Software Engineering Institute

Using ArchE in the Classroom:
One Experience

Felix Bachmann
Len Bass
Philip Bianco
Mark Klein

September 2007

TECHNICAL NOTE
CMU/SEI-2007-TN-001

Software Architecture Technology Initiative

Carnegie Mellon

Table of Contents

List of Figures

List of Tables

Abstract

The Architecture Expert (ArchE) tool serves as a software architecture design assistant. It embodies knowledge of quality attributes and the relation between the achievement of quality attribute requirements and architecture design. This technical note describes the use of a pre-alpha release of ArchE in a graduate-level software architecture class at Clemson University. ArchE was used to assist the students in the architecting process. The tool was then evaluated by the students and instructor. The instructor felt that ArchE met his objectives as a pedagogical tool. The students, although critical of the pre-alpha status of ArchE, were enthusiastic about the benefits of having the step-by-step guide to the architect's designing process as provided by ArchE.

1 Introduction

The Carnegie Mellon® Software Engineering Institute has developed and is distributing the Architecture Expert (ArchE) [SEI 2007, Bachmann 2003]. ArchE is a software tool intended to serve as an architect's assistant. It aids in developing architectures that possess specified levels of required qualities. The version of ArchE used in the case reported here was a pre-alpha release version.[1] ArchE embodies knowledge of theories regarding quality attributes and uses these theories to predict quality attribute responses of the architecture in given situations. This technical note details the use of ArchE in a graduate computer science course on software architecture. This usage was an early investigation into the effectiveness of various aspects of ArchE. This note will examine aspects of using ArchE as a tool to teach about architecting.

ArchE is intended to be an assistant to the designer rather than a designer. ArchE has knowledge of quality attributes but no knowledge of any problem domain. Consequently, ArchE can offer advice about satisfying quality attribute requirements but does not know what this advice means to the architect with respect to the domain of the system.

ArchE uses responsibilities to represent units of computation within the design being generated [Wirfs-Brock 2002]. A *responsibility* is an action, a set of knowledge maintained by or a set of decisions to be carried out by a software system or an element of the system. A computer system can be characterized in terms of the responsibilities of the system, their interrelationships, and their assignment to elements.

A sample interaction of an architect with ArchE might begin with the architect inputting the features (functions) that the system being designed must provide. The architect then inputs quality attribute requirements and, optionally, a prespecified portion of the design such as the use of specific components.

ArchE then requests additional information necessary to determine quality attribute behavior, such as the execution times or the cost of changing various features.

Then ArchE proposes an initial design, points out the quality attribute requirements not satisfied by this design, and proposes a collection of architectural transformations to improve the design with respect to the quality attribute requirements.

The architect selects a transformation and provides additional information for the new elements of the design, such as meaningful names, execution times, or cost of change. This process continues until either all the quality attribute requirements are satisfied or ArchE has no more proposals.

ArchE currently has quality attribute knowledge of real-time performance and modifiability.

®   Carnegie Mellon is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1   ArchE V2.1 and associated user guide is now available at http://www.sei.cmu.edu/architecture/arche.html.

ArchE is implemented as an Eclipse application, which provides immediate familiarity with the user interface and concept of operation for anyone who has used Eclipse or other Eclipse-based applications. In the case described here, many of the students have used Eclipse in previous courses.

We begin by presenting the context in which ArchE was being used and then discuss the problem for which the students were asked to create an architecture. We discuss the pedagogical issues associated with teaching software architecture and then present the evaluations of the students and the instructor.

In general, the students were enthusiastic about a tool that provided step-by-step assistance in the architecture design process, and the instructor was enthusiastic about the use of ArchE as a pedagogical tool. We present the students comments verbatim in Appendix B.

1.1 CONTEXT

ArchE was used in Computer Science 875, Software Architecture, a graduate course in the computer science curriculum at Clemson University, during the Spring 2006 semester. There were 18 students in this course who were divided into six teams. The course is typically populated by 75–80% masters of science students and 20–25% doctoral students. The course has been offered once a year for the past six years. The instructor for the course was John McGregor, an author of this report.

The course uses the Software Engineering Institute (SEI) Attribute-Driven Design (ADD) method, which emphasizes achieving the required levels of specified quality attributes [Bass 2003, Ch. 7]. Students study architectural tactics and methods for making tradeoffs among qualities as the architecture is defined. The book *Software Architecture in Practice* is used as the basic reference text [Bass 2003]. Students also read both research and experience reports about architecture.

The course requirements include a semester-long project. The students incrementally define an architecture for a specific product. They begin by defining the requirements for the product, including desired qualities. The tactics-based approach takes the students through an iterative decomposition approach. Their final deliverable is an architecture model that is documented according to the SEI Views and Beyond Approach [Clements 2003].

1.2 THE INVESTIGATION

ArchE was introduced to students in the last six weeks of the spring 2006 semester. The students had been working on developing the architecture for the Clemson Travel Assistant System (CTAS) using ADD for most of the semester. Teams of four to five students used the use case notation of the Unified Modeling Language (UML), an Object Management Group standard [Wikipedia 2007], and the Architecture Analysis and Design Language (AADL), a standard of the Society of Automotive Engineers, to represent their particular architectures [SAE 2004].

Students used ArchE to construct explicit architectural models for the modifiability and performance quality attributes. The students used existing requirements and a starting set of scenarios that

were constructed as part of an exercise using the SEI Architecture Tradeoff Analysis Method® (ATAM®) from earlier in the semester [Clements 2002]. These scenarios were expanded to more completely consider the two targeted quality attributes.

The use of ArchE by the class had two primary objectives:

1. to investigate the usability of ArchE from the perspective of graduate students

2. to investigate the usefulness of ArchE in an instructional setting

Our use of ArchE was not a controlled, statistical study. Students and the instructor provided their impressions and perspectives on the experience.

1.2.1 Usability of ArchE by Students

One of our goals was to see how easily students were able to use ArchE to build architecture models that satisfy precisely specified quality attribute requirements. The students in the course had a wide range of backgrounds; they included foreign exchange students with little previous software engineering experience, beginning master's degree students with only coding experience, and PhD students with work and classroom experience in software engineering. A few of the students had never used an Eclipse-based product, while most used Eclipse regularly.

Students were given a brief demonstration of ArchE in one 75-minute class meeting. They were also given a step-by-step tutorial that guided the student through the use of ArchE. (This tutorial is provided in Appendix C.) Each class session included time for discussion about ArchE and the models the students were constructing.

The main data collected relative to the usability of ArchE by graduate students were comments from the students. These comments are summarized in Section 4.1 and reported verbatim in Appendix B. The instructor also used ArchE and contributed additional comments, which are included in the summary in Section 4.

1.2.2 Usefulness of ArchE in Training Software Architects

The instructional study examined the usefulness of ArchE for training new software architects. John McGregor has been training software architects both in the university and as an industry consultant for 15 years. He was interested in whether the use of an expert system would help the students learn about applying architectural tactics.

John McGregor worked with each team in the class and observed its use of ArchE. The main data collected regarding the usefulness of ArchE were submitted to him in class by students, in the form of reports and comments. These comments are summarized in Section 4.2.

---

2 The Class Problem

The class problem was to design a traveler's assistant that would allow the traveler to plan individual itineraries, including those for multiple-mode trips, such as those combining taxi, airplane, train, and bus trips. The product was named the Clemson Traveler Assistant System (CTAS) and intended for execution on a variety of platforms, including handheld devices carried by a traveler, dashboard devices, and home or office desktops. The handheld devices would include features such as wireless connections to check schedules and make reservations in real time and a global positioning system (GPS) to aid in estimating time of arrival at a destination. The complete problem description can be found in Appendix A.

The problem had several elements that made it well suited for this experiment.

- Performance has a high priority for the CTAS. Performance is one of the qualities for which the current version of ArchE has a reasoning framework. The CTAS has at least two levels of performance requirements.

    1. The interface with a wireless communication device requires a hard real-time response.

    2. Searching the space of possible itineraries for the optimal itinerary must be done sufficiently fast for the traveler to react and follow the directions in the selected itinerary. This type of requirement requires a soft real-time response.

