Reasoning about the Architecture for the Clemson Transit Assistance System

John D. McGregor
Clemson University
johnmc@cs.clemson.edu

Introduction

Creating the software architecture for a product involves defining a structure that can accommodate all of the functional requirements for the product and that will facilitate the realization of levels of qualities desired by stakeholders. In creating that structure, decisions are made to separate functions into modules and to wire the modules together in ways that enhance the non-functional qualities. These decisions involve many competing, and some times contradictory, factors. Reasoning about the system to make the correct decision involves estimating the impact of each change on all of the non-functional requirements.

The SAT team at the SEI is developing an automated reasoning assistant to aid architects in making these decisions. ArchE[1] provides the architect with the ability to model the functional requirements and to establish relationships among the requirements. The tool also provides a reasoning engine that can evaluate the levels of certain quality attributes that will be present in products built from the modeled structure. The reasoning engine guides the architect through decision making to refine the structure by suggesting tactics that will enhance the quality about which you are reasoning.

In this example, we will create a modifiability model and a performance model for a personal travel assistant. First we introduce the problem through its requirements. Second, we present the models built in ArchE and describe a sequence of decisions.

CTAS

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. The wireless device allows the traveler to periodically update their information and reconsider their itinerary. The result of using CTAS should be as efficient a trip as is possible given the conditions at the time of travel.

The stakeholders in 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 as cheap as

[1] This document is based on the pre-release version of ArchE. It will be updated as new releases become available.

possible so a wide range of people can afford them. Users want good value and ease of use. Developers want to use reliable and familiar technologies to speed development.

The stakeholders in CTAS have a number of interests. The users of the system are interested in traveling to their destination and will use a system that makes that trip easier provided using the system is not too difficult to understand. The governments of locales served by the system are interested in reducing traffic congestion and generally reducing the impact of travel on other area concerns. The providers of information to the system, such as hotel owners, railroads, and parking lot owners, are interested in having their information accessible to as many travelers as possible as accurately as possible. Most of these providers are also interested in maximizing their revenue.

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

CTAS Actors

The actors in CTAS have a number of differing goals. The CTAS user wants to plan and execute a trip in the cheapest, fastest, or shortest manner possible (Different users will rank these criteria differently). Secondary actors include information providers such as parking lots, 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 expanded by attaching the device to expansion devices. A CTAS device's feature set will have the ability to adapt to a changing set of peripheral devices.

Table 1 - CTAS actors

CTAS user	The user has a few routine destinations to which they will travel repeatedly from a usual origin, e.g. home to work. They need to be able to plan and revise trips on an ad hoc basis. They need easy to understand itineraries that reflect their familiarity with the itinerary.
CTAS information provider	Any actor that provides data to CTAS for use in computing itineraries. A vehicle in which the CTAS is being transported may provide time to destination information. A parking lot may provide its location and availability information.
CTAS smart unit	A special kind of information provider that is a building or transportation vehicle that provides information about itself such as a map or rules or help.
CTAS device	Any device on which an instance of CTAS may be hosted. This may be a dedicated device or a multi-purpose device such as a smartphone or PDA.
CTAS related hardware	An abstract secondary actor. 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. A certain set of preplanned peripherals are automatically recognized and the behavior of the system will adjust to the presence or absence of the peripheral. 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 which can expand the CTAS device's bandwidth range of output devices, etc.

Qualities

Using the ISO 9126 framework we specify the following qualities:

Table 2 - CTAS Qualities

Functionality	Accurate	The itinerary produced by 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 producing faulty results.
	Interoperable	CTAS should be able to accept information from as many information providers as possible. Any standards should be identified and followed.
	Secure	Communication between CTAS and information providers should be reasonably secure but since it is only the availability of resources, security is not a primary concern. Communication between CTAS and the user should be secure.
Reliability	Recoverable	Any itinerary should be persistent even in the event of spontaneous reboot of the system.
Usability	Understandability	The system should be understandable by persons with an 8th 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	CTAS should be able to produce an itinerary within 30 seconds of receiving the command.
	Resource utilization	CTAS should be capable of operating in 256 M of dynamic memory.
Maintainability	Analyzability	A CTAS maintainer should be able to estimate the effort for a requested modification within 4 hours.
	Changeability	A CATS maintainer should be able to accomplish

		most changes within 3 working days.
	Testability	CTAS should be moderately testable.
Portability	Adaptability	CTAS will be capable of being ported to a new device by replacing externally linkable drivers.
	Installability	CTAS should be packaged using an automated installer usable by anyone meeting the usability requirements.
	Replaceability	Upgrading CTAS should be possible using the same automation as for the initial installation.

A Quality Attribute Workshop was conducted and resulted in Modifiability (Changeability) and Performance (Time Behavior) as two of the five highest priority qualities. These will be the basis for the models developed below.

