Automated capture and retrieval of architectural rationale

H. Richter, P. Schuchhard, and G.D. Abowd
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{hrichter,pascal,abowd}@cc.gatech.edu

Abstract

The Software Architecture Analysis Method (SAAM) was developed at the Software Engineering Institute in the mid-90's and has been effective in helping industrial projects to uncover a shared understanding of the high-level organization of large software systems as well as to reveal how that structure is impacted by suggested changes to system requirements. In the MORALE project at Georgia Tech, we are further investigating the potential for a process such as SAAM to capture the architectural rationale of an evolving software project. One of the features of SAAM is that it is a people-oriented process. Yet this creates difficulties in keeping track of the very rich discussions on system organization and change. In this paper, we present the use of ubiquitous computing technology to augment the conventional SAAM process through automated capture, integration and access to the architectural discussions and artifacts produced by a SAAM session.

Keywords

architectural analysis, SAAM, ubiquitous computing, automated capture

1 INTRODUCTION

Software evolution is a costly and time consuming software development activity. Effective system evolution requires understanding both the way that an existing system accomplishes its tasks and the mission-oriented rationale for any changes that feed its evolution. The MORALE project at Georgia Tech addresses the problem of designing and evolving complex software systems. The MORALE acronym summarizes its goals:

- Mission ORiented: We want the legacy system enhancement process to be driven by the mission to be accomplished rather than by purely technical criteria.
- Architectural: The most time consuming and costly alterations to software are those that distort architecture, by which we mean its structure and behavior. We want to provide a mechanism for predicting the impact of architectural changes so that the risks of making those changes can be ascertained early in the evolution process.
- Legacy Evolution: We are concerned with the evolution of legacy systems. We want to provide a cost effective way of analyzing existing software, and once analyzed, extracting those parts of it which can be used in the new version.

MORALE addresses these goals by integrating several different technologies: reverse engineering to extract and visualize architectural information (Jerding 1995); requirements analysis to structure and understand an organization's changing mission-oriented goals (Potts 1994); and finally software architecture impact analysis to determine how requirements changes affect the high-level organization of a software system.

This paper is focuses on a particular process in support of the latter technique. Specifically, we make use of the Software Architecture Analysis Method (SAAM) developed at the Software Engineering Institute in the mid-90's (Bass 1998, Kazman 1996 and Kazman 1994). SAAM is a scenario-based method that can help extract how changing requirements will impact an already existing software system. In this context, we can understand the entire SAAM process as a design rationale technique that is centered on software architecture. We therefore refer to SAAM as an *architectural rationale* technique.

The SAAM process, summarized in Section 3 below, is a very people-oriented technique, and as such, has advantages and disadvantages. The main advantage is that it is a practical and useful technique for extracting a common understanding of the high-level software organization of a complex system. This is particularly important in that software architecture is rarely properly documented for the benefit of future designers. The other important advantage of SAAM is that it makes explicit how intended changes will impact the existing system, oftentimes helping an organization avoid undesirable development efforts. One disadvantage of SAAM is that it assumes that designers are able to describe the existing organization of the software, even when the original architects are not available. Secondly, SAAM requires additional time and effort to produce a coherent written summarization of the results of the analysis.

One goal of the MORALE project is to address the first disadvantage of SAAM by integrating with visualization techniques in reverse engineering to uncover the architecture of an existing system. In this paper, we are concerned with how ubiquitous computing technology might address the second disadvantage. At Georgia Tech, we have a lot of experience in building environments that capture live experiences in order to make them available for later review and summarization. Most of our experience in this area has been applied to the education domain, in a project entitled Classroom 2000 (Abowd 1998, Abowd 1998b, and Abowd 1996). A typical SAAM session is a live event involving discussions by 3-10 designers, managers and facilitators that is centered around drawings of the architecture on a public display. By converting the public display to an electronic whiteboard surface and recording the discussion with digital streaming media technology, we can automatically capture the SAAM sessions and then provide the ability to salvage summary information afterwards. By allowing the ubiquitous computing infrastructure to do what it does best, record and capture activity, we can free the humans to do what they do best, synthesize and understand activity in a technical discussion.