- Modifiability is also a high priority for the CTAS. It is the other quality about which the current version of ArchE can reason. Traveler assistants are a relatively new type of product. The state-of-the-art feature set is expanding rapidly, which makes the ability to modify products as the domain evolves a high priority.

- The units that ArchE uses to represent elements in the architecture—responsibilities—have a natural relationship to the use cases engaged to specify the product. The initial set of responsibilities can be built from the set of use cases. Additional responsibilities are defined as these initial responsibilities are decomposed into more fine-grained responsibilities.

- The class problem covered a wide range of different structures and system types. Trade-offs between dedicated and Web-based interfaces as well as other issues make the problem a rich source of examples for classroom discussion. Interfacing with services is currently a topic of much interest and provides one approach to structuring the system. For example, the New Jersey/New York Port Authority currently provides a traveler assistant for planning trips from Newark Liberty Airport to a variety of destinations, including downtown Manhattan.

3 Pedagogy

The course used three major pedagogical devices: lecture/discussion, hands-on out-of-class exercises, and independent review of the current research literature. After a brief overview of the course, this report will focus on the portion of the course that incorporated ArchE.

1. lecture – Each lecture session began with an initial period in which students asked questions about previous lectures, assigned readings, and exercises that involved presentations of concepts and techniques. The questions often led to presentations on techniques for handling specific problems.

2. exercises – The CTAS problem was attacked in a series of increments. Each exercise was a step toward a complete architecture.

3. literature review – Students were assigned research papers to read, summarize, and critique.

ArchE was used during the last six weeks of the semester. It was incorporated into each of the pedagogical elements of the course.

• lecture – Lectures covered a number of architectural patterns and the tactics associated with each, with emphasis on those tactics available in ArchE. The concept of a reasoning framework was explained using the modifiability framework in ArchE [Bass 2005].

• exercises – Teams were given the tutorial and a goal. Each team worked independently outside of class to complete the CTAS architecture design with appropriate levels of performance and modifiability.

• literature review – Students were assigned SEI technical reports that explained the concepts of reasoning frameworks and provided a sample framework.

4 Evaluation

ArchE was evaluated by both the students and the course instructor at the end of their use of ArchE. Data was collected in the two areas of interest: usability and instruction. In this section we summarize those evaluations.

4.1 STUDENT EVALUATION

The students' evaluation comprised comments made to the instructor and their responses on a questionnaire distributed via email after the conclusion of the course. The questionnaire included the following questions:

- How did ArchE make the architecture definition process better?

- How did it make it worse?

- What features did you like the best?

- What suggestions do you have for additional features?

The students' evaluations are summarized in this section, but verbatim comments from each project team are included in Appendix B.

The students liked having a tool such as ArchE that would "look over their shoulders and make suggestions." Architecting a software system is a complex task with many facets. Students find that once they understand the true nature of architecture, they are often intimidated by the large number of possible actions that they could take at any moment and are happy to get any help they can. The students also made comments that reflect the immaturity of ArchE as a tool. They would like improvements in the user interface, the functionality provided, and the documentation.

4.1.1   Usability

Most of the students had used Eclipse before and found the basic features of the tool easy to use. The use of responsibilities and functions as the basic building blocks of the architecture was harder to grasp, since the course had discussed only modules and had not focused on responsibilities. However, a brief discussion of "mapping" from a requirements view of the system to a functional view provided enough information to keep the students moving forward.

Perhaps the feature students liked the best was the automatic computation of quality attribute levels. Methods given in the literature for building performance models and other formal mathematical models are sometimes difficult to understand and always tedious to compute. ArchE handled all of that detail for the student and provided answers rapidly, allowing the students to build multiple models with different parameter values in much less time than they could evaluate the model for a single scenario in a manual performance model. To quote Team 6, "The entire knowledge of calculating the dependent parameter and applying the tactics to adjust the independent parameters resides in the reasoning frameworks, which makes the architecture definition process simpler."

4.1.2   Instruction

ArchE helped the students gain a better understanding of specific tactics. When a tactic is applied in ArchE, the quality attribute value is reevaluated, and all scenarios are marked as met or not based on this value. This immediate feedback allowed the students to more clearly understand the relationship between the tactic and its effect on the model.

To quote Team 2: "The overall concept is very convincing...with a little refining the software should be great. The interface was very intuitive and prompted us to enter values wherever the current values were conflicting or erroneous. That helped a lot. It showed us the exact slot value that needed to be changed as the relationships were also clearly underlined."

4.2 INSTRUCTOR EVALUATION

The instructor's evaluation was based on discussions with students, observation of student teams using the tool, and using the tool to construct demonstrations. Teams were asked to report in class on their progress and any problems they encountered. Initially there were problems with the operation of an unfamiliar tool. The in-class reports allowed teams to assist each other (and inform the instructor) about some techniques that were not obvious. The instructor, John McGregor, is one of the authors of this report.

4.2.1   Usability Study

Since ArchE has an expert system component, the results of a user's actions were not always predictable. The use of slightly different parameters made seemingly similar models behave differently. At first this appeared to be an aggravation, but it resulted in numerous interesting discussions. The students were forced to pay closer attention to the meaning of each of the values they assigned, which helped them understand the impact of each decision.

The instructor found that ArchE still requires some features for use in the classroom and perhaps in industrial settings as well. There needs to be a mechanism that lets the user "undo" a decision. This mechanism amounts to unrolling the inference engine's latest actions. The students worked around this deficiency by entering a model and trying one parameter value, then entering the same model again but using a different value for the same parameter. An undo option would make it easy to investigate several different scenarios quickly.

A second useful feature would be a graphical view of the architecture. ArchE provides a table view of the relationships among functions that is essentially the architecture, but it is difficult for most people to quickly understand. A graphical view would aid students in locating sections of the architecture that could use improvement.

Exposing the reasoning framework for a quality attribute would make ArchE more useful in the classroom. Although students were happy to let ArchE compute new quality attribute values, they would learn more if they studied the reasoning framework and its underlying theory. This informational need could be addressed through more complete documentation.

4.2.2   Instructional Study

Initially it was difficult for students to understand the structures used in ArchE's architecture representation. It was a finer grained representation than had been used through most of the course and prompted a class discussion of the appropriate granularity to use in architecture representation.

Using ArchE changed how the class operated. The instructor became more of a moderator and facilitator than the source of information. Students could understand the effects of their decisions without the instructor's pointing them out because ArchE produced the explanations.

5 Conclusion

The use of ArchE in Computer Science 875 at Clemson University was largely a positive experience. The use of ArchE affected the way the instructor taught a portion of the course by enabling him to convert lectures to more interactive sessions. It also affected the students' approach to architecture by alerting them to the rationales behind particular architectural decisions.

The students' experiences with ArchE are summarized by the comments from the six teams:[2]

- Team 1: "The method to apply tactics and obtain the information from the reasoning framework also helps making it better."

- Team 2: "The overall concept is very convincing."

- Team 3: "The good thing about ArchE during architecture design process is that it automatically computes the effect of changing one quality attribute on the whole architecture and rearrange the cost of different scenarios."

- Team 4: "The scenario based approach makes it easier to think about how architectural decisions will impact the required quality attributes of the system."

- Team 5: "ArchE helps to make the architecture process easier."

- Team 6: "The entire knowledge … resides in the Reasoning Frameworks which makes the architecture definition process simpler."

During 2006-2007, a newer version of ArchE was used as a portion of a tool chain that used a requirements tool as input to ArchE and ended with the execution of a simulator based on the output of ArchE.

---

[2] Comments are presented verbatim and have not been edited.

Appendix A   Complete Problem Specification

The Clemson Traveler Assistant System (CTAS)

The CTAS is an itinerary planning system that allows a traveler to plan the routes and modes of transportation needed to travel from one point to another. It executes on a variety of platforms, including a wireless handheld device, and allows travelers to periodically update their information and reconsider their itineraries. Using the CTAS should result in as efficient a trip as is possible given the conditions at the time of travel.

The stakeholders in the CTAS range from the users and developers to government leaders and business owners. Business owners want their costs of providing information to be low. They need to be able to automatically update their information as, for example, cars leave or enter a parking lot. Government leaders want the devices to be affordable to a wide range of people. Users want good value and ease of use. Developers want to use reliable and familiar technologies to speed development.

