

Groove: A Case Study in Adaptive Architecture

Charles Edward HERRING

*Department of Computer Science and Electrical Engineering
University of Queensland, Brisbane, Australia, 4072
herring@dstc.edu.au*

ABSTRACT

The Viable System Architecture is applied to the problem of analyzing and extending an existing software system. This software architecture approach is based on Stafford Beer's Viable System Model. The Groove collaboration system is chosen to illustrate how this software architectural methodology can be used to extend an existing system with adaptive features. The Groove system architecture is mapped into the Viable System Architecture representation. An adaptive user interface feature is evolved using fuzzy control techniques.

Keywords: Viable System Architecture, Viable System Model, Adaptive Software Architecture, Adaptive User Interface, Fuzzy Control, Groove.

1. INTRODUCTION

The Viable System Architecture (VSA) is a software architectural style or approach derived from the Viable System Model. The VSA can be used to design new systems or to analyze existing ones. This paper shows how the VSA can be used to evolve an existing complex software system into an adaptive system in a precise and orderly manner. The new Groove [1], Peer-to-Peer (P2P), collaboration system is chosen for this purpose.

The paper is organized as follows. First, a brief introduction to the VSA is given as this is necessary to understand its application. Next, the Groove "Shared Space Architecture" is mapped into this framework. Based on this mapping examples of how to evolve the Groove architecture into an adaptive system are given.

2. VIABLE SYSTEM ARCHITECTURE

The Viable System Architecture (VSA) has been formally described elsewhere [2, 3]; here a conceptual introduction will be given. Consider the VSA as simply "model-view-controller meets control theory." The VSA architectural style is described from that angle here. The left side of Figure 1 shows the Model-View-Controller (MVC). The controller in MVC got its name from the fact that it is, or was in the original Smalltalk, "controlling" the view. In that role it is an "open-loop" (no feedback) controller.

The right side of the figure shows MVC recast as a closed-loop control system. This organization identifies the *human as the controller* of the model (plant/process) and MVC's controller and view as transducers. That is, MVC's "controller" is a transducer that translates input from the human into commands to the model and MVC's "view" is a transducer that translates model state information into output (feedback) for the human controller. That the system is embedded in an environment is also indicated on the right in Figure 1.

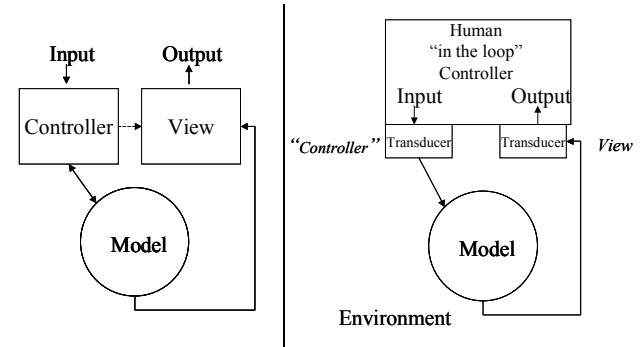


Figure 1 MVC as a Control System

This perspective is useful in that provides a conceptual framework for answering the following questions about adaptive software systems and their architectures:

1. What forces act on the adapting system?
2. What is the system adapting to?
3. What structures are undergoing adaptation?
4. What are the mechanisms of adaptation?
5. What part of its history does the system retain?
6. How are different adaptive strategies compared?

The above questions must be answered in order to make Groove, or any software application, an adaptive system and answering them is subject of the next sections. The framework to assist in answering those questions is developed here. Figure 2 is an enhanced diagram of a control system and illustrates several features central to understanding the approach. The figure shows detailed structure within the controller and within the plant. This is a simplified picture of the Viable System Model [4] on which the approach is based and hence named the Viable System Architecture.

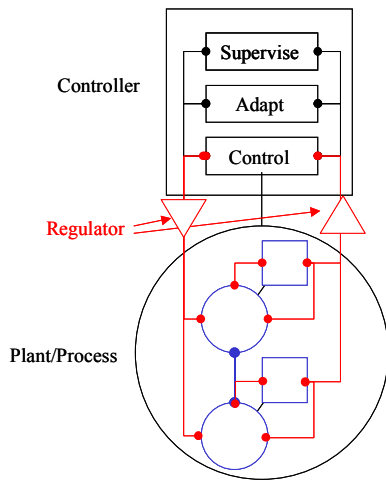


Figure 2 Simplified Viable System Model

The configuration shown in the diagram is a “Supervisory-Adaptive-Controller.” The functionality of the controller is divided into three separate components. The forces that drive this separation are important for software architectural evolution as will be discussed below. The responsibilities of these three components are as follows. First is the function of Control that includes the Regulatory circuit. This is a “closed-loop” controller such as a thermostat used to control room temperature. The Regulatory circuit is distinguished. It is responsible for scheduling and coordination (indicated by the triangle on the right) and for auditing (the triangle on the left). The Controller and Regulator are concerned with the “inside and now” operation of the plant.