We will present an initial prototype built to investigate how automated capture tools can support an architectural rationale technique such as SAAM. The prototype, called SAAMPad, is a meeting room environment centered around a large electronic whiteboard. We will demonstrate how this prototype environment has been shaped to support naturally defining a software architecture and capturing scenario-based impact analyses during a SAAM session.

Overview of Paper

In Section 2 we provide background on design rationale and automated capture environments. In Section 3 we further describe the SAAM process in the context of evolution and architectural rationale. In Section 4 we describe the SAAMPad prototype and demonstrate its use on a canonical and simple architectural case study, the Key Word in Context (KWIC) system. Finally, we conclude with our future plans for rationale capture of SAAM.

2 BACKGROUND AND RELATED WORK

We have introduced the SAAM process as a design rationale technique centered on architecture. In the following section we discuss design rationale in general as well as specific techniques that help to capture or structure the rationale behind some artifact. Next, we look at the more general problem of capture of live experiences as a paradigm for multimedia authoring and retrieval.

2.1 Design Rationale

Design rationale is the explanation behind the design - why the design is the way it is. Rationale can include assumptions made about the system, the alternatives considered, and the reasoning behind decisions. Often this rationale is contained only in the minds of the developers of the system. Yet this undocumented information could help system evolution by exposing previous decision points, alternatives, and assumptions that are vital to efficiently changing the software. Rationale capture, however, can require additional effort to document and organize information alongside the design.

Traditional approaches to rationale capture have involved two methods for documenting and structuring design rationale: process-oriented and structure-oriented. A process-oriented approach, such as Issue Based Information Systems (IBIS) (Rittel 1973), focuses on documenting rationale as it occurs during design meetings. A structure-oriented approach, such as the Questions, Options, and Criteria (QOC) notation (Shum 1994), focuses instead on a post hoc structuring of the rationale to show the complete design argument. Neither approach appears to have been widely accepted in the software community.

One weakness of process-oriented techniques is that they can disrupt and, therefore, alter the actual design activity that they are trying to support. Conklin et al. (1991) attempted to reduce this overhead by capturing a rationale trace using the IBIS method with less disruption to the design process. They developed a graphical tool, called gIBIS, and an indented textual tool for capturing and managing the network of issues of IBIS and tested the tools successfully on an industrial project. It is our intent to create a tool that is process-oriented, capturing rationale as it occurs naturally, but without the need for additional rationale capture personnel at the design meeting. Simply by capturing a structured discussion such as a SAAM evaluation session, we will be able to record important design rationale as it happens.

Architectural rationale is a more specific design rationale, referring to the reasoning behind the software architecture, or high-level system organization. One project looking specifically at architectural rational is the WinWin project (Boehm 1994 and Bose 1995). The WinWin approach involves stakeholders collaborating to negotiate win conditions, tradeoffs, and issues during the requirements and high-level design phases of development. Extensions to this requirements framework provide explicit schemas for recording the rationale and linking it to the architecture. The design rationale model is constructed of win conditions, issues raised involving those conditions, options suggested to address the issues and agreements adopting certain options. Additionally, the rationale can consist of reasons and constraints on these elements. These rationale elements serve as the basis for the collaboration and direct the discussion among the stakeholders. Using the WinWin support system, users could enter win conditions and propose issues and options for discussion, allowing the rationale model to be constructed actively by the system during the negotiation discussion.

While both gIBIS and WinWin attempt to reduce the overhead in capturing rationale, they focus on particular elements that must still be formally documented during the discussions. While structure-oriented techniques do not interrupt design discussions, they risk losing critical information by waiting until after the fact to record rationale. A nice balance could be achieved if there was a way to capture the rationale as it occurs, but without introducing interruptions to the design process. In this next subsection, we discuss the general problem of capture that aims to provide a ubiquitous yet unobtrusive service.