CTAS Use Cases

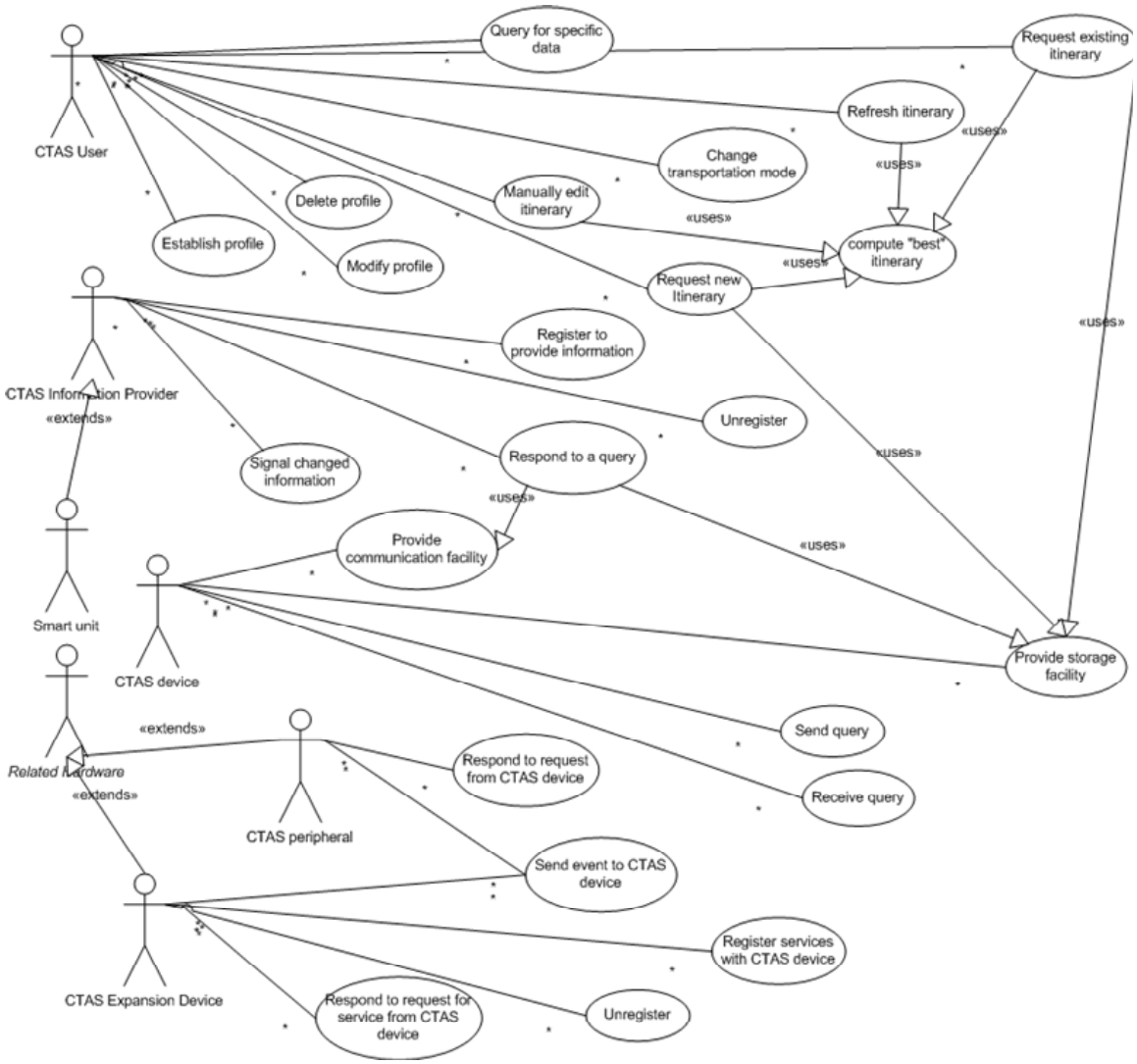


Figure 1 - Use Case Diagram

These uses of CTAS will serve as the starting point for decision making.

ArchE

The ArchE tool is developed on top of the Eclipse 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 in [Bass 2005] is created and then incorporated into the ArchE reasoning engine.

ArchE uses the Jess rule-based inference engine. 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 which it maps onto a set of responsibilities. These responsibilities are associated with one another via various relationships. The set of responsibilities is modified as are the relationships through the decisions of the architect. 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 (but 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 the value of that attribute that should be achieved during the scenario. For example, “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, scenarios, and the mappings among and between these. 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. Therefore, there are almost always many possible sequence 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 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. A set of responsibilities were derived 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 may be too fine grained in some cases.

Table 3 - Example use case

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

4. 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.
5. ArchE will create a corresponding set of responsibilities. Initially it is simply a one-to-one mapping with the functions. (Items automatically generated by ArchE are in green type.)