Next is the Adaptive component. Adaptation is the behavior of a system to change itself in response to environment disturbances. A system adapts in order to maintain the stability of the plant’s internal variables. The component responsible for adaptation is shown in the figure above of the Controller. This is the component concerned with the “outside and then” operation of the system. There is a range of possible adaptive behavior. The simplest adaptation occurs by changing the strategies of the controller. More complex types of adaptation include changing the structure of the system (reconfiguration) and discarding or acquiring new components (trading). These behaviors are the results of the adaptive component “tuning” the controller and thereby stabilizing the plant.

The Supervisory component maintains the overall goals and objectives of the system in the form of policy statements. This component governs the range of acceptable adaptive and control behavior. It acts as a tuner on the adaptive-controller pair.

Finally, referring to Figure 2, the plant (process, model, software component) is shown composed of more (2 in this case) controller-plant systems. The model is recursive in that each level is made up of similarly structured sub-systems: each system is embedded in a super-system and embeds sub-systems. The particular

specification of the interfaces between systems is key as it *defines the component framework*. In the Viable System Architecture the unique nature of this component and its interfaces define the component frameworks. There is only one component “pattern” and it defines the framework or the interconnections of systems at each level and across levels.

In summary, the Viable System Architecture provides a conceptual model or framework for thinking about adaptive software systems. It is an approach for developing adaptive software systems and *analyzing existing systems* with the goal of evolving them into adaptive systems. Software that results from this approach will be *viable* software systems. Viable software is achieved by realizing the dual and interrelated qualities of (1) adaptive architecture (design-time adaptability) and (2) supervisory-adaptive-control (run-time adaptability).

3. GROOVE ARCHITECTURE MAPPING

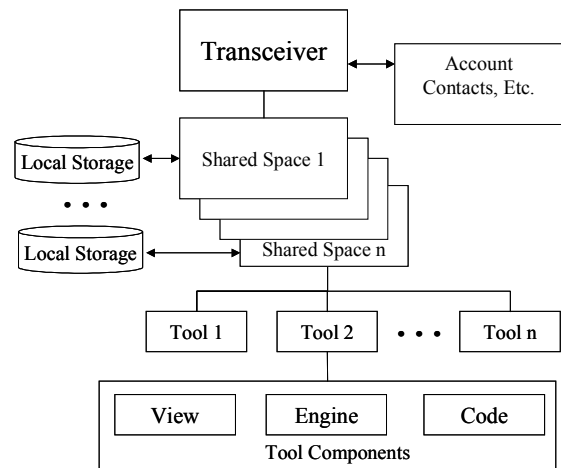


Figure 3 Groove Shared Space Architecture

This section presents a mapping of the Groove Shared Space Architecture into the Viable System Architectural framework. The goal is to show how the approach lends itself to analyzing an existing software system and how it provides an architectural migration path for the introduction of adaptive features. Figure 3 is a representation of the Shared Space Architecture obtained from Groove documentation. It shows the following relationships among the components that make up the user interface and interactive tools. At the top is the Transceiver. This is the shell or overall window within which the user operates. The Transceiver hosts a number of Shared Spaces each with its own local storage. These spaces are the centers of collaboration between users. The diagram shows that a Shared Space contains some number of tools and the internal structure of a tool is also shown. Finally, there is a box to the right of the Transceiver indicating the necessary support functions for maintaining account information, preferences, and contacts.

From the above it is seen that the Shared Space Architecture is a hierarchical architecture and three layers can be identified: Transceiver, Shared Spaces, and Tool Sets. The subsections that follow apply the Viable System Architecture to each of these layers. Each subsection is organized as follows. First there is an architectural drawing of the layer mapping the Groove components into a VSA diagram. The relationship of the components to each other and the framework is described.

Transceiver

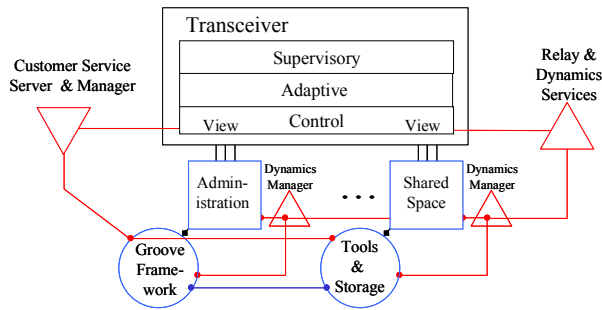


Figure 4 Transceiver and Shared Spaces

The Transceiver is shown in Figure 4 as the top-level controller of a number of Shared Spaces as well as providing Administration functions for the Transceiver. Shared spaces are Groove’s distributed collaborative work areas. A shared space is shown as the controller for Tools and Storage and described in the next section. A shared space’s View (an ActiveX control, shown inside the Transceiver controller) is the user interface presented by the space in the Transceiver. Each shared space is shown as having a Dynamics Manager and to the right of the Transceiver is the overall Dynamics Service. The Dynamics Service is one part of the “mediator” component in Groove’s “mediated” MVC architecture. Dynamics is responsible for recording and sequencing commands generated by local user actions and playback of commands received from remote Transceivers. The other part of the mediated MVC is the Relay Service. It provides the communications and coordination mechanism to join Transceivers into a peer-to-peer system and several other functions. These functions are described in the section on Web Services and Groove.net below.