2.2 Capture

One of the potential features of a ubiquitous computing environment is that it could be used to record our everyday experiences and make that record available for later use. Indeed, we spend much time listening to and recording, more or less accurately, the events that surround us, only to have that one important piece of information elude us when we most need it. We can view many of the interactive experiences of our lives as generators of rich multimedia content. A general challenge in ubiquitous computing is to provide automated tools to support the capture, integration and access of this multimedia record. The purpose of this automated support is to have computers do what they do best, record an event, in order to free humans to do what they do best, attend to, synthesize, and understand what is happening around them, all with full confidence that the specific details will be available for later perusal.

The Classroom 2000 project is mainly concerned with capture, integration and access in support of lecture based education (Abowd 1998, Abowd 1998b, Abowd 1996). The many streams of activity in a typical lecture ---what is spoken, what is seen, what is written down on a whiteboard and what is shown on public displays--- are combined to provide a rich interactive experience that is becoming increasingly more difficult to capture using traditional pen and paper notes.

Other research teams have used this same notion of capture, integration, and access to facilitate collaborative or personal experiences. Work at Xerox PARC focused on capturing technical meetings to support summarization by a single scribe who was often not well versed in the subject of the meetings (Minneman 1995, Moran 1997). More work at PARC (the Marquee system, Weber 1994) together with work at Hewlett-Packard (the Filochat system, Whittaker 1994), Apple (Degen 1992), and MIT's Media Lab (Stifelman 1996) demonstrates the utility of personal note-taking with automatic audio enhancement for later review.

The challenge in applying ubiquitous technology to the SAAM process is to provide an organization of multimedia streams that facilitates access to the architectural rationale. This requires going beyond simple record and playback schemes to providing meaning to the various activities and records of a SAAM session. Since SAAM is a structured process, as explained in the next section, our task is made simpler.

3 SAAM

The Software Architectural Analysis Method, or SAAM, is a structured method for understanding the high-level organization of a software system and determining the impact of requirements changes on that structure. SAAM was developed at the Software Engineering Institute (SEI) to enable software developers to compare different proposed architectures based on how they would be impacted by current and future requirements of the system. However, the same technique may be used to determine how an existing system may be affected by evolution. The SAAM method revolves around group discussions by the various stakeholders in the system, including designers, customers, and users.

Software architecture in SAAM refers to the components into which a system is divided at a gross level of system organization, and the ways in which those components behave, communicate, interact, and coordinate with each other. Components may be processes, tasks, or classes. These components may communicate via system calls, message passing, or event broadcasting. Communications are sometimes referred to as connectors. SAAM attempts to determine the quality of the architecture, or in the case of evolution, the impact of the evolution on the design.

Quality of a system can only be measured with respect to some attribute. SAAM uses scenarios to express particular quality attributes or new system behaviors. The analysis team then discusses how well or how easily the architectural design satisfies the demands placed on it by each scenario. SAAM's scenarios are brief descriptions of some anticipated or desired use of a system. One example might be "users can change the color of the window borders." Scenarios can differ widely in breadth and scope. Scenarios should also include all the different roles involved in a system, such as the user, the operator, and parts of the software.

The following steps outline the process and products of a SAAM evaluation used in the context of MORALE:

1. Describe existing architecture. The architectural descriptions need to be understandable by all parties involved in the analysis. They need to include the system's computation and data components, as well as all the connectors.
2. Develop scenarios. Develop task scenarios that illustrate the kinds of activities the evolved system must support and the kinds of anticipated change to the system
3. Perform scenario evaluations. For each scenario, determine whether the architecture can execute it directly, or whether a change would be required to execute it (which we refer to as an indirect scenario.) For each indirect scenario, list the changes to the architecture that are necessary for it to support the scenario and estimate the cost of performing the change. A modification to the architecture means that either a new component or connection is introduced or an existing component or connection requires a change in its specification.
4. Summarize the information. Summaries could include tables of components and scenarios and the set of changes required for each, or diagrams highlighting changed components.