The stakeholders in the CTAS have a number of interests. The users of the system are interested in making travel to their destinations easier, provided using the system is not difficult. The governments of locales served by the system are interested in reducing traffic congestion and generally reducing the impact of travel on area businesses. The providers of information to the system, such as hotel owners, railroad operators, and parking lot owners, are interested in having their information accessible to as many travelers as possible and as accurately as possible. Most of these providers are also interested in maximizing their revenue.

The scope of the CTAS is the software running on the device. We will assume that the information services, such as those that provide transportation schedules or make reservations for parking places and other resources, are available. The architecture under development will be for the CTAS only.

CTAS Actors

The actors in the CTAS have a number of differing goals. The CTAS user, the primary actor, wants to plan and execute a trip in the least expensive, fastest, or shortest manner possible. (Different users will rank these criteria differently.) Secondary actors include information providers such as parking lot operations, transit systems, taxi companies, airlines, and map services. These actors want to attract business by providing fast response and accurate data. Information providers will change with locale and may change dynamically as they go offline outside their hours of operation. A CTAS device will have a core set of features that may be expandable through attachment to an expansion device. A CTAS device's feature set will have the ability to adapt to a changing set of peripheral devices.

Table 1 describes the actors in the CTAS.

Table 1:   CTAS Actors

Actor	Description
CTAS user	Users have a few routine destinations to which they will travel repeatedly from a usual origin, for example, from home to work. They must be able to plan and revise trips on an ad hoc basis. They need easy-to-understand itineraries that reflect their familiarity with the route they take.
CTAS information provider	Any actor that provides data to the CTAS for use in computing itineraries. A vehicle in which the CTAS is being transported may provide time-to-destination information. A parking lot operation may provide its lot location and availability information.
CTAS smart unit	A special kind of information provider such as a building or transportation vehicle that provides information to its users, such as a map, rules, or automated help
CTAS device	Any device on which an instance of the CTAS may be hosted. This may be a dedicated device or a multipurpose device such as a smart phone or Personal Digital Assistant.
CTAS-related hardware	An abstract secondary actor that can be any piece of hardware that touches a CTAS device
CTAS peripheral	A secondary actor that adds a specific capability to the CTAS device such as GPS capability. Certain preplanned peripherals are automatically recognized, and the behavior of the system will adjust to their presence or absence. For example, when there is no GPS peripheral attached, the system asks the user for a location.
CTAS expansion device	A secondary actor that provides a larger, more capable platform, such as a vehicle or service port that can expand the CTAS device's bandwidth range of output devices

Qualities

Using the ISO 9126 framework, we specify the qualities in Table 2.

Table 2:   CTAS Qualities

Quality	Subquality	Scenario-Specific Requirements
Functionality	Accuracy	The itinerary produced by the CTAS should be as accurate as the information provided to it. The system should fail visibly if the capacity of the system is exceeded, rather than produce faulty results.
	Interoperability	The CTAS should be able to accept information from a wide range of information providers. Any formal or de facto standards used should be identified and followed.
	Security	While communication between the CTAS and information providers should be reasonably secure, this is not a primary concern, since the information involved is publicly available. However, communication between the CTAS and the user should be very secure.
Reliability	Recoverability	Any itinerary should be available for use even in the event of spontaneous reboot of the system.
Usability	Understand-ability	The system should be understandable to users with an eighth-grade reading ability.
	Learnability	The system should be learnable by a person capable of following the instructions for operating consumer electronic products.
	Operability	The system should be operable by anyone capable of operating a telephone keypad.
Efficiency	Time behavior	The CTAS should be able to produce an itinerary within 30 seconds of receiving the command.
	Resource utilization	The CTAS should be capable of operating in 256 MB of dynamic memory.
Maintainability	Analyzability	A CTAS maintainer should be able to estimate the effort for a requested modification within four hours.
	Changeability	A CTAS maintainer should be able to accomplish most changes within three working days.
	Testability	The CTAS should be testable with a level of effort one-third of the total development effort.
Portability	Adaptability	The CTAS will be capable of being ported to a new device by replacing externally linkable drivers.
	Installability	The CTAS should be packaged with an automated installer usable by anyone meeting the usability requirements above.
	Replaceability	The CTAS should be upgradeable through the same process used for the initial installation.

When the students performed an SEI Quality Attribute Workshop (QAW) [Barbacci 2003], they identified modifiability (changeability) and performance (time behavior) as two of the five highest priority qualities. Those qualities are the basis for the models developed below.

CTAS Use Cases

The uses of the CTAS shown in Figure 1 will serve as its basic requirements. Additional requirements were derived from these basic ones.
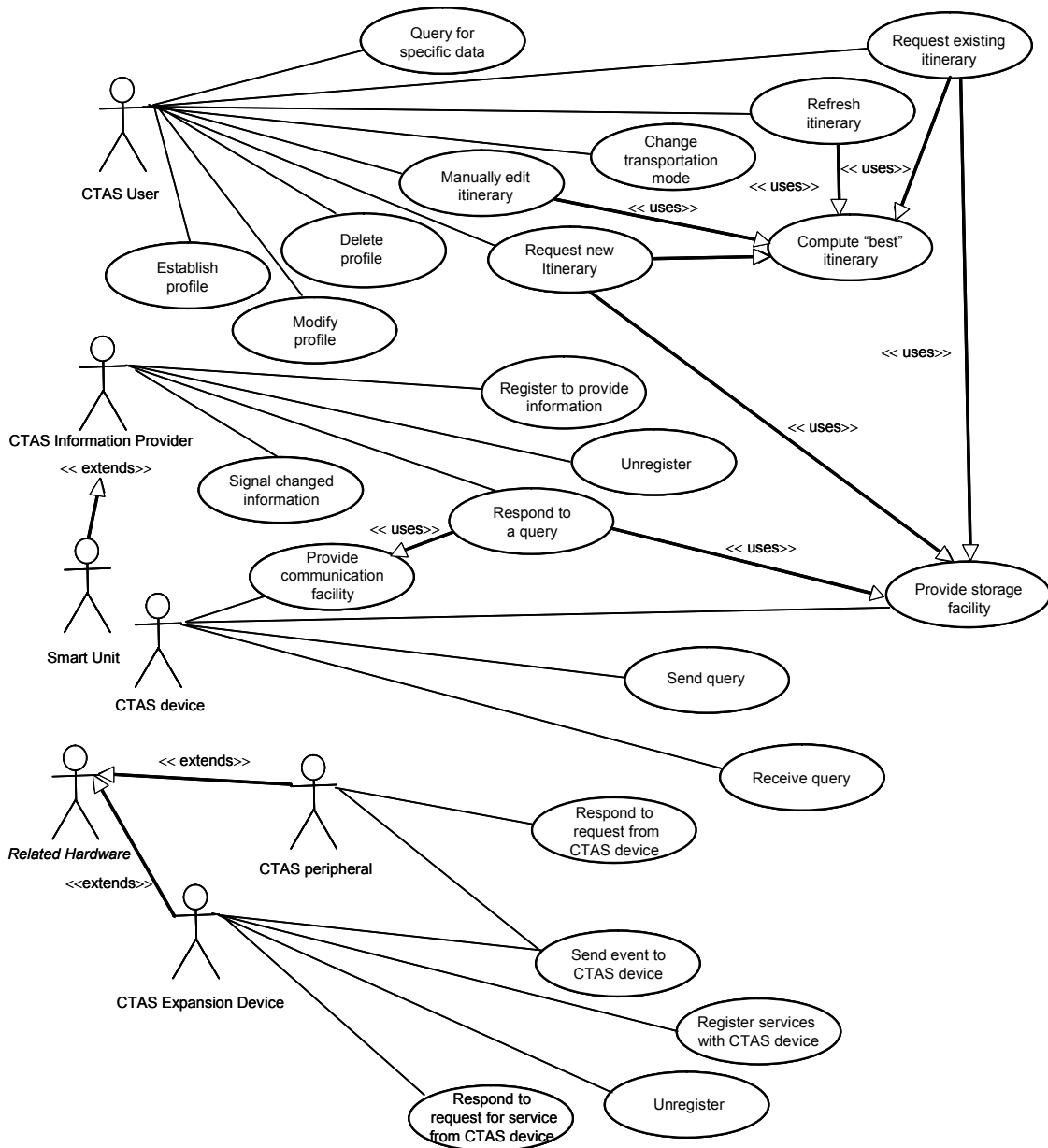
Figure 1:   Use Case Diagram

Appendix B   Student Feedback

Below is the verbatim feedback received from the 18 students who were divided into six teams. [3]

How did ArchE make the architecture definition process better?