Customer Services is shown on the left of the Transceiver in the role of the auditor. Customer Services is composed of the Customer Service Manager (CSM), Customer Service Server (CSS), a Watchdog, and Event Adapters. The CSM, which runs on the client system, is a set of interfaces used for reporting significant events to a CSS. The CSM also tracks application and product usage to gather information so that Groove can optimize a user’s experience. The watchdog is responsible for starting, restarting, and terminating Groove system processes. Event adapters process events received from the CSM.

The CSM can interact with the component manager to download and install new versions of components, and to undo installation if necessary.

Finally, a special sub-system of the Transceiver is Administration. This includes all the features needed by users to run and maintain the system. Included here are Instant Groove, UI Services, Accounts and user identification. This system is the “shell” that lets the user interact with shared spaces.

Shared Spaces

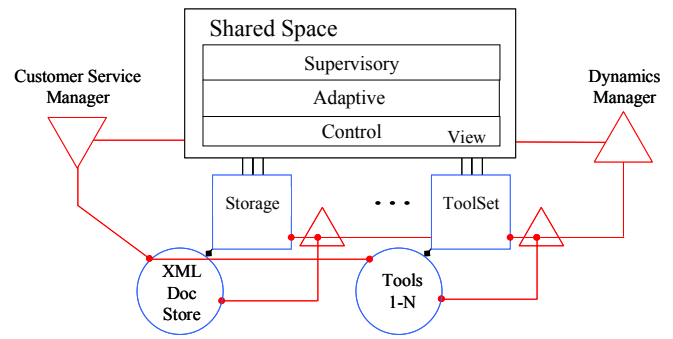


Figure 5 Shared Space with ToolSet and Storage

Figure 5 shows the organization of a Shared Space. Each shared space has one or more Tool Sets and one XML document store. A tool set is a logical grouping of tools used for some purpose. The purpose of tool set is to allow a group to work on more than one type of project within a single shared space. That is, to have a set of tools specific to a given project or task. Tool sets are recursive in that they may contain tool sets. In terms of the view, tool sets present a container window with tabs across the bottom that permits the user to select the different tools.

The XML document store maintains information on all aspects of a shared space. Every shared space on a device has a dedicated local Dynamics Manager, which maintains a deterministic, distributed consistency model for all changes to the shared space. The Dynamics Manager is responsible for executing changes to shared space data requested by tools. It is the mediator in Groove’s mediated model-view-control architecture. The Dynamics Manager also provides automatic recovery in case of communications failures. The goal of Dynamics Services is to coordinate and manage changes made both locally and remotely (on other shared space members’ devices) to preserve the user’s intent. It coordinates, executes, receives, and transmits changes to a shared space’s data model to each member in a deterministic fashion, so that each member’s version of the data is the same.

Tools Sets

Figure 6 shows the organization of a Tool Set within a Shared Space (Figure 5). A tool set is a logical grouping of tools used for some purpose. The primary purpose of tool set is to allow a group to work on more than one

project within a single shared space. This means that a single group of people could use different sets of tools for different projects. Tool sets organize tools into a container window with tabs across the bottom that permits the user to select among the different tools. The tab also shows in parentheses the number of members currently using the tool and hovering shows the names of the members. The diagram above shows one local XML document store. There is only one store per shared space as shown in Figure 5, but it is also shown on this diagram as each tool communicates directly with the document store.

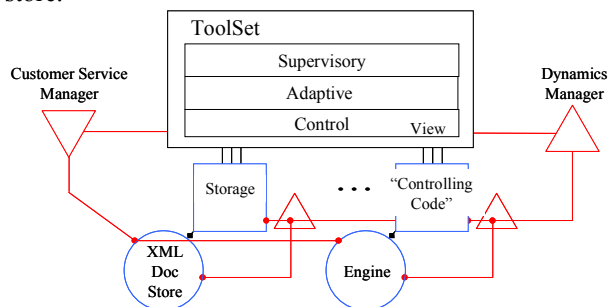


Figure 6 Tool Set and Tools

Examples of tools include notepad, sketchpad, calendar or even a game. The components that make up a tool are the View, Engine and the “Controlling Code” or Glue that binds them all together into a tool template. A view is one or more ActiveX controls that present an interface to a user. The view takes user input and requests the engine to generate a “delta” to represent that input.

A tool’s engine is responsible for maintaining and changing the tool’s stored data model. An engine creates and executes deltas on behalf of the tool’s view and acts on “deltas” (the unit of change) received from other shared space member tools to update the local tool’s state. A tool can subscribe to another tool’s view, but does not need to provide a view. The engine updates the view. Tools use the Document Store for persistence as shown above. Tools work with the Dynamics Manager. A tool requests a delta from the dynamics manager, and uses the supplied delta as a container for its engine’s commands. When the tool has completed writing the engine’s commands into the delta container, it submits the delta to the dynamics manager for execution and eventual dissemination.

4. ADDING ADAPTIVE FEATURES

This section shows how the Viable System Architecture approach leads to