4 SAAMPAD

During a SAAM session a great deal of architectural rationale can be discussed. Constraints and assumptions may be raised while understanding the original architecture. Scenario evaluation may involve considering a number of solutions and making tradeoffs. Yet users trying to take detailed notes of all of this information are not likely to fully participate in the discussions. Additionally, it is hard to know exactly how important certain items are during discussion so that they may be documented more fully. Instead, they only become important at the time they are actually needed. As such, we propose a new way of automatically capturing the entire experience of a SAAM session and allowing easy access to the information later.

Our biggest challenge in providing automated capture support is to enhance the SAAM process *without* changing the way the method currently works. The capture tool we propose must (1) allow the users to concentrate on the SAAM process and not on the capture, (2) provide summary information of the SAAM activities, and (3) provide users with meaningful ways to access all of the captured information.

The approach we used to achieve these goals was to start with a general capture tool and extend it to support SAAM specific activities. We used Zenpad, the central tool in the Classroom 2000 system, as a basis. Zenpad supports the capture of audio and video streams as well as annotations made on an electronic whiteboard. The SAAMPad extensions involve providing specific behavior to support SAAM activities, such as drawing an architecture and marking changes, and supporting different behaviors for the different phases of the SAAM process. Additionally, we created visualizations of SAAM sessions to facilitate the review and retrieval of the recorded information. We first describe an example to help illustrate the tool. Then we step through how users will interact with SAAMPad during the different stages of the SAAM process.

KWIC

To facilitate further discussions of SAAMPad, we introduce a simple case study that we performed. We used the Key Word In Context System (KWIC) described by Parnas (1972) as follows:

> The KWIC index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters. Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

There have been several proposed architectural solutions for this system. For our purposes, we needed to pick one solution as an "initial" architecture that could then be evolved based on some specific change criteria. The initial architecture we used was based on a shared-memory model. We then performed a SAAM evaluation for the proposed changes listed in chapter 9.3 of Bass et al. (1998). Examples taken from this case study will be used throughout the rest of the paper.

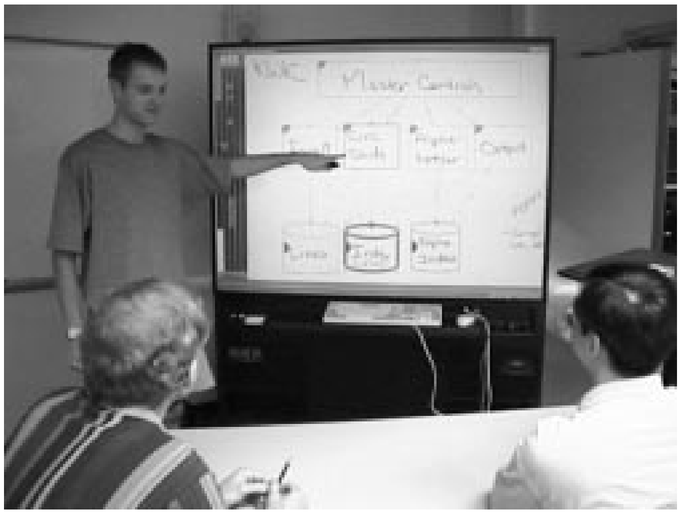
Figure 1. The SAAMPad environment.

4.1 Using SAAMPad

Participants in the SAAM session gather around an electronic whiteboard in a room capable of recording audio or video. Figure 1 is a picture taken of someone using SAAMPad in such an environment. The participants then proceed to follow the steps outlined in the SAAM method. SAAMPad adjusts its behavior to support each particular SAAM step. SAAMPad's functionality is described for each of the SAAM stages below.