1. ArchE supports building models that reason about quality attributes like modifiability and performance. Each reasoning framework uses algorithms to compute estimates of the quality about which it reasons. Hence, the user interface for developing scenarios for the quality attributes is a convenient one, which makes the architecture definition better. Also, the method to apply tactics and obtain the information from the reasoning framework also helps making it better.

2. The overall concept is very convincing...with a little refining the Software should be great. The interface was very intuitive, and prompted us to enter values wherever conflicting or erroneous. That helped a lot. It showed us the exact slot value that needed to be changed as also the relationships were clearly underlined.

3. The good thing about ArchE during architecture design process is that it automatically computes the effect of changing one quality attribute on the whole architecture and re-arrange the cost of different scenarios. So it helps architecture to trade off on different qualities and see how it affects its architecture automatically.

4. The scenario based approach makes it easier to think about how architectural decisions will impact the required qualities attributes of a system. We experimented with only a few scenarios but I think ArchE will prove to be much more useful when there are a lot of scenarios to consider.

5. ArchE helps make the architecture definition better by allowing the user to create a mapping between scenario and responsibility. This helps in showing if all responsibilities are covered by the scenarios.

6. ArchE provides a convenient user interface for developing scenarios for the quality attributes. It has built in Reasoning Frameworks for the 2 quality attributes - modifiability and performance. These Reasoning frameworks resolve the conflicts among different quality attribute specific models which may involve creating new responsibilities or splitting earlier responsibilities or adjusting the independent variables of the specified scenarios. The entire knowledge of calculating the dependent parameter, applying the tactics to adjust the independent parameters, resides in the Reasoning Frameworks which makes the architecture definition process simpler.

How did it make it worse?

1. The dependencies formed in the architecture makes the ArchE fall into a cycle, due to which while we try changing the parameters in one scenario, has an overall effect on the other, which makes it troublesome. Also, while we were doing our project we faced a diffi-

---

[3]   The student responses are presented here in their original form and have not been edited.

culty wherein ArchE was not able to analyze the scenarios even when it was mapped to responsibilities.

2. The software seems to need a lot of processing power to run...it worked fine on my laptop, but I can't say the same about others.

3. Sometime ArchE applies some tactic which is completely not related. Like while doing the assignment ArchE applied the tactic to combine 2 performance scenarios and it just appended the scenarios which as a whole sentence doesn't make any sense.

4. ArchE gives us the numbers but the task of interpreting the data is left to the Architect. That is good for an experienced architect but it is difficult for junior architects to understand ArchE's results. The error messages are also not very helpful.

5. In ArchE, whenever you are creating a new scenario or responsibilities, there are specified fields for user to input. But these fields may be a source of confusion for novice users not knowing which fields are required and what impact they might have. Also, for some pre-populated selection boxes, the users might want to have something not already specified. I could not find a way to add more choices to those selection boxes.

6. While resolving the conflicts between the scenario requirements, ArchE sometimes ends up caught in a cycle. Adjusting parameter values of one scenario causes changing the values of the other parameters. Also sometimes it doesn't analyze the scenarios even when the scenarios are mapped to responsibilities.

What features did you like the best?

1. The best feature that I liked about ArchE was the method in which it gradually moves ahead and develops scenarios with the help of functions, mapping with responsibilities and association of these responsibilities via relationships, and using tactics to acquire the desired quality attributes.

2. As mentioned before, the prompting for correct values is a boon and takes a lot of the stress away.

3. The user interface for the ArchE is very friendly and didn't take much time to understand its functionality.

4. The clean interface based on Eclipse. For a prototype I thought it was well designed. (Although it is a little difficult to learn how to use it)

5. The questions and alerts section is a feature that I think stands out. It will ask questions that the user might not have thought about. (although not all the time) But it has room for improvements to provide better help.

6. I liked the user interface for the scenario development. And also I liked the application of design tactics like the wrapper tactic, encapsulation tactic.

What suggestions do you have for additional features?

1. It would be best if, we could have a pictorial view of the relationships with the associated responsibilities and their mappings with the scenarios, it would give us a better understand-

ing of where we are and what steps have to be taken to achieve the desired goal decided upon.

2. Documentation!!! Nothing is really clear when you use it the first time and you just need to experiment. Maybe something that uses lesser processing power.

3. ArchE should support other quality attributes also other than performance and modifiability. Also while doing the assignment when writing response measure for one of the performance scenario it doesn't show to enter the cost in terms of memory and just has option in execution time.

4. A better way to associate scenario and responsibility, and, function and responsibility. There was a lot of clicking when trying to do those tasks and I believe there must be a better way to do it. I would also like to see a report generating plug-in that creates a .PDF file that has all the outputs from ArchE. The tabs are a little hard to read.

5. I am not entirely sure what would happen if we use ArchE for a really large architecture. If there are many scenarios, functions, and responsibilities, I am concerned that the user might be required to scroll through many pages just to find what he needs. For an additional feature, maybe have the capability to organize somehow into separate files or folders to keep it modular.

6. A visual representation of the mapping between scenarios and responsibilities along with the scenario response measure values would give a clearer picture of the design. The user interaction required for applying the tactics for design could be further minimized.

Appendix C   Tutorial

The ArchE tool is developed on top of the Eclipse integrated development environment (IDE) as a stand-alone tool rather than a plug-in. The tool currently supports building models that reason about modifiability and performance qualities. For each quality, a reasoning framework of the form described by Bass is created and then incorporated into the ArchE reasoning engine [Bass 2005].

ArchE uses the Jess rule-based inference engine [Friedman-Hill 2003]. The advice is in the form of Jess rules, allowing for incremental building of the expertise. These rules form the expert knowledge of architectural tactics to achieve specific qualities. The reasoning engine is not accessible by users for modification at this time. However, there is a Jess console that shows which rules have fired.

ArchE begins with a set of required functions that it maps onto a set of responsibilities. These responsibilities are associated with each other via various relationships. The set of responsibilities is modified through the decisions of the architect as are the associated relationships. Each reasoning framework uses algorithms to compute estimates of the quality about which it reasons. As the set of responsibilities and the relationships among them change, the estimates are revised. Based on the computed quality values, ArchE suggests tactics to the architect and will help by automatically changing some portion of the model while leaving some portions to be updated manually. (These are noted as suggestions.)

ArchE is driven by the need to build an architecture that satisfies a set of scenarios. A scenario addresses a specific quality attribute and specifies its value that should be achieved during the scenario. For example, "The CTAS can be modified to accept a new source of information in less than half a day's effort" is a modifiability scenario.

Using ArchE to Guide Decisions

ArchE makes suggestions and asks questions based on the current sets of responsibilities and scenarios and the mappings among them. The architect is free to reject suggestions or to process them in any order, except where ArchE is asking for data needed to compute a value. Typically, the architect could choose from many different actions sequences, each evoking a different response from ArchE. Therefore, there are almost always many possible sequences of actions that the architect could take, and each may evoke different responses from ArchE. The following is one possible sequence of interactions with ArchE for creating the architecture for the CTAS.

1.   Create a new ArchE project using the File | New menu selection.

2.   Select the Functions tab in the upper right pane of the ArchE main screen.

3.   Enter the basic functions that are at the level of granularity for the model you wish to create. For example, a function might be created for each use of the system shown in the use case diagram in Figure 1. The CTAS architecture team decided this would be more fine grained than they had time to handle. The team derived a set of responsibilities from the use cases

by aggregating related use cases into a single abstract concept. Alternatively, the functions/responsibilities can be extracted from the use cases. Table 3 shows the typical structure for the portion of a use case scenario that describes the user/system interaction. The right-hand column is essentially a list of responsibilities that could be used directly in ArchE. However, this list may be too fine grained in some cases.