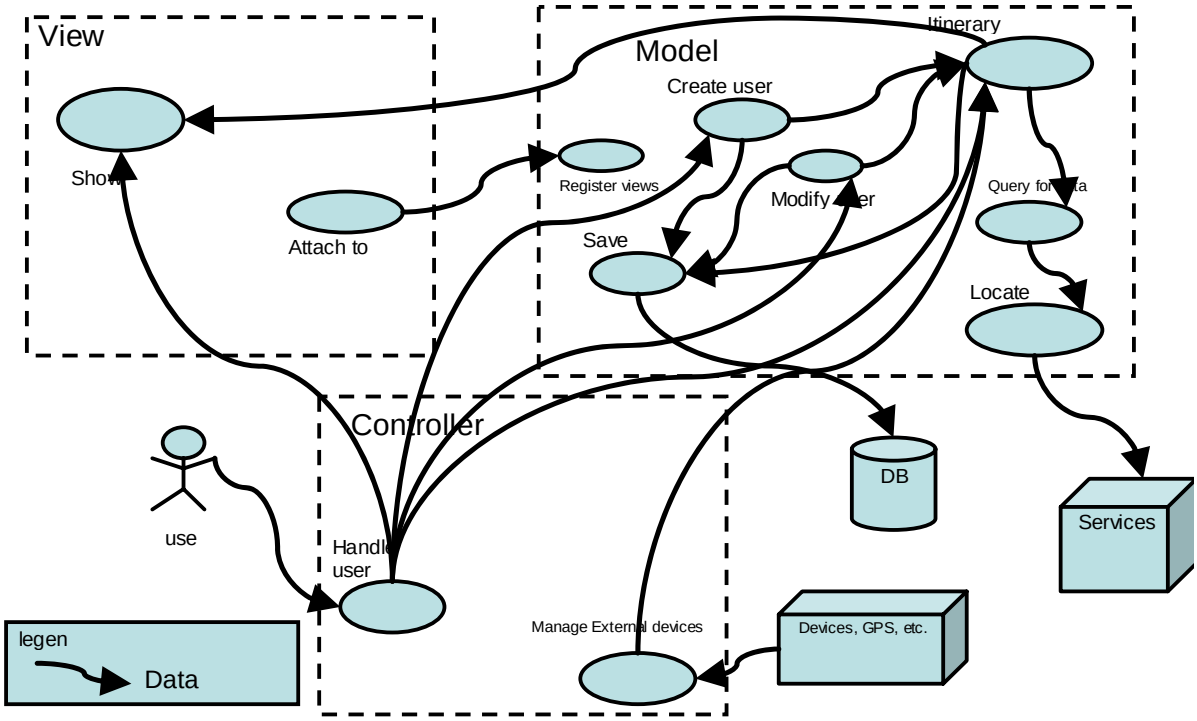


Figure 2 - Responsibility graph

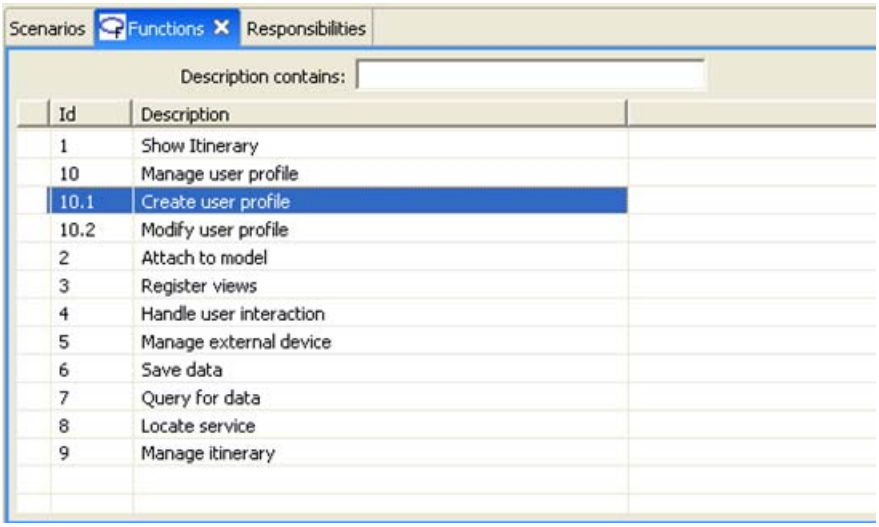
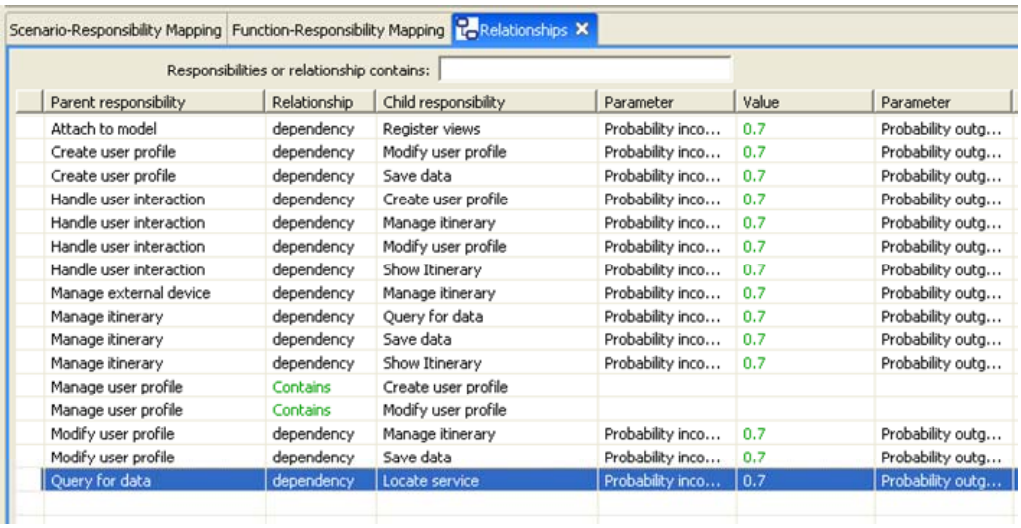


Figure 3 - Functions entered in ArchE

6. Use the relationships pane 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.)



Parent responsibility	Relationship	Child responsibility	Parameter	Value	Parameter
Attach to model	dependency	Register views	Probability inco...	0.7	Probability outg...
Create user profile	dependency	Modify user profile	Probability inco...	0.7	Probability outg...
Create user profile	dependency	Save data	Probability inco...	0.7	Probability outg...
Handle user interaction	dependency	Create user profile	Probability inco...	0.7	Probability outg...
Handle user interaction	dependency	Manage itinerary	Probability inco...	0.7	Probability outg...
Handle user interaction	dependency	Modify user profile	Probability inco...	0.7	Probability outg...
Handle user interaction	dependency	Show Itinerary	Probability inco...	0.7	Probability outg...
Manage external device	dependency	Manage itinerary	Probability inco...	0.7	Probability outg...
Manage itinerary	dependency	Query for data	Probability inco...	0.7	Probability outg...
Manage itinerary	dependency	Save data	Probability inco...	0.7	Probability outg...
Manage itinerary	dependency	Show Itinerary	Probability inco...	0.7	Probability outg...
Manage user profile	Contains	Create user profile			
Manage user profile	Contains	Modify user profile			
Modify user profile	dependency	Manage itinerary	Probability inco...	0.7	Probability outg...
Modify user profile	dependency	Save data	Probability inco...	0.7	Probability outg...
Query for data	dependency	Locate service	Probability inco...	0.7	Probability outg...

Figure 4 - The relationships between responsibilities

7. 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 6. Select the appropriate type of scenario – modifiability or performance - for the scenario in the drop box 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.

Modifiability Model

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

8. 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 Figure 5.

Source

- End user
- Developer
- System administrator

Stimulus

- Add {functionality, quality, capacity}
- Delete {functionality, quality, capacity}
- Modify {functionality, quality, capacity}
- Vary {functionality, quality, capacity}

Artifact

- Interface
- Platform
- Environment
- Other system

Environment

- At runtime
- At compile time
- At build time
- At design time

Response

- Locate place to modify
- Make modification without ripples
- Test modifications
- Deploy modification

Response measure

- Cost in terms of number of elements
- Effort
- Money
- Impact on other modules

Figure 5 - General scenario selection table

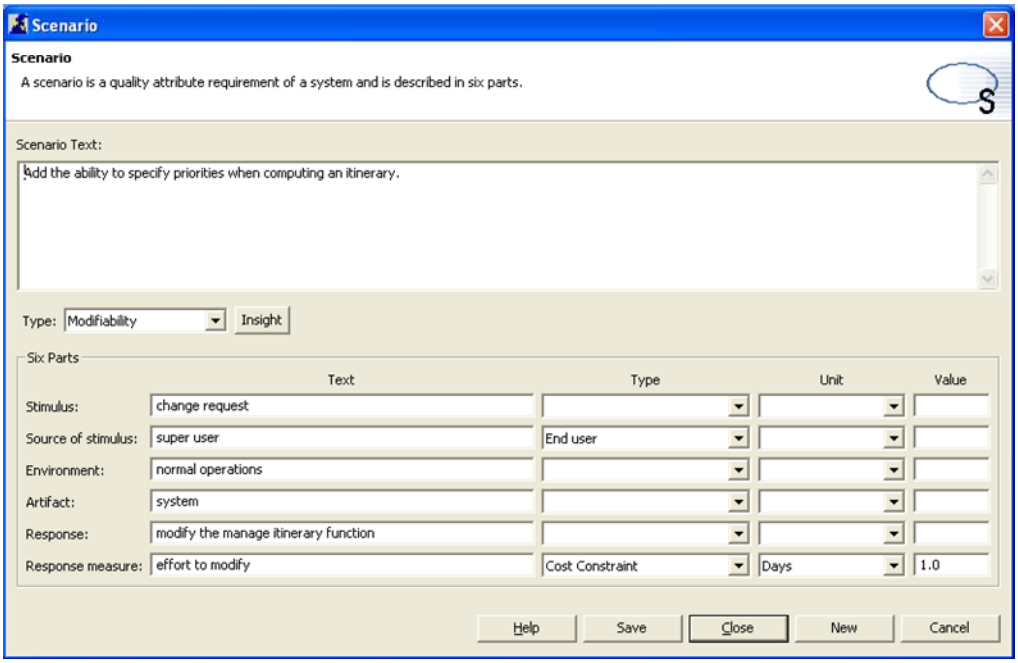


Figure 6 - Scenario entry screen

9. Relate the scenarios to specific responsibilities using the mapping pane, shown in Figure 7 on the lower right. A modifiability scenario would address changes to one or more of the responsibilities related to the scenario or to the relationships among the responsibilities.