1. Describe the existing architecture. We refer to this as the architecture generation phase. The team of evaluators draws the architecture onto the electronic whiteboard, much as they would draw upon a traditional whiteboard. We created a simple gesture-based interface that would allow for a clean box-and-line drawing. A gesture in SAAMPad is simply a penstroke made with the provided electronic pen. A designer begins to draw a box on the whiteboard and it is quickly recognized as a component that can then be sized. Different types of components (e.g., processes, data repositories) are typically distinguished by different shapes. Once a component is created by a simple gesture, a designer can change it to another pre-defined type by tapping on a control point on the component. A name can be associated to the component by writing inside the component boundary. A gesture from one component to another creates a connector. Portions of the diagram can be selected and moved on the electronic surface. SAAMPad supports hierarchical architectural descriptions by allowing the designer to "open up" a single component and draw a sub-architecture within it. The result of this generation phase is shown for the KWIC system in Figure 2. What is important to note here is that all of the activity in generation is captured. Discussion is recorded

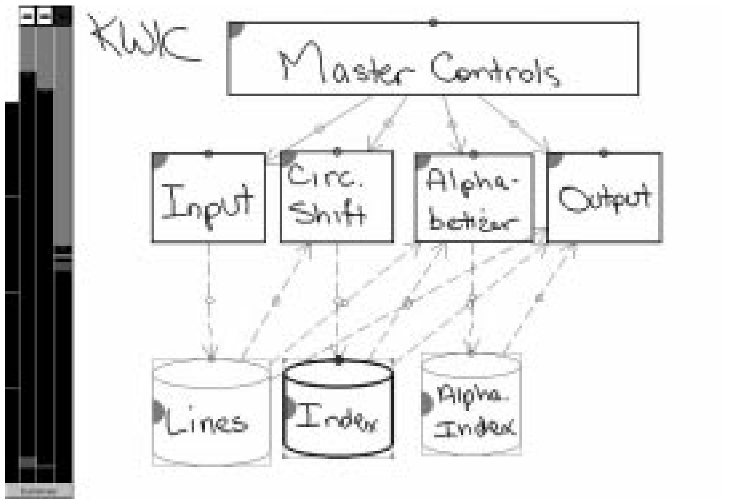

Figure 2. The initial architecture for the KWIC system

and the times associated with various actions on the whiteboard surface (all activity with the pen) is timestamped to allow us to later index into the discussion.

2. Develop scenarios. The designers must document the concrete scenarios that will be the basis for the SAAM evaluation on the initial architecture. SAAMPad provides a simple interface to record brief names associated with scenarios as well as explanatory text. Though this part of the process is not explicitly captured, it would be a simple and potentially useful task to record the scenario brainstorming session.

3. Perform scenario evaluations. Once the initial architecture is generated and the set of scenarios is created, the designers can evaluate the architecture against each scenario. For each scenario, the initial architecture is displayed in black. As the scenario is discussed, designers can point to parts of the architecture. Pointing causes that portion of the architecture (most often a component) to be highlighted, announcing to the audience and to SAAMPad what part of the architecture is being discussed. This information is timestamped and used later to visualize the results of the scenario. If the designers decide that a particular architectural element is to be altered to address the needs of the current scenario, then a simple tap with the pen on that element marks it as changed, and changes its color from black to red. New components or connectors can be added to the system using the same gestures as in the generation phase, and will show up in blue on the board. Elements to be deleted are grayed out. Any additional information that is deemed relevant can be handwritten on the board as well.

4. Summarize the information. The purpose of the previous steps, from SAAMPad's perspective, is to capture relevant timestamps associated to important SAAM activities. The timestamped activity is associated to parts of the architecture, meaning that this information can be used to provide a number of automatically generated summaries and visualizations of the architecture that will help designers understand the overall impact of the SAAM session. Designers can view these visualizations for a quick overview of the impact of the SAAM session, and also use them to index into the other multimedia streams (e.g., the recorded audio and video of the session) to find more details. Figure 3 shows an example visualization of a KWIC scenario evaluation. This visualization is further explained in the following section.