Table 3:   Use Case Example

The user	The system responds by
1. selects new itinerary	1. creating a blank itinerary
	2. loading the user profile for the current user
	3. raising a dialog asking for information
2. enters travel information	4. computing a new itinerary
	5. displaying the new itinerary

The team initially adopted the Model-View-Controller (MVC) architecture as the top-level architecture. The responsibilities are represented in Figure 2 superimposed on the MVC modules. Figure 3 shows the result of entering the functions into ArchE.

ArchE creates a corresponding set of responsibilities that initially is simply a one-to-one mapping with the functions.
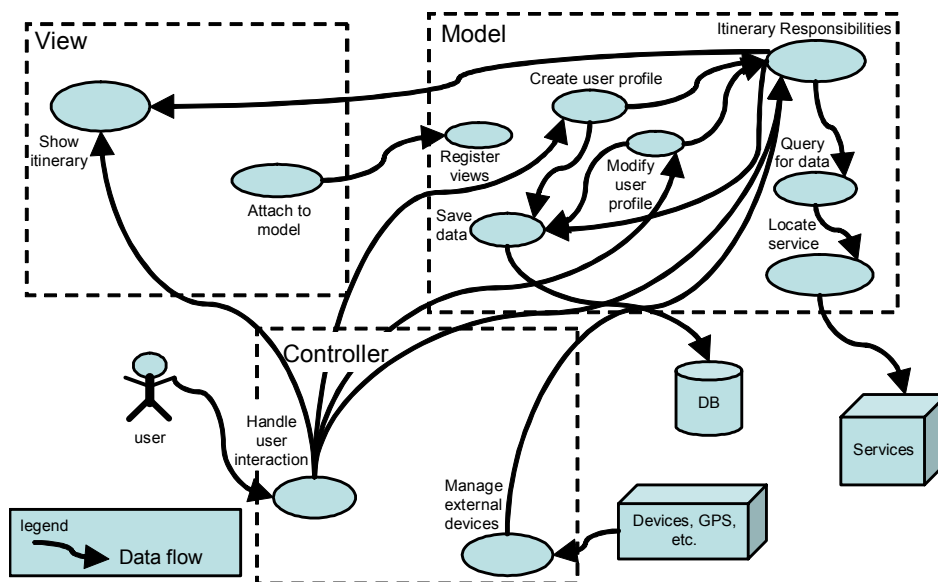
Figure 2:   Responsibility Graph

Figure 3:   Functions Entered in ArchE

4. Use the Relationships View to enter dependencies between responsibilities. For example, one responsibility may contain another responsibility, or one responsibility may provide data to another. These relationships, shown in Figure 4, capture a graph such as the one shown in Figure 2. (The Value column shows the probability that a change to one responsibility will propagate along the relationship, causing a change to be needed for the related responsibility.)

Figure 4:   The Relationships Between Responsibilities

5. Select the Scenarios tab in the upper right pane and enter scenarios that follow the SEI quality attribute scenario format. The dialog box is shown in Figure 5. Select the appropriate type of scenario—modifiability or performance—for the scenario in the drop-down menu labeled Type immediately below the scenario entry window. In this example we will only do modifiability scenarios, but pay attention because you will be asked to create a performance model at the end of this exercise.
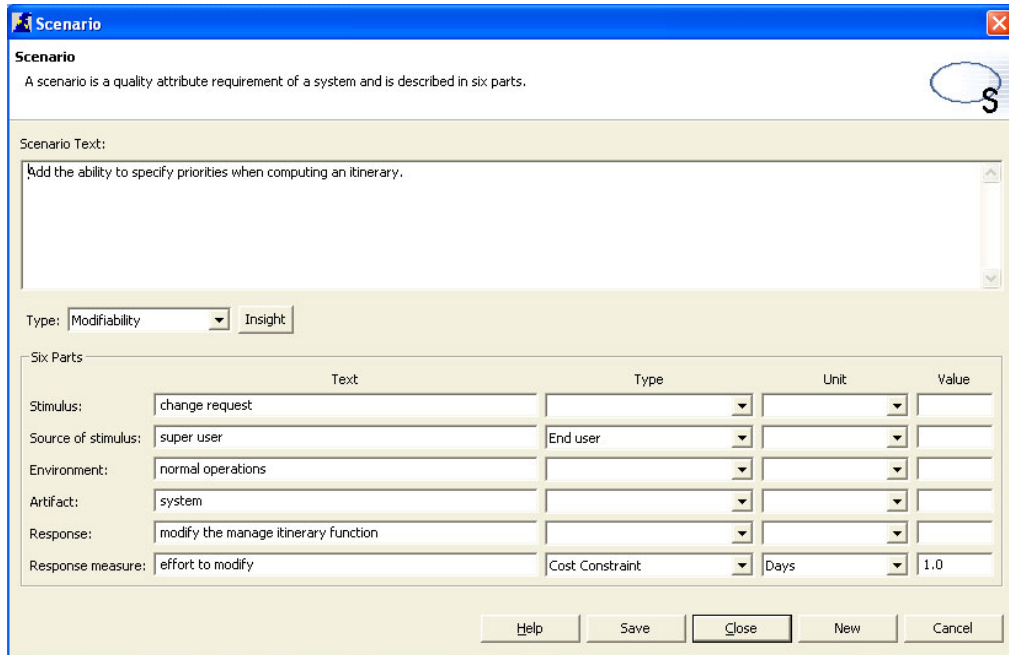
Figure 5:   Scenario Entry Screen

The modifiability model is formed from the modifiability scenarios. In this section we drill down in a modifiability model for the CTAS.

Modifiability scenarios address specific modifications to the products that are built from the architecture; for example, changing the architecture to allow different priorities on criteria, such as shortest distance or lowest cost, when computing an itinerary.

The general scenario generation table for modifiability scenarios is shown in Table 4.

Table 4: General Scenario Selection Table

Source	Environment
• end user	• at runtime
• developer	• at compile time
• system administrator	• at build time
	• at design time
Stimulus	**Response**
• add {functionality, quality, capacity} {functionality, quality, capacity}	• Locate place to modify.
• delete {functionality, quality, capacity}	• Make modification without side effects.
• modify {functionality, quality, capacity}	• Test modifications.
• vary {functionality, quality, capacity}	• Deploy modification.
Artifact	**Response Measure**
• interface	• cost in terms of number of elements
• platform	• effort
• environment	• money
• other system	• impact on other modules

6. Relate the scenarios to specific responsibilities using the mapping pane, shown in the center of Figure 6. A modifiability scenario would address changes to one or more of the responsibilities related to the scenario or to the relationships among them.
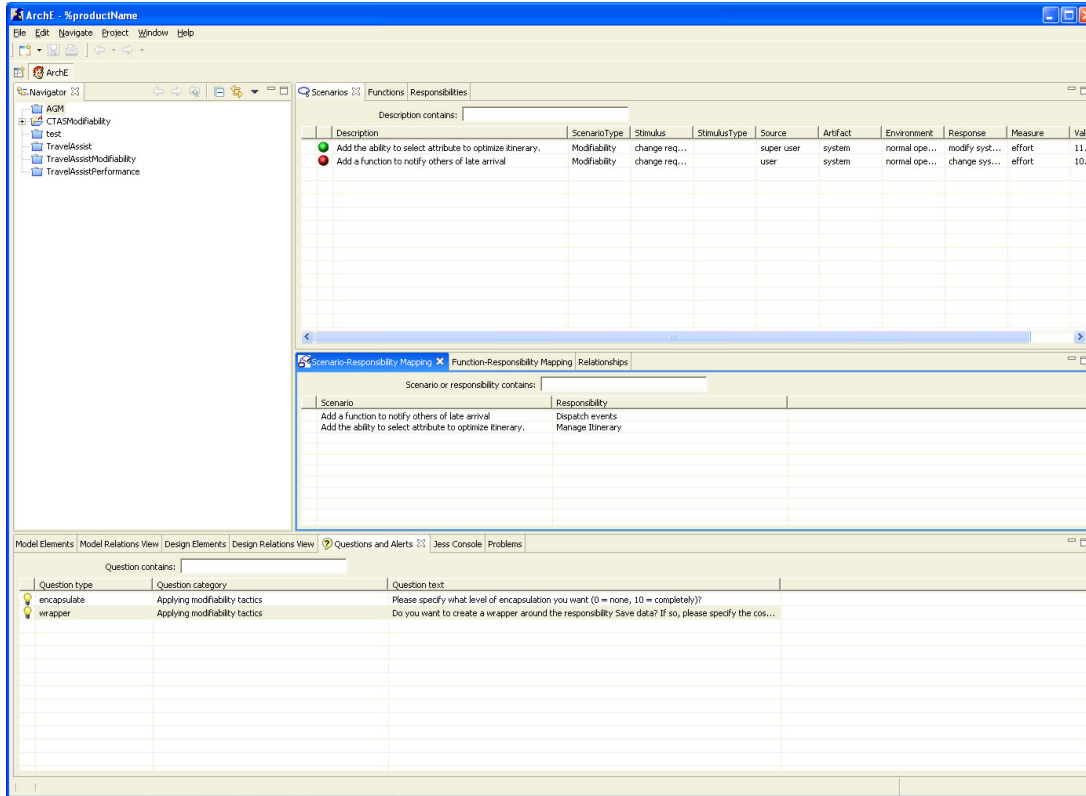
Figure 6: Scenario/Responsibilities View