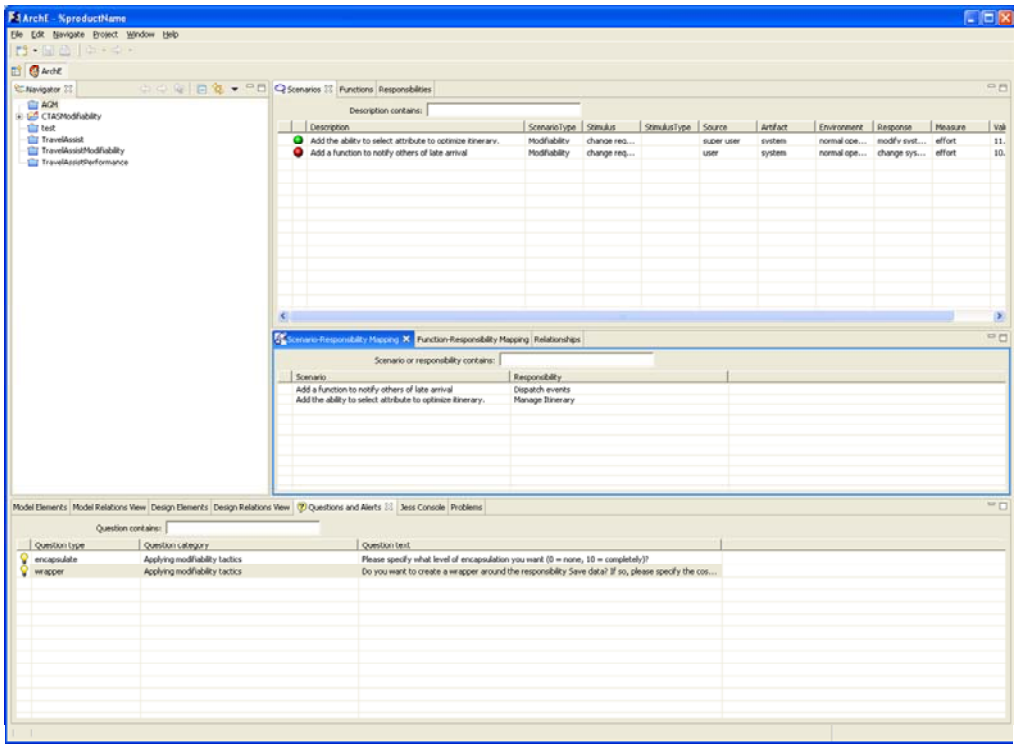


Figure 7 Scenario/Responsibilities

10. In the questions pane at the bottom, ArchE will ask questions to gain the data it needs to reason about the architecture. For 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 or not since the scenarios’ response measure is also in “days of effort”. A green ball to the left of a scenario, in the scenario pane shown in Figure 8, indicates the scenario is satisfied, and a red ball indicates the scenario is not satisfied given the current data.

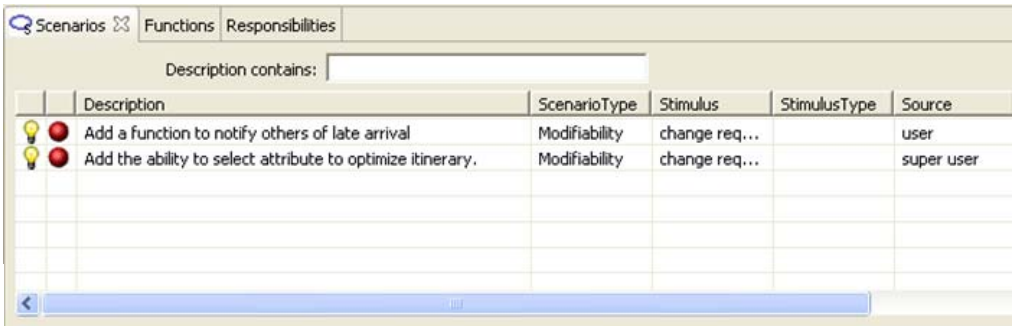


Figure 8 - Scenario pane

11. In the questions pane, Figure 9, 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.”

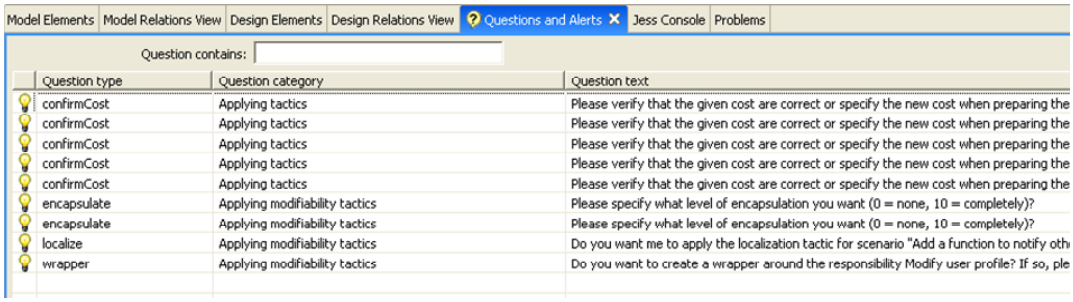


Figure 9 - Questions pane

Selecting the localize tactic produces the dialog shown in Figure 10. 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 the possibility of saving some amount of effort in future changes if the localize tactic is applied. 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 13 and Figure 14 show those responsibilities that are modified. In Figure 13 three responsibilities that have a common dependency each have a smaller oval inside each responsibility representing the portion of those responsibilities that is assumed to be in common. In Figure 14 a red oval represents the new responsibility, which localizes the common responsibility.