4.2 Visualization

In order to facilitate the retrieval of architectural rationale, we need to organize the captured information around meaningful structures of the SAAM process. Architectural diagrams act as the center of discussion, provide a visual representation of the system, and indicate the effects that scenarios have. As such, these diagrams serve as the central focus of the capture and access of the rationale. We chose to use gesture recognition to detect architectural elements, enabling SAAMPad to recognize events surrounding the architecture without disrupting the users' discussion. We then organized the visualization around the architecture and these events. Additionally, the SAAM process is composed of different phases, namely architecture generation, scenario generation, and scenario evaluation. SAAMPad supports these different phases with tailored capture and visualizations.

The events that SAAMPad currently recognizes are creation, change, deletion, and focus of attention. We use color to represent creation, change, and deletion events on the architecture. Components or connectors that are created in the current phase are represented in blue. Elements that are marked as changed are represented in red. Deleted elements are grayed out. Elements that are present from a previous phase are represented in black. This use of color provides meaning to the participants about the status of the architectural elements during a live SAAM session as well as provides a quick way to summarize the activity of a particular session just by looking at the architectural diagram. For example, in Figure 3 we can see that the "Circ. Shift", "Alphatetizer", "Output", "Index Lines", and "Alpha. Index" components required change for this scenario, as well as the connectors leaving the "Index Lines" component. This color scheme does limit expressiveness by not allowing users to create their own color schemes or expand upon ours. However, by using this standard, SAAMPad can accurately determine the color of each object during capture and provide a standard, easy to understand visualization.

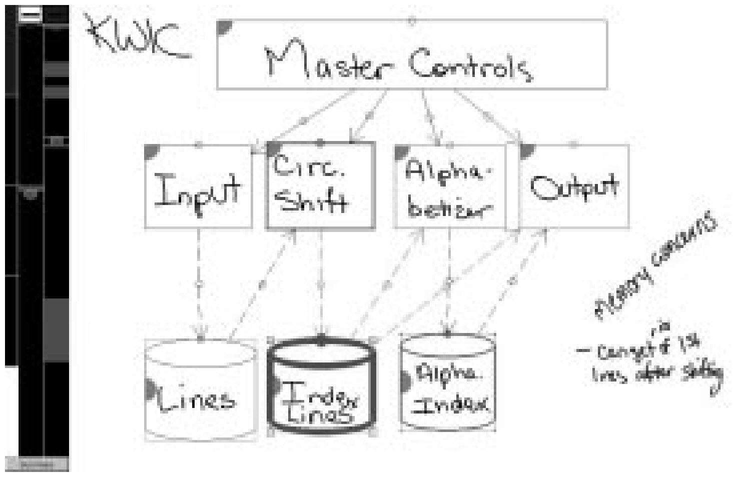

Figure 3. Visualization of a scenario evaluation of the KWIC system. The timeline on the far left represents the entire SAAM session, while the other timelines show the activities for two specific components.

The discussions surrounding the architectural diagrams are often non-linear and sometimes seem chaotic. Without knowing what was the focus of attention (FoA) at any time during the session, users have a hard time understanding what is happening in a scenario evaluation. Therefore, in order to capture a scenario successfully, we need a way to track the FoA. We noticed that in traditional sessions, users point to parts of the diagram to indicate the subject being talked about. The electronic whiteboard we used uses special IR-pens for input. Within a reasonable distance from the board, these pens also work as pointing devices. We added the possibility for the user to highlight one or more components in a diagram, thus indicating what the FoA is at that moment. This is a very hardware-specific solution and it will be hard to adjust this technique to other input devices. However, it does prevent the users to make a big adjustment in the way they perform SAAM. Future research will attempt to use other computational perception techniques to elicit the FoA.