In the Questions and Alerts View (see bottom of Figure 6), ArchE presents questions to obtain the data it needs to reason about the architecture. Regarding modifiability, it will ask for cost data, expressed in "days of effort," related to modifying the responsibilities. ArchE will use this data to determine whether the scenarios are satisfied, since the scenarios' response measure is also in "days of effort." A green ball to the left of a scenario, in the Scenarios View shown in Figure 7, indicates the scenario is satisfied, and a red ball indicates the scenario is not satisfied given the current data.

Figure 7:   Scenarios View

In the Questions and Alerts View, shown in Figure 8, ArchE suggests tactics that will either allow ArchE to build a complete estimate or that ArchE reasons will improve the modifiability of the architecture. In this illustration ArchE suggests two applications of "encapsulate" and one of "localize."

Figure 8:   Questions and Alerts View

Selecting the localize tactic produces the dialog box shown in Figure 9. The impact analysis in ArchE indicates that, given the current relationships among the current responsibilities, one of our scenarios is dependent on several responsibilities. The analysis indicates that applying the localize tactic might result in a saving of some effort in making future changes. In the localize tactic, a new responsibility is created that will take on a portion of the other responsibilities, allowing the architect to lower the estimate of effort required for changes to the other responsibilities affected by the scenario.
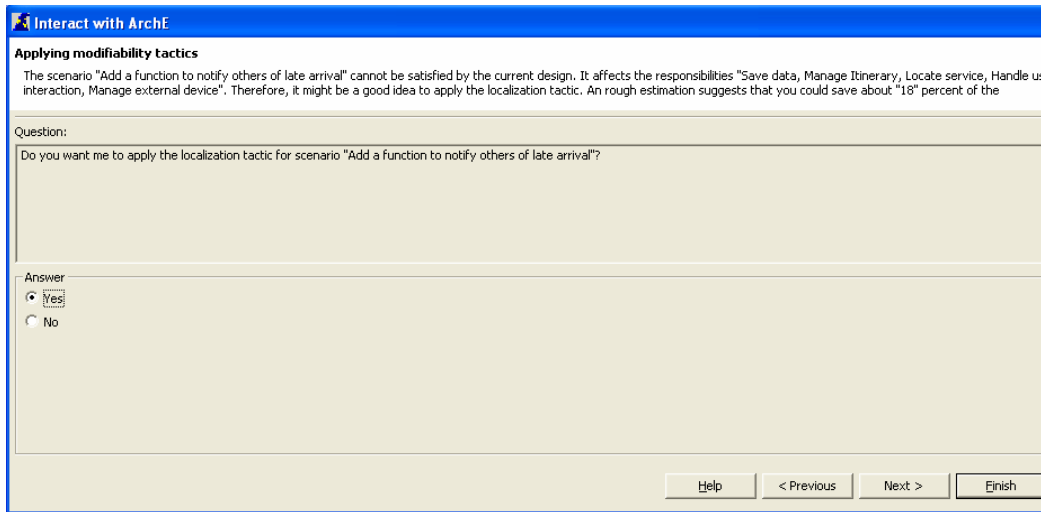
Figure 9:   Applying Tactic Dialog Box

The Applying Tactic Dialog Box opens with "Yes" selected under Answer.

7. Accept the "Yes" response by clicking Next to evoke creation of a new responsibility, as shown in Figure 10.
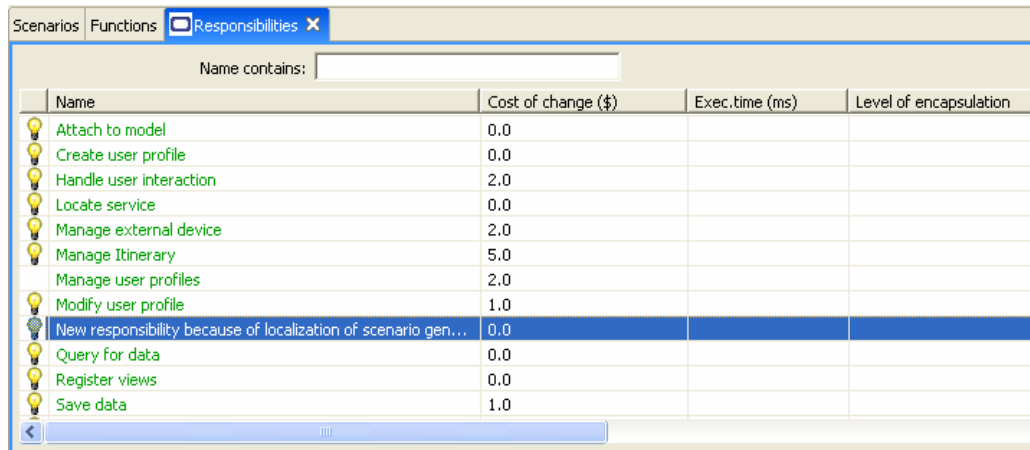
Figure 10:   New Responsibility

8. Manually edit the new responsibility, giving it a meaningful name in place of the generated name.

This editing results in the new responsibility shown in Figure 11. The new responsibility has been named "Dispatch events."

Figure 11: Edited Responsibility
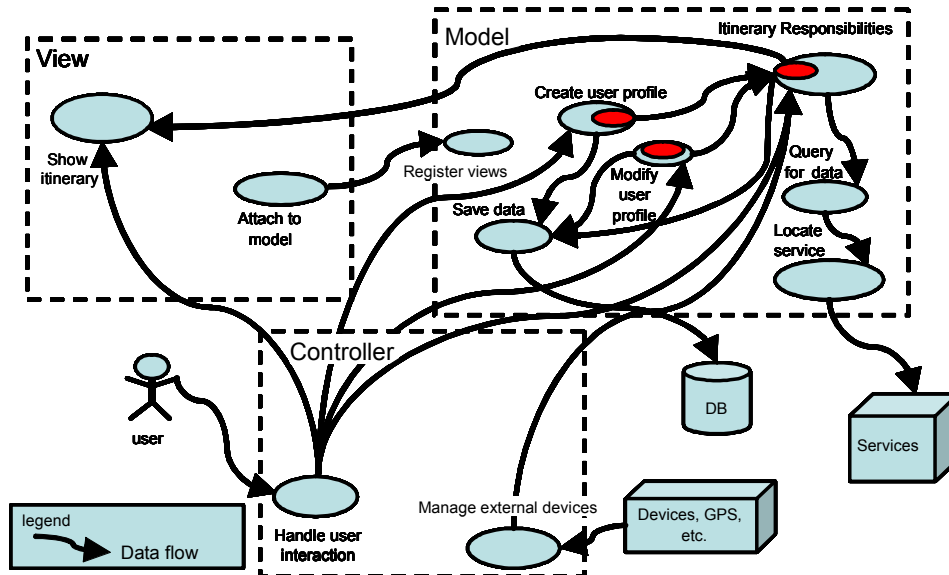
Figure 12: Identify Common Responsibility

