build team, who actually built and configured the servers according to these specifications.

The architects reported that as a result of using AWB, they saved 200–300 hours on this engagement. They further estimated that during subsequent server consolidation, having the AWB models as a starting point will save them 60 percent of the architecture effort, as compared with projects where no suitable architectural documentation exists, and 10 percent, as compared with projects where some

architectural documentation exists in Word and Visio. The architects indicated that AWB's consolidated view of the functional and operational models will allow them to maintain an accurate, up-to-date description of the architecture as it evolves.

Further, the architects reported that the excitement generated by the use of the tool was a catalyst in getting application and infrastructure architects to collaborate—a distinct benefit of using AWB. They also reported that with AWB they could instantly satisfy customer requests for summary diagrams of the IT environment. Before AWB, pulling together such diagrams was a daunting task. The filtering options of AWB diagrams and work products were also crucial, in that it was often necessary to produce diagrams of subsets of the environment. Finally, the architects reported, "Without AWB we'd still be documenting [this consolidation], or maybe we wouldn't be documenting it at all," and they indicated that AWB smoothed the handoff to the operations group that would run the system.

The consulting architect responsible for quality assurance (QA) on the project reported that use of AWB made QA much faster and more certain. The review of the architecture took roughly half the time that it would have taken otherwise. The QA architect reported that AWB helped the application and infrastructure architects follow GS Method more closely and use ADS more appropriately. The QA architect was delighted at how easy it was for him to navigate around the various models. The improved quality and easy understandability of the architectural models both likely contributed to the reduced QA review time. Overall, the QA architect reported that instead of the usual many calls to solution architects asking for missing information or for clarifications, for this review there were just three calls asking why a certain approach was taken or if something had not been considered, and AWB facilitated very effective communication. As for the learning curve inherent in any new tool, the QA architect remarked, "I was able to get up to speed and productive with AWB remarkably quickly."

Lessons learned

In this section, we present the lessons learned from working with a large number of IT architects on realistic industrial-scale architectures; namely, what issues are of major importance to the practitioner community and what resonates strongly with them. We hope these insights will help steer researchers toward areas that will have a lasting impact on practice.

User population

Most IT architects are under enormous time pressure. Thus, they will not adopt any tool that takes very long to learn or that takes longer to use in production work than the tools they are currently using. This is the case even if the new tool offers significant "downstream" benefits. AWB addresses this by trying to be as streamlined as possible for the work styles we observed.

Practitioners are constantly being bombarded by improved or extended metamodels, processes, methods, and best practices. Keeping up with this is daunting and overwhelming. Any tool that provides "roadmaps" or guidance in these areas is greatly appreciated. Although more detailed process-driven and technique-driven guidance would be a bonus, AWB reminders begin to address this issue.

Many IT architects are not computer science majors; instead they are business or IT majors (a number of them have remarked that a "graph" is something that shows the value of their stock portfolio over time). For these users, any tool that supports a document-editing or diagram-editing approach to modeling is far more effective for them than a tool that only supports more explicit graph manipulation (i.e., adding or deleting nodes and edges). AWB is beginning to address this with the modeling by forms approach. Further, allowing users to work at multiple levels of detail with respect to the metamodel has proven to be effective. For example, when working with GS Method and ADS, it is more effective for some users to view and manipulate models as components simply being placed on servers. Such users can disregard details of the component interfaces, operation signatures, deployment units, and so forth.

The context for IT architecture work

In the real world, "green field" systems, that is, those starting completely from scratch, are very rare. Tools simplifying the process of integrating with legacy systems would therefore provide enormous benefits. Such tools could support rapid reverse-engineering or import of models of legacy systems and highlight contact points between the system being developed and the external and legacy

systems with which it will be integrated. In addition, tool support for asset reuse and sharing between (or federation of) models is becoming increasingly important.

Architecture work never takes place in a vacuum—it is performed by teams which include the architecturally "uninitiated." Examples include both "upstream" customer subject matter experts and business analysts and "downstream" developers and operations staff. Tools for architects should therefore incorporate conventional team support for their models and integrate well with tools for others on the team. An integrated tool suite should allow for relatively seamless collaboration between team members. When material is exported to other tools and can be altered there, support is needed for "round-tripping"; that is, it should be possible to accept revisions to the material and fold the changes back into the artifacts being maintained by the architectural tool. This can be challenging because integration with opaque external tools can be difficult. More research into suitable techniques is needed.

Consulting architects often find that customers have standardized their own custom representation of architecture and specific work-product formats, and they insist that these be used. As a result, for an architectural tool to be acceptable it must be easily extensible and customizable. AWB's layered architecture supports this. Further, consultants are expected to deliver polished work products and reports suitable for many stakeholders. Any tool that produces anything less creates the burden of having to transfer the content into another tool for final polishing, with resulting consistency management issues.