When trying to visualize an entire recorded SAAM evaluation, we want to show all of these events in a way that would support effective browsing by the designers. The solution we chose was to represent the entire SAAM evaluation along a timeline that explicitly delimited the generation phase and the individual scenario evaluation sessions. In Figures 2 and 3, the timeline for the whole KWIC SAAM evaluation is shown on the extreme left of the screen as a simple image broken into a number of sections. The top section indicates the generation phase of the SAAM evaluation. Tapping on that portion of the screen reveals a timeline area for the generation phase in the adjacent portion of the screen as well as the architectural diagram in the main portion. This phase-specific timeline can show information for any of the architectural elements shown. For example, tapping on the component labeled "Input" in Figure 2 would reveal a timeline of events in the generation phase that are related to that component, indicating when in the phase that component was created and when it was the focus of attention of the discussion. The phase-specific timeline is interactive. Tapping it at various places will launch an audio or video player at that point in the recorded session. We have used RealNetworks streaming media for this purpose. The architectural diagram itself is interactive, so tapping various parts of the diagram will also launch the audio or video at the point when that element was created (during generation phase) or altered (during a scenario evaluation phase).

Besides this timeline visualization, we are also exploring ways of visualizing an entire SAAM session, instead of one phase or evaluation at a time. We have implemented a version of a 'fisheye' diagram, showing the components and connectors in the initial diagram with the line width representing the frequency that they change over all of the scenario evaluations. This gives a quick indication of which are most affected by the scenarios. Another view we have implemented is a simple textual table of the architectural components and the scenarios in which they are created or require change. These visualizations provide compact summaries of the overall results of the SAAM evaluation.

5 CONCLUSIONS AND FUTURE WORK

SAAM is a people-oriented architectural analysis method for extracting how changing requirements impact an existing software system. SAAM discussions can be rich in reasoning about the architecture and the impact of changes. Yet trying to capture this information places a burden on those performing a SAAM evaluation. We introduced SAAMPad, a tool for the automated capture and retrieval of architectural rationale surrounding SAAM. SAAMPad utilizes ubiquitous computing technology extended with architectural semantics to achieve this goal while preserving the SAAM process. One potential drawback of this tool is that it concentrates only on supporting the SAAM process and might be less suited when developers want to try a new way of evaluating systems. However, by concentrating on a single method, we were able to provide more meaningful ways of capturing and accessing the rationale.

SAAMPad has so far only been evaluated on a simple case study of the KWIC system. The results from this case study look promising, but their value is limited as a more complex system will likely produce new insights. The KWIC case study verified that the capture of the scenarios was easy and the visualizations were automatically generated. The color scheme was particularly useful in providing a quick indication of what happened during a scenario evaluation. The timelines allowed searching for detailed information but were a bit awkward to use due to the fact that their relationship with their components is currently not very clear. We plan to perform additional case studies on larger, real-world systems. Additionally, we plan to investigate more visualization techniques to improve the access to the architectural rationale and the automated summaries that are created.

We hope that the additional case studies will help us identify specific areas of improvement.

We would also like to take advantage of recognition technologies so that the system can better understand the content of information being captured and facilitate more useful retrieval. For example, voice and handwriting recognition could be used to allow the user to perform a text-based search through the scenarios. We have experimented with these technologies in the larger Classroom 2000 project. We can also think of other uses for this; for example, making the system aware what kind of information is discussed at any moment in the session, can help determine the architectural focus of attention. Better semantic capture will enhance the summary reports, since components could then be referenced by their name.

Besides session summaries, there is no way to communicate with other software architecture tools. We intend to use ACME (Garlan 1997) for this purpose. ACME is an extensible inter-communication language for architectural tools. Its extensibility allows us to add specific information to components that other tools can then use. By making SAAMPad ACME aware, we can also import existing architectural descriptions that would seed a SAAM evaluation.

6 ACKNOWLEDGEMENTS

This effort is sponsored by the Defense Advanced Research Projects Agency, and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-96-2-0229. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory, or the U.S. Government.

7 REFERENCES

Abowd, G.D., Brotherton, J., and Bhalodia, J. (1998) Automated Capture, Integration, and Visualization of Multiple Media Streams, in *Proceedings of the 1998 conference on IEEE Multimedia and Computing Systems*.