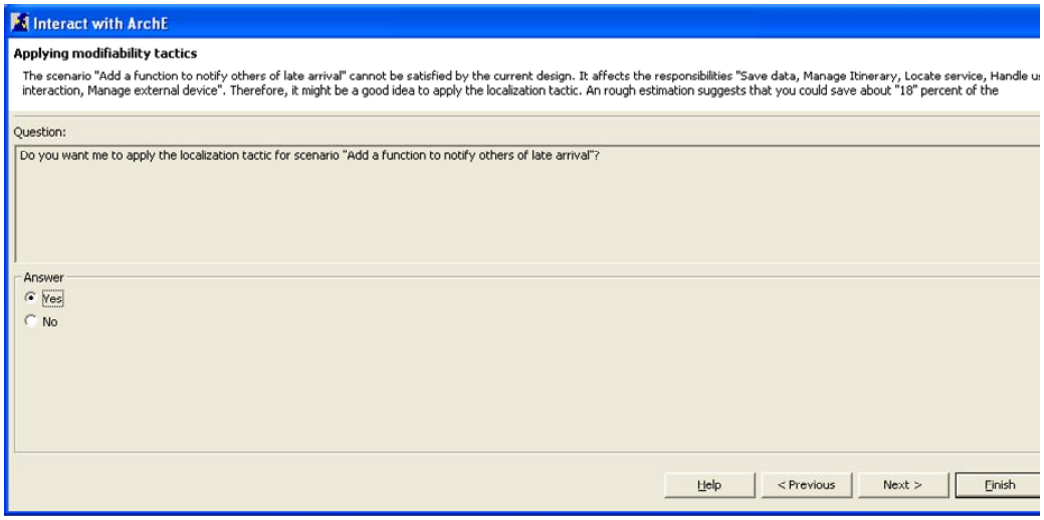


Figure 10 - Applying tactic dialog

Accepting the “Yes” response results in a new responsibility being created, as shown in Figure 11. The architect now manually edits the new responsibility giving it a meaningful name in place of the generated name.

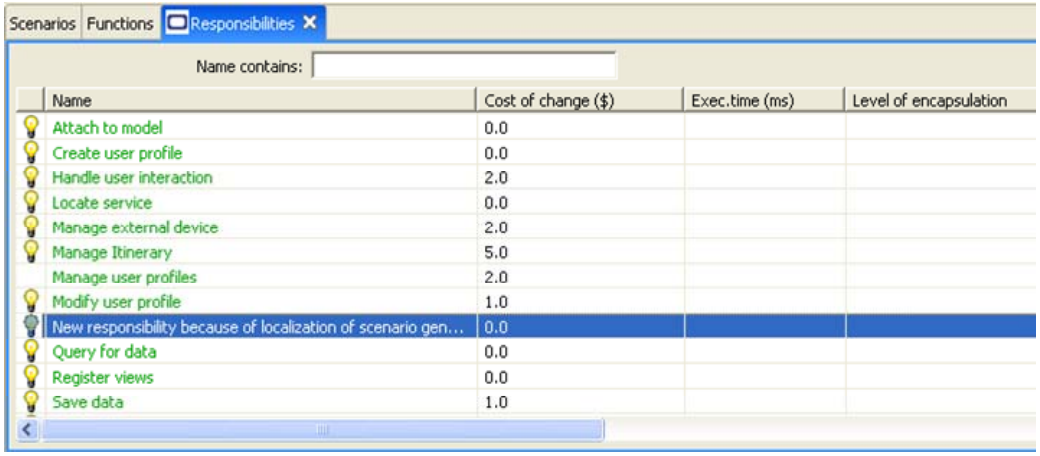


Figure 11 - New Responsibility

That editing results in the new responsibility shown in Figure 12. The new responsibility has been named “Dispatch events.”