An ideal in this context would be to provide effective support for asset reuse, which is regarded as a major key to success for large-scale architecture organizations. A facility that allows effective harvest, search, and reuse of assets would be a boon to architects, increasing productivity and improving the quality and consistency of delivered architectures. Effective asset reuse entails support not just in finding assets but also in using them; for example, helping to map parts of an existing model to proper "touch points" in the assets to be reused.

Leverage by automation

The feedback we have received from IT architects indicates that they are pleased with the extent to which AWB supports their style of doing the basic work of producing architectures and generating work products. At the same time, they tell us that now that AWB supports a full representation of architectures, they eagerly anticipate the leverage that could be obtained by automating many of the time-consuming tasks they currently do manually.

By keeping the generated work products up to date and consistent, AWB already saves its users an enormous amount of time and improves the quality of their work products. Other ways that AWB could save users time include automation of the following functions:

1. *Impact analysis based on traceability*—The links in AWB currently provide a measure of traceability, that is, a user can start from one artifact and follow links to other artifacts that somehow are related to, or gave rise to, that artifact. Given this traceability information, practitioners would greatly appreciate automated analysis of the impact of proposed changes to the architecture.
2. *NFR formalization, propagation, budget allocation, and aggregation*—If AWB were to allow for finer-grained and more detailed formalization of nonfunctional requirements, it would be possible for automated analysis to propagate NFRs from the point where they are specified (e.g., on the basic categories of interactions in the system context) to points further "downstream" in the architecture, for example, to use cases, scenarios, components, servers, and network connections. This would provide a basis for algorithms for load forecasting, performance analysis, and capacity planning.
3. *Scenario walkthroughs (for validation)*—Architects spend a great deal of time validating architectures by putting together interaction and sequence diagrams that trace progress through the system for a given scenario, including both scenarios where everything proceeds normally and scenarios where things like high-availability features come into play. With sufficient formalization, AWB could automatically produce diagrams and allow users to introduce faults or constraints to determine the effect on the interactions.
4. *Derivation of lower levels from higher levels*—Part of the modeling process involves producing progressively more elaborate and detailed specifications from initial higher-level specifications.

Given a representation of techniques, best practices, and heuristics, AWB could automatically produce starting points for lower-level specifications from higher-level specifications.

5. *Systems configuration*—Practitioners spend an inordinate amount of time configuring systems (hardware, middleware, and software), making technology choices, selecting compatible versions, and setting parameters. Given a knowledge base, AWB could automatically produce a starting point for systems configurations, thereby saving practitioners significant amounts of time and allowing them to produce configurations that are more likely to be complete and effective.

The preceding only scratches the surface of the kinds of work that could be automated once full architectural models are captured in a tool such as AWB.

FUTURE WORK

There are a number of areas for future work and innovation as AWB continues to evolve. Some of these were alluded to in the previous section. Those included modeling by forms, variable levels of metamodel and model detail, integration with other tools and providing for "round-tripping," extensibility of the metamodel by the end user, a convenient means of work product definition, and facilitating asset reuse and reference architectures. The previous section also described a number of areas for future work relating to various forms of automation.

Controlled experiments in which two teams conduct the same engagement in parallel are prohibitively expensive and often impossible when customer meetings and interviews are part of the process. We do hope to collect measurements from production engagements done using AWB, and to compare those measurements with benchmarks established for similar engagements. Until such studies are undertaken and completed, our results are primarily qualitative rather than quantitative, and we rely on feedback from highly experienced senior practitioners to determine the success of AWB at achieving its goals.

Additional areas for future work include team support; namely, providing an effective means by which one user can survey and understand the changes made to a model by another user before accepting the changes or merging other changes with them; and support for "stages" of an architecture; namely, being able to support a number of versions of an architecture concurrently, with promotion of model fragments from one stage to the next. Another area is that of enterprise architecture compliance; that is, providing support for verification that an architecture complies with an established enterprise architecture.

CONCLUSION

AWB is a tool for IT architects to gather, structure, and maintain the information that constitutes the architecture of an IT system (or collection of IT systems). It emphasizes evolution from partial, informal, overlapping, and inconsistent information into precise formal models that constitute "actionable" specifications. With AWB, users maintain one composite model in a central repository and then generate many formatted work products as reports that are always accurate and up to date.

AWB innovations and features (to date) include: opportunistic modeling, complementary textual hierarchical and diagrammatic model manipulation, viewpoints, reminders, model refactoring, work product modeling, generation, and direct editing and GS Method operational-modeling support.

AWB has been used in production engagements by IT architects, and the response has been very positive. Work is now proceeding on generation of artifacts that feed into downstream solution development activities, as well as closer integration with tools for development, deployment, and testing. In addition, work is underway on using AWB to produce reference architectures as reusable assets for a number of the industries served by IBM. AWB has proven to be highly effective, even in these early field trials, and it will continue to evolve even as it is deployed ever more broadly.