Abowd, G.D., Atkeson, C.G., Brotherton, J., Enqvist, T., Gulley, P., and LeMon, J. (1998b) Investigating the capture, integration, and access problem of ubiquitous computing in an educational setting, in *Proceedings of the 1998 conference on Human Factors in Computing Systems (CHI'98)*, 440-7.

Abowd, G.D., Atkeson, C.G., Feinstein, A., Hmelo, C., Kooper, R. Long, S., Sawhney, N. and Tan, M. (1996) Teaching and Learning as Multimedia Authoring: The Classroom 2000 project, in *Proceedings of the ACM Conference on Multimedia (Multimedia'96)*, 187-98.

Bass, L., Clements, P., and Kazman, R. (1998) Software Architecture in Practice, Addison-Wesley.

Boehm, P., Bose, P., Horowitz, E. and Lee, M.J. (1994) Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach, in *Proceedings of the International Conference on Software Engineering (ICSE 17)*.

Bose, P. (1995) A Model for Decision Maintenance in the WinWin Collaboration Framework. *Knowledge Based Software Engineering (KBSE'95)*.

Conklin, J.E. and Yakemovic, K.C.B. (1991) A Process-Oriented Approach to Design Rationale. *Human-Computer Interaction*, **6**(3&4), 357-91.

Degen, L., Mander, R. and Salomon, G. (1992) Working with Audio: Integrating Personal Tape Recorders and Desktop Computers, in *Proceedings of 1992 conference on Human Factors in Computing Systems (CHI'92)*, 413-18.

Garlan, D., Monroe, R.T. and Wile, D. (1997) ACME: An Architecture Description Interchange Language, in *Proceedings of CASCON'97*, 169-83.

Jerding, D. and Rugaber, S. (1997) Using Visualization for Architectural Localization and Extraction, in *Proceedings of the Fourth Working Conference on Reverse Engineering*, October.

Kazman, R., Abowd, G., Bass, L., and Clements, P. (1996) Scenario-Based Analysis of Software Architecture. *IEEE Software*, **13**(6), 47-56.

Kazman, P., Bass, L., Abowd, G. and Webb, S.M. (1994) SAAM: A Method for Analyzing the Properties of Software Architectures, in *Proceedings of the International Conference on Software Engineering (ICSE 16)*, 81-90.

Minneman, S. Harrison, S. Janseen, B., Kurtenbach, G., Moran, T., Smith, I. And van Melle, B. (1995) A Confederation of Tools for Capturing and Accessing Collaborative Activity, in *Proceedings of the ACM Conference on Multimedia (Multimedia'95)*.

Moran, T.P., Palen, L., Harrison, S., Chiu, P., Kimber, D., Minneman, S., van Melle, W., Zelweger, P. (1997) Salvaging Multimedia Meeting Records, in *Proceedings of the 1997 conference on Human Factors in Computing Systems (CHI'97)*, 202-9.

Parnas, D.L. (1972) On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, **15**(12), 1053-58,

Potts, C., Takahashi, K., and Anton, A.I. (1994) Inquiry-Based Requirements Analysis. *IEEE Software*, **11**(2), 21-32.

Rittel, H. and Webber, M. (1973) Dilemmas in a general theory of planning. *Policy Science*, **4**, 155-69.

Shum, B.S. and Hammond, N. (1994) Argumentation-Based Design Rationale: What Use at What Cost? *International Journal of Human-Computer Studies 40*, **4**, 603-52.

Stifelman, L.J. (1996) Augmenting real-world objects: A paper-based audio notebook, in *Proceedings of 1992 conference on Human Factors in Computing Systems (CHI'92)*, 199-200.

Weber, K. and Poon, A. (1994) Marquee: A tool for real-time video logging, in *Proceedings of 1994 conference on Human Factors in Computing Systems (CHI'94)*, 58-64.

Whittaker, W., Hyland, P. and Wiley, M. (1994) Filochat: Handwritten notes provide access to recorded conversations, in *Proceedings of 1994 conference on Human Factors in Computing Systems (CHI'94)*, 271-77.