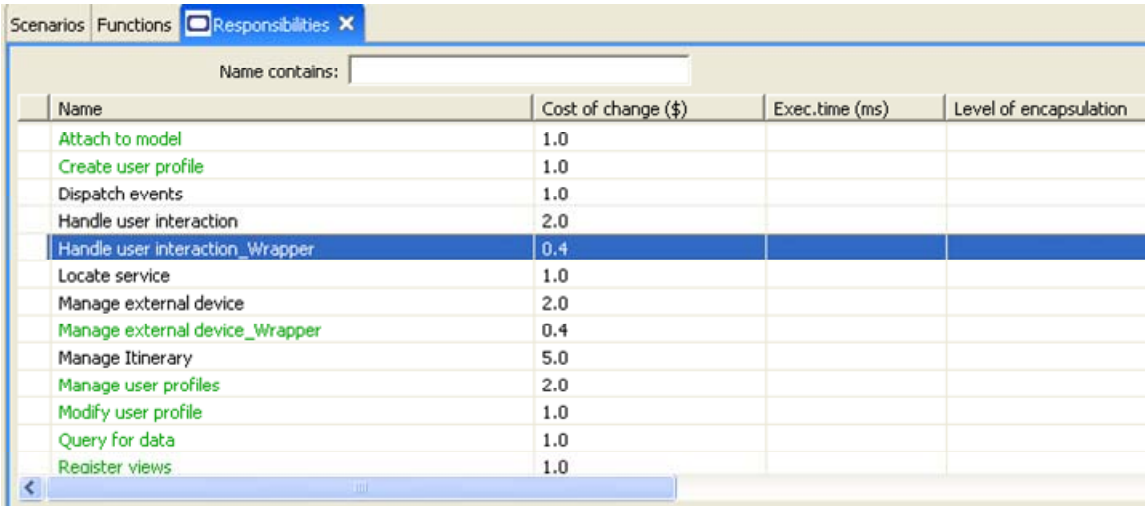


Figure 12 - Edited responsibility

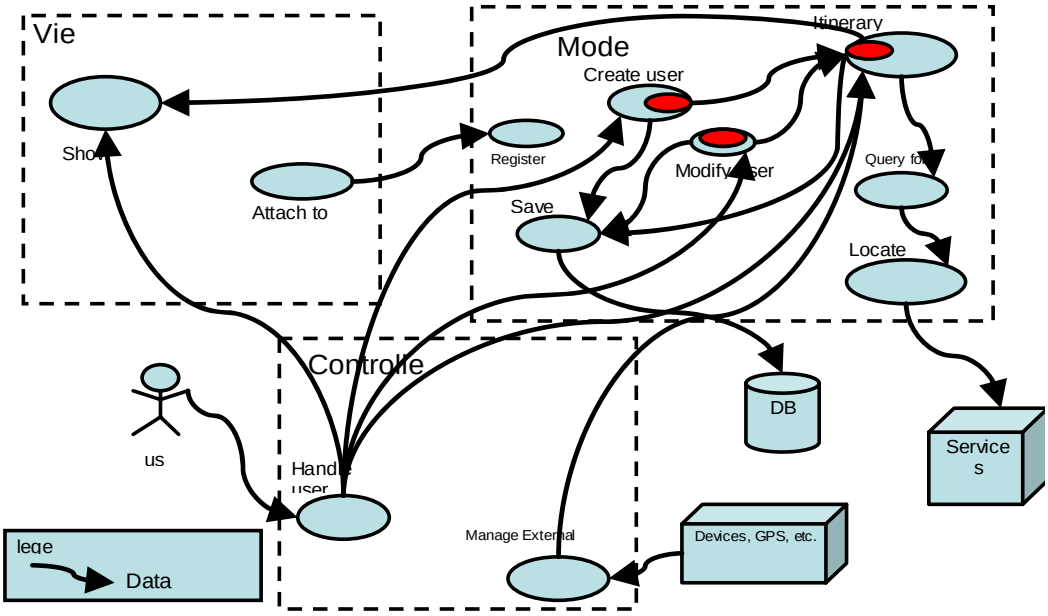


Figure 13 - Identify common responsibility

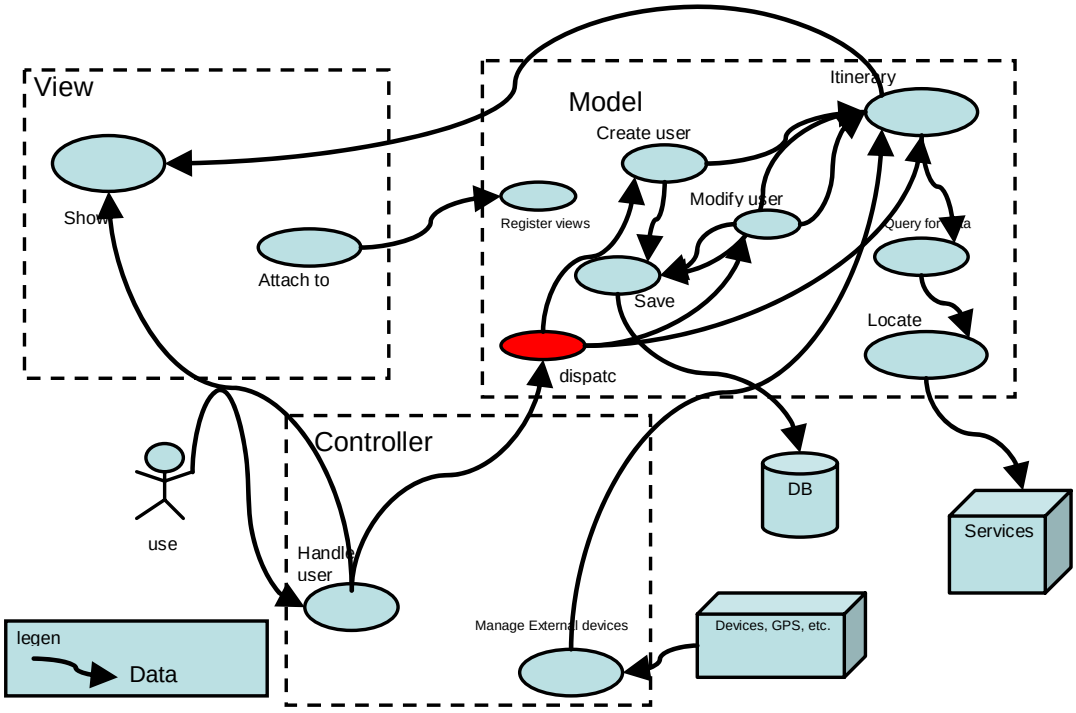


Figure 14 - Revised graph of responsibilities

12. ArchE adds a number of suggestions to the questions pane after the localize tactic has been applied. Many of them, the “confirmCost” and “moveDependency”

suggestions guide the architect to places in the model that may need to be changed based on the new responsibility.

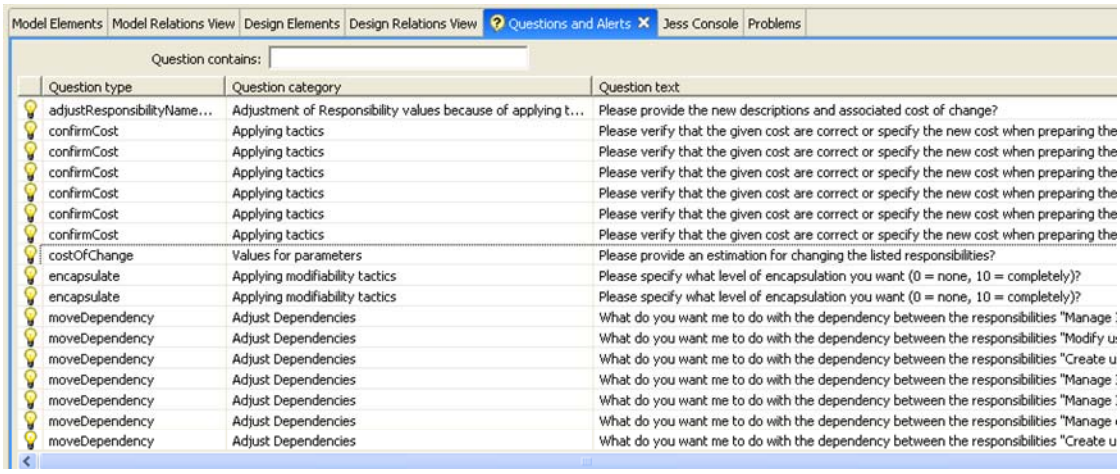


Figure 15 - Suggestions after localize

13. The moveDependency suggestion allows the architect to add the new responsibility into the network of relationships. ArchE makes the suggestion for each responsibility that is affected.