ACKNOWLEDGMENTS

We gratefully acknowledge the support and collaboration of a number of our colleagues, including Daniel Yellin (as reviewer and mentor), Francoise Legoues (our IGS partner), and ADS leaders and developers Ian Charters, Ed Kahan, Philippe Spaas, and Carl Spencer. We would also like to thank our many beta and pilot users, especially Matt Bloom, Thorsten Gau, Scott McCauley, Kevin Robson, Christian Schiller, and Marc Walford.

*Trademark, service mark, or registered trademark of International Business Machines Corporation.

**Trademark, service mark, or registered trademark of Object Management Group, Inc., Telelogic AB, Microsoft Corporation, or the Eclipse Foundation in the United States, other countries, or both.

CITED REFERENCES

1. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, Reading, MA (1998).
2. IEEE Std 1471–2000 IEEE Recommended Practice for Architectural Description of Software-Intensive Systems—Description, http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html.
3. P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software* **12**, No. 6 (1995).
4. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA (1999).
5. R. Youngs, D. Redmond-Pyle, P. Spaas, and E. Kahan, "A Standard for Architecture Description," *IBM Systems Journal* **38**, No. 1, 32–50 (1999).
6. Object Technology International, Inc., "Eclipse Platform Technical Overview," http://www.eclipse.org/whitepapers/eclipse-overview.pdf.
7. S. Abrams, J. Smith, R. Bellofatto, R. Fuhrer, D. Oppenheim, J. Wright, R. Boulanger, N. Leonard, D. Mash, and M. Rendish, "QSketcher: An Environment for Composing Music for Film," *Proceedings of the Fourth Conference on Creativity and Cognition*, Loughborough University, U.K., 2002, ACM Press, New York, pp. 157–164.
8. T. C. Lethbridge and D. Skuce, "Beyond Hypertext: Knowledge Management for Technical Documentation," *Proceedings of the 10th ACM Annual International Conference on Systems Documentation* ACM Press, New York (November 1992), pp. 313–322.
9. D. Skuce and T. C. Lethbridge, "CODE4: A Unified System for Managing Conceptual Knowledge," *International Journal of Human Computer Studies* **42**, 413–451 (1995).
10. V. Bush, "As We May Think," *Atlantic Monthly* (July 1945).
11. D. C. Engelbart and W. K. English, "A Research Center for Augmenting Human Intellect," *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference* (December 1968), pp. 395–410, http://bootstrap.org/augdocs/friedewald030402/researchcenter1968/ResearchCenter1968.html.
12. R. Racko, "A Cool Tool Tool," *Software Development Magazine* (May 2004), http://www.jaczone.com/papers/05sd.Racko21-26.pdf.
13. R. Balzer, "Tolerating Inconsistency," *Proceedings of the 13th IEEE International Conference on Software Engineering*, IEEE Computer Society Press, Austin, Texas (May 1991), pp. 158–165.
14. S. Brand, *How Buildings Learn: What Happens After They're Built*, Penguin Group, Canada (1995).
15. B. Bloom, "Lopsided Little Languages: Experience with XQuery," *Proceedings of the Second International Workshop on XQuery Implementation, Experience and Perspectives* (2005).

Accepted for publication March 6, 2006.
Published online July 12, 2006.

Steven Abrams
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (sabrams@us.ibm.com). Dr. Abrams is a research staff member in the Software Technology department of IBM Research managing the Business Application Modeling group. With that team, he develops tools that help people understand, describe, architect, visualize, and validate enterprise applications more easily and naturally than possible with traditional tools. He has had a varied career in fields such as computer music, robotics, computational geometry, and CAD/CAM and rapid prototyping tools. Dr. Abrams recently served on the National Academy of Sciences committee on Information Technology and Creativity. He studied at Columbia University where he earned B.S., M.S., and Ph.D. degrees in computer science.

Bard Bloom
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (bardb@us.ibm.com). Dr. Bloom received a Ph.D. degree from the Massachusetts Institute of Technology in 1989, taught at Cornell University until 1995, and has worked at the Watson Research Center since then.

Paul Keyser
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (pkeyser@us.ibm.com). Dr. Keyser studied physics and classics at St. Andrews' School, Duke University, and the University of Colorado at Boulder. After a few years of research and teaching in classics at the University of Alberta at Edmonton, Cornell University, and other places, he returned to his first love, programming. He is currently crafting Java™ and Eclipse™ plug-ins for the Watson Research Center in the Semantic Analysis area. His publications include work on gravitational physics, stylometry, and ancient science and technology.

Doug Kimelman
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (dnk@us.ibm.com). Dr. Kimelman is a research staff member at the Watson Research Center. His general interests include the structure and behavior of complex systems, and "human computer bandwidth." Specific areas in which he has worked include: parallel systems, operating systems, debuggers, performance tools, software visualization, software development environments, and technologies for specifying and implementing large-scale IT systems.