Figure 12 and Figure 13 show those responsibilities that are modified. In Figure 12, three responsibilities that have a common dependency each have a smaller oval inside each responsibility, representing the portion of those responsibilities assumed to be common to all three. In Figure 13 a red oval represents the new responsibility, which localizes the common responsibility.
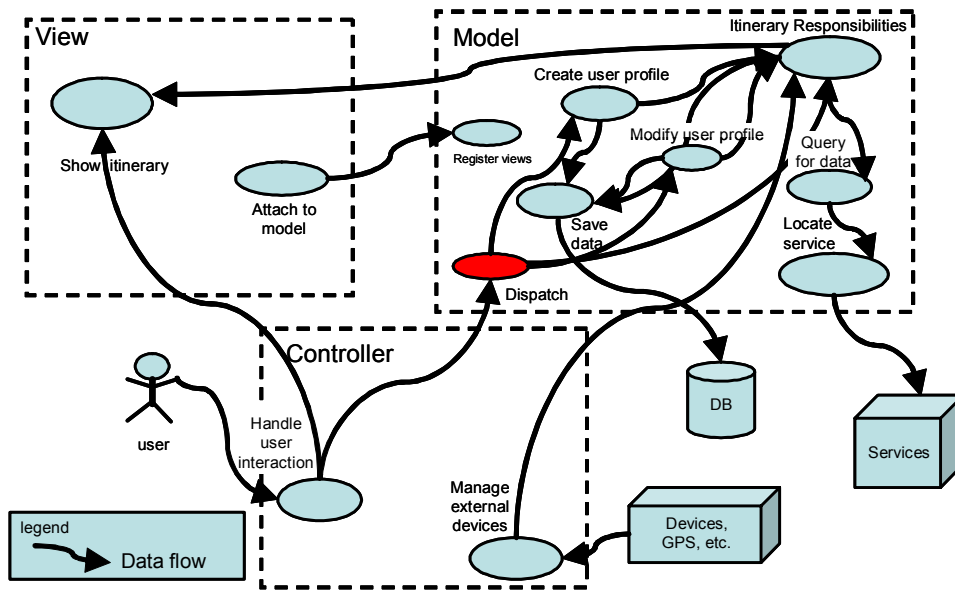
Figure 13: Revised Graph of Responsibilities

ArchE adds a number of suggestions to the Questions and Alerts View after the localize tactic has been applied. Many of these suggestions, such as "confirmCost" and "moveDependency," guide the architect to places in the model that may require changing based on the new responsibility.

Figure 14: Suggestions After Localize

The "moveDependency" suggestion allows the architect to add the new responsibility into the network of relationships. ArchE makes the suggestion for each responsibility affected.
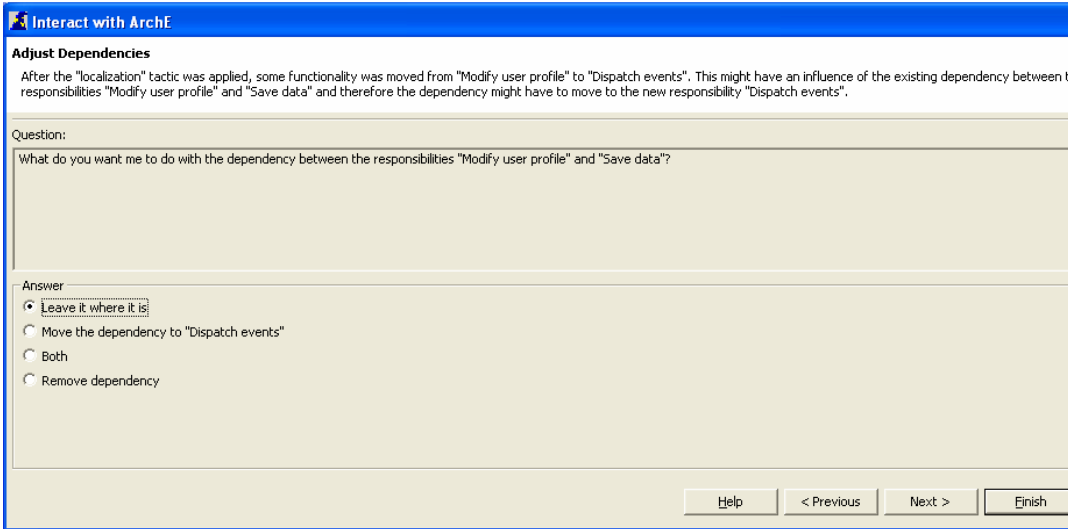
Figure 15: Adjust Dependency Dialog Box

ArchE is now suggesting that the new responsibility be encapsulated. Figure 16 shows the ad-vice given by ArchE. We elect not to encapsulate, indicated by leaving the level of encapsula-tion at 0.0, since the "Manage interface to devices" responsibility is already a single point and encapsulating will not further enhance the architecture.
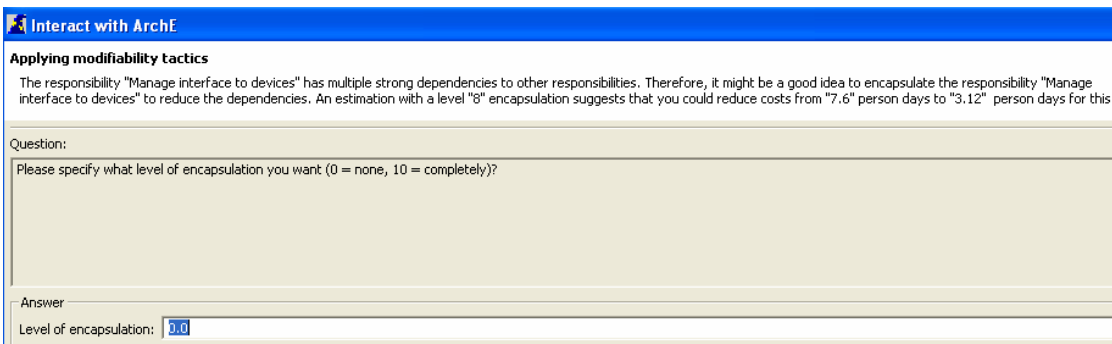

Figure 16: Encapsulation Dialog Box

Figure 11 illustrates how the change shown in Figure 9 ripples through the model. Figure 11 shows the new values on each responsibility; it also shows the wrappers that resulted from two applications of the "apply wrapper tactic" that are illustrated in Figure 17.
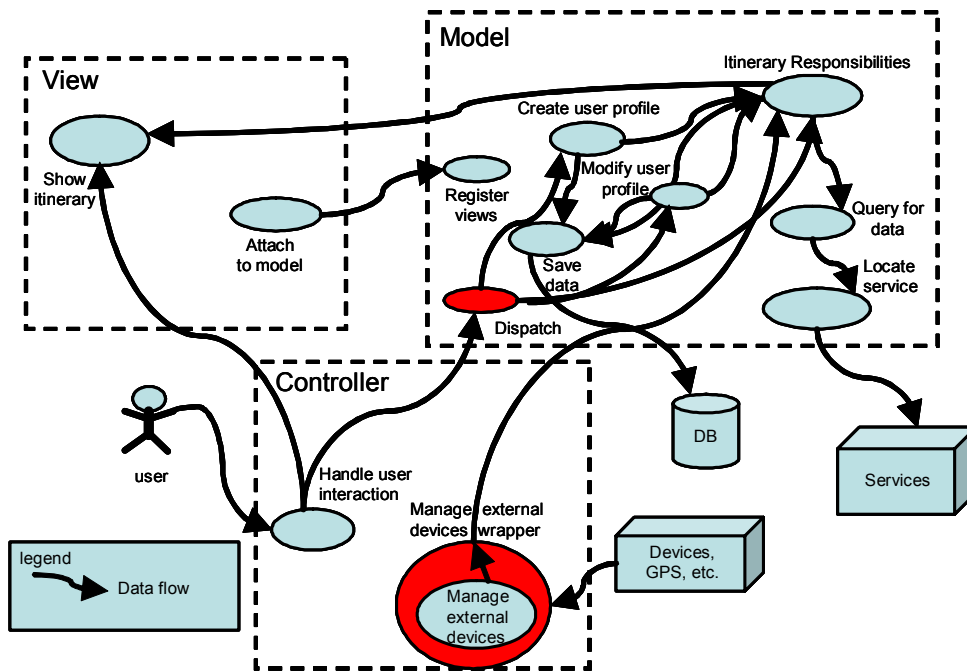
Figure 17:   Application of the Wrapper Tactic

After we chose to adjust responsibilities and modify the costs of doing so, neither of the scenarios was satisfied. ArchE has no further suggestions (since we rejected items such as encapsulation). Another possibility is to review the propagation probabilities. The estimated efforts may be based on how likely a change is to propagate from one responsibility to another. By changing those probabilities (which are default values in ArchE),

we might be able to affect the scenarios.