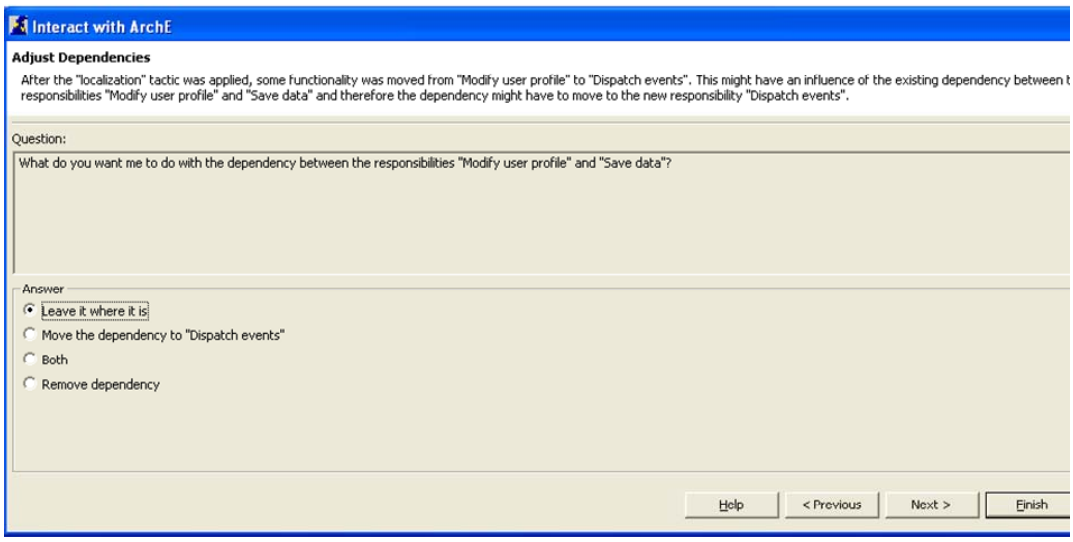


Figure 16 - moveDependency action

14. ArchE is now suggesting that the new responsibility be encapsulated. Figure 17 shows the advice given by ArchE. We elect not to encapsulate, indicated by leaving the level of encapsulation at 0.0, since Manage interface to devices is already a single point and encapsulating will not further enhance the architecture.

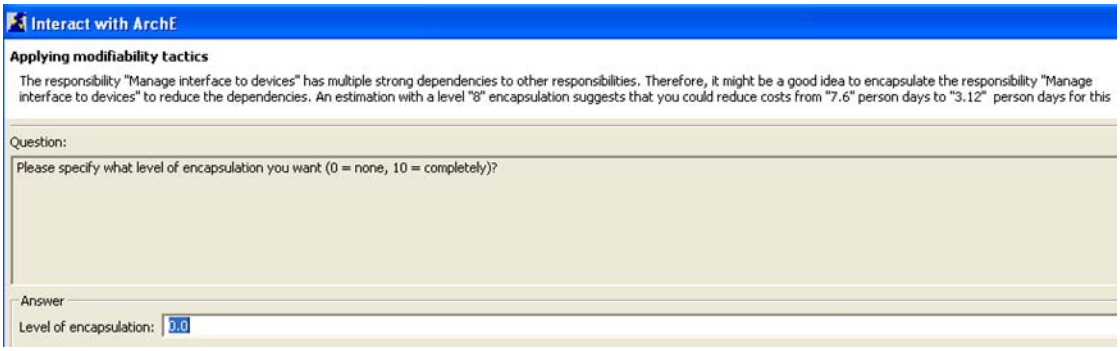


Figure 17 - Dialog about encapsulation tactic

15. Figure 12 illustrates how making the change shown in Figure 10 ripples through the model. This shows the new values on each responsibility and shows the wrappers that resulted from two applications of the "apply wrapper tactic," Figure 18

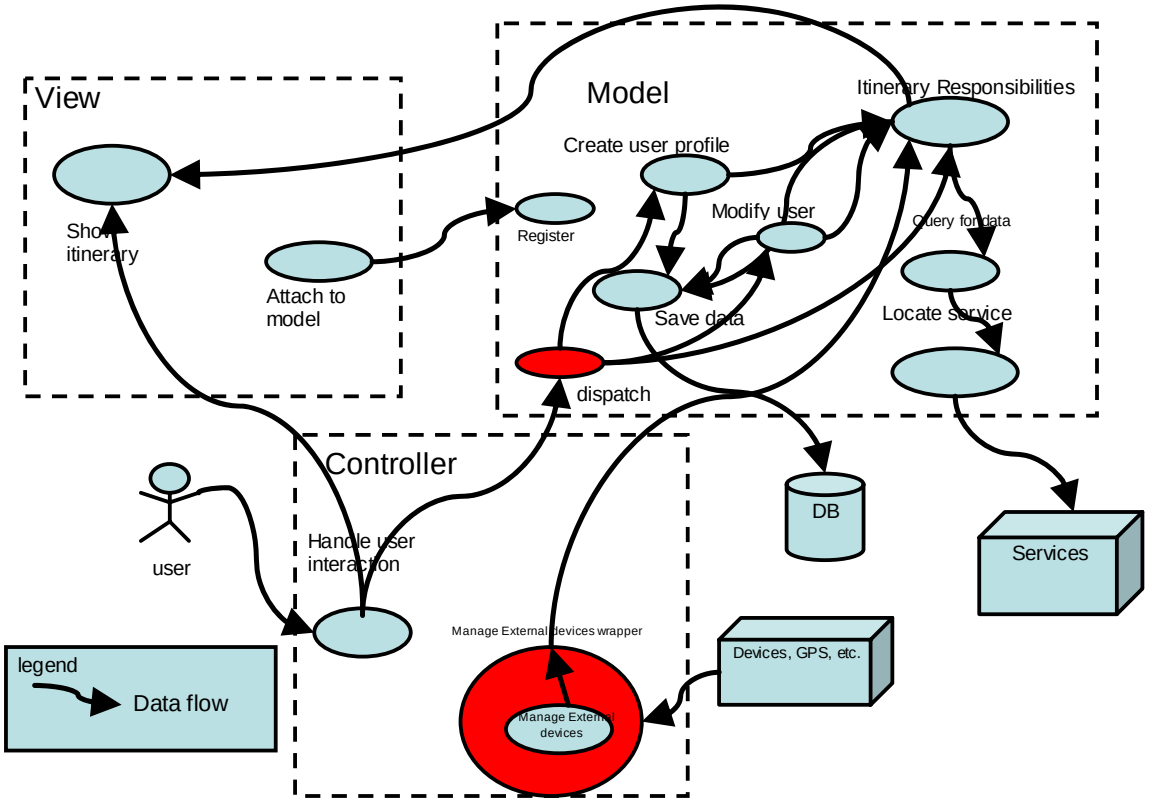


Figure 18 - Application of the wrapper tactic

16. After adjusting responsibilities and modifying the costs of change of responsibilities, 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.

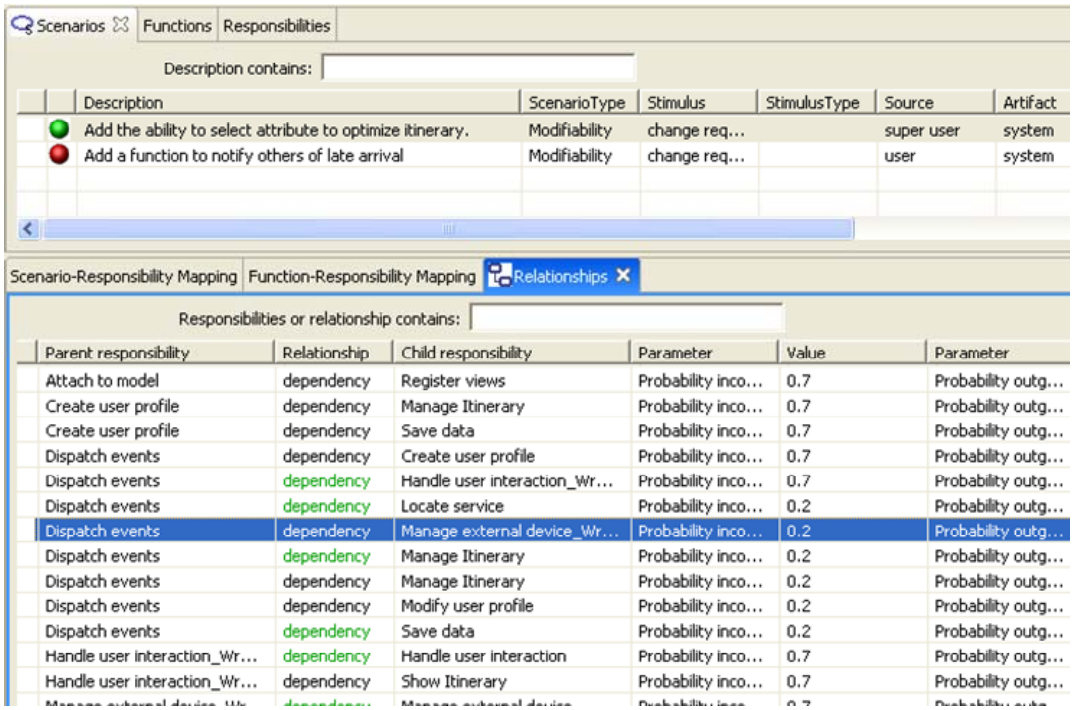


Figure 19 - Probabilities modified

Figure 19 shows that modifying the propagation probabilities of a set of the dependencies associated with Dispatch Events leads to the scenario “Add ability to select attribute to optimize itinerary” becoming satisfied.

17. The goal in the second scenario, shown in Figure 20, was increased until the scenario was satisfied. Its not an elegant solution but it may be the realistic amount of effort.

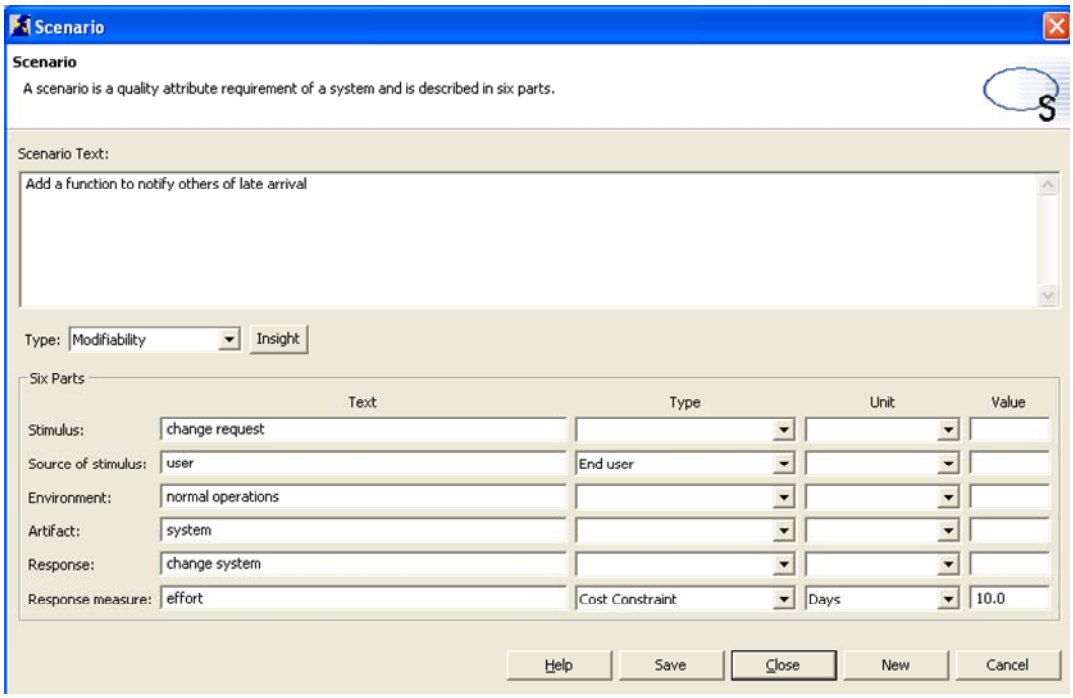


Figure 20 - New status of the model

18. **In the version used to produce this document, the Jess Console provides interesting viewing. You can see which of the rules in the fact base have fired. In Figure 21, you can see that rules 2160 and 568 have fired. By accessing the FactBase file found in the project directory you can 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.**

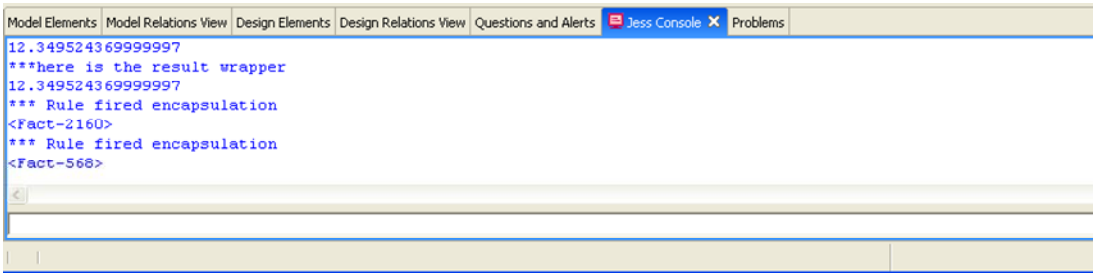


Figure 21- Jess Console

Performance Model

The performance model is formed from the performance scenarios.

And that is your job.

References

Len Bass, James Ivers, Mark Klein, Paulo Merson, Reasoning Frameworks (CMU/SEI-2005-TR-007).

Robert J. Ellison, Andrew P. Moore, Len Bass, Mark Klein, Felix Bachmann, Security and Survivability Reasoning Frameworks and Architectural Design Tactics (CMU/SEI-2004-TN-022).

Jinhee Lee, Len Bass, Elements of a Usability Reasoning Framework (CMU/SEI-2005-TN-030).