Eric Nelson
IBM Software Group, 3100 Smoketree Court, Raleigh, North Carolina 27604-1054 (ericnels@us.ibm.com). Dr. Nelson is a Certified Senior IT Architect in the Federal CTO Strategic Technology Architecture team. He received a Ph.D. degree in cognitive psychology from the University of Chicago in 1987, where he worked for a time as the university's technical liaison to the National Science Foundation (NSF) supercomputer centers, assisting faculty and students in the design and implementation of their research projects for NSF systems and the Argonne National Laboratory parallel-computing facility. While at Chicago, he led the development of a PC-based population dynamics simulation system that was part of a multi-university biological-sciences laboratory-simulation collection called BioQuest, which won an EDUCOM innovation award in 1991. In 1992, he moved to Tokyo, where he taught psychology and computer science and

was a software consultant in financial services. He continued consulting in Singapore in both financial services and modeling and simulation for games and real-time training systems. Upon returning to the United States in 1997, he joined IBM Global Services, where he was part of the Enterprise Architecture and Technology Center of Excellence, working with a broad range of Fortune 100 clients in financial services, distribution, retail, and media. In 2002, he joined the Federal CTO Strategic Technology Architecture team and now works with Department of Defense and civilian agencies on technology strategy and transformation to open architectural systems and service-oriented architectures.

Wendy Neuberger
IBM Sales and Distribution, 609 Harvard Street, Vestal, New York 13850 (wneuberg@us.ibm.com). Ms. Neuberger is a Senior Technical Staff Member and Certified IT Architect in IBM Sales and Distribution, Distribution Sector. She is responsible for the development and deployment of retail reference architecture that integrates industry solutions, best practices, and tooling. Prior to joining Sales and Distribution, she was in IBM Global Services, Application Management Services (AMS). Ms. Neuberger was a member of the AMS Research Institute, championing several technology innovation projects, including Architects' Workbench. She received a B.S. degree in business management and an M.B.A. degree in management information systems from Binghamton University and joined IBM in 1983.

Tova Roth
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (tova@us.ibm.com). Ms. Roth is an advisory software engineer at the Watson Research Center. In addition to the Architects' Workbench, her work at IBM Research has included visualization of complex systems, optimization of dynamic systems, debugging tools, performance tools, and tools for aspect-oriented development. Prior to joining IBM Research, Ms. Roth worked on visualization of financial data and computer-graphics tools.

Ian Simmonds
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (simmonds@us.ibm.com). Mr. Simmonds is an advisory software engineer in the Business Application Modeling department at the Watson Research Center. He received a B.A. degree in mathematics from Cambridge University in 1987. Prior to joining IBM, he developed PCTE-based software engineering tools for ESPRIT's PACT and ATMOSPHERE projects and EUREKA's EAST Environment and contributed to the standardization of ECMA and ISO PCTE. He joined IBM in 1993 and since 1995, has researched methods and tools for business and IT consulting (specifically for the insurance industry), requirements management, systems envisioning, and (starting in 2002) IT and enterprise architecture.

Steven Tang
IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York, 10532 (stang@us.ibm.com). Dr. Tang is a software engineer at the Watson Research Center. His major focus areas are advanced user interface (UI) interaction techniques, frameworks, and graphical and rich text clients. Specifically, he led the development of a query-driven browser-based UI that supports modeling through declaratively specified forms. Dr. Tang received both an M.S. degree and a Ph.D. degree in electrical engineering from Stanford University. He published several papers on UI builders and frameworks at the UIST conference between 1991 and 1994. Dr. Tang joined IBM Research in 2003, where he has been applying his UI expertise on tooling support for modeling applications. Prior to joining IBM, Dr. Tang worked for Fujitsu Network Communications as a senior manager for UI development, and he also co-founded a company in 1999 that specialized in rich Ajax applications supporting incremental two-way updates.

John Vlissides
Dr. Vlissides was an IBM research staff member from 1991 and a member of the IBM Academy of Technology from 1998. He was best known for his part in creating the field of software patterns and for his first book, *Design Patterns*, coauthored with Gamma, Helm, and Johnson (known in the field as the "Gang of Four" or "GoF"). He and the Gang of Four were recently awarded the 2005 ACM SIGPLAN Programming Languages Achievement Award for their work on design patterns. Dr. Vlissides' research interests were in software design tools and techniques (especially object-oriented ones), design patterns, application frameworks and builders, software visualization, and tools for user interface development. He received a B.S. degree in electrical engineering from the University of Virginia and M.S. and Ph.D. degrees from Stanford University. Dr. Vlissides passed away in November, 2005, after a battle of more than a year and a half with a brain tumor. ■