Figure 18: Probabilities Modified

Figure 18 shows that modifying the propagation probabilities of a set of the dependencies associated with Dispatch Events leads to satisfaction of the scenario "Add ability to select attribute to optimize itinerary."

The goal in the second scenario, shown in Figure 19, was increased until the scenario was satisfied. It's not an elegant solution, but it may involve the realistic amount of effort.
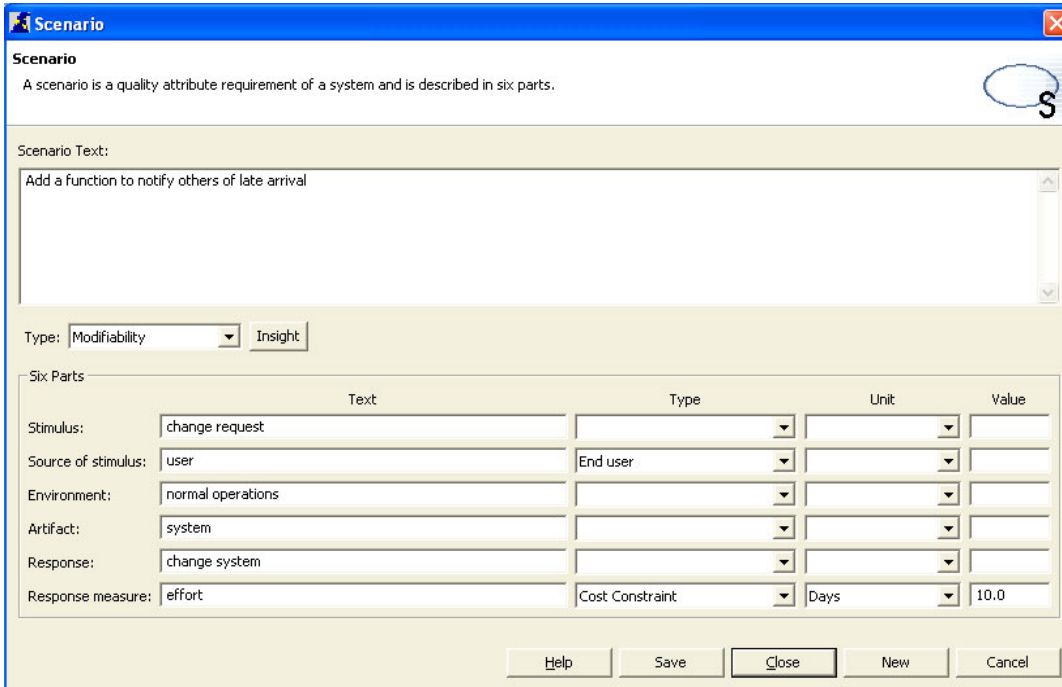
Figure 19: New Status of the Model

In the version used to produce this document, the Jess Console provides interesting viewing. It's possible to see which rules in the fact base have fired. In Figure 20, it's apparent that rules 2160 and 568 have fired.

9. Access the FactBase file found in the project directory to determine what those rules say. Also note that the values used to determine whether a scenario has been satisfied are printed on the console, although there is no association between the number printed and a scenario.
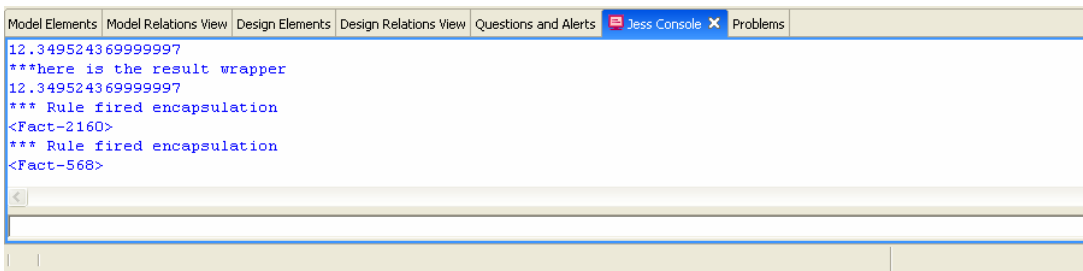
Figure 20: Jess Console

References

URLs are valid as of the publication date of this document.

[Bachmann 2003]
Bachmann, F.; Bass, L.; & Klein, M. *Preliminary Design of ArchE: A Software Architecture Design Assistant* (CMU/SEI-2003-TR-021, ADA421618). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
http://www.sei.cmu.edu/publications/documents/03.reports/03tr021.html

[Barbacci 2003]
Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs),* 3rd ed. (CMU/SEI-2003-TR-016, ADA418428). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html

[Bass 2005]
Bass. L.; Ivers, J.; Klein, M.; & Merson, P. *Reasoning Frameworks* (CMU/SEI-2005-TR-007, ADA441248). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003.
http://www.sei.cmu.edu/publications/documents/05.reports/05tr007.html

[Bass 2003]
Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice*, 2nd ed. Boston, MA: Addison-Wesley, 2003.

[Clements 2002]
Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.

[Clements 2003]
Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond.* Boston MA*:* Addison-Wesley, 2003.

[Friedman-Hill 2003]
Friedman-Hill, Ernest. *Jess in Action: Java Rule-Based Systems*. Greenwich, CT: Manning Publishers, 2003.

[SAE 2004]
Society of Automotive Engineers (SAE). *Architecture Analysis and Design Language (AS5506).* Warrendale, PA: SAE, 2004.

[SEI 2007]
Software Engineering Insitute. *Architecture Expert (ArchE) Tool*.
http://www.sei.cmu.edu/architecture/arche.html (2007).

[Wikipedia 2007]
Wikipedia. *Unified Modeling Language.*
http://en.wikipedia.org/wiki/Unified_Modeling_Language (2007).

[Wirfs-Brock 2002]
Wirfs-Brock, R. & McKean, A. *Object Design.* Boston, MA: Addison-Wesley, 2002.

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, search-ing existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments re-garding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquar-ters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. **AGENCY USE ONLY** (Leave Blank)	2. **REPORT DATE** September 2007	3. **REPORT TYPE AND DATES COVERED** Final	
4. **TITLE AND SUBTITLE** Using ArchE in the Classroom: One Experience		5. **FUNDING NUMBERS** FA8721-05-C-0003	
6. **AUTHOR(S)** John D. McGregor, Felix Bachmann, Len Bass, Philip Bianco, Mark Klein			
7. **PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. **PERFORMING ORGANIZATION REPORT NUMBER** CMU/SEI-2007-TN-001	
9. **SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. **SPONSORING/MONITORING AGENCY REPORT NUMBER**	
11. **SUPPLEMENTARY NOTES**			
12A **DISTRIBUTION/AVAILABILITY STATEMENT** Unclassified/Unlimited, DTIC, NTIS		12B **DISTRIBUTION CODE**	
13. **ABSTRACT (MAXIMUM 200 WORDS)** The Architecture Expert (ArchE) tool serves as a software architecture design assistant. It embodies knowledge of quality attributes and the relation between the achievement of quality attribute requirements and architecture design. This tech-nical note describes the use of a pre-alpha release of ArchE in a graduate-level software architecture class at Clemson University. ArchE was used to assist the students in the architecting process. The tool was then evaluated by the stu-dents and instructor. The instructor felt that ArchE met his objectives as a pedagogical tool. The students, although criti-cal of the pre-alpha status of ArchE, were enthusiastic about the benefits of having the step-by-step guide to the architect designing process as provided by ArchE.			
14. **SUBJECT TERMS** Architecture Expert, ArchE, tool, architecture design assistant, quality attribute		15. **NUMBER OF PAGES** 42	
16. **PRICE CODE**			
17. **SECURITY CLASSIFICATION OF REPORT** Unclassified	18. **SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	19. **SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	20. **LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18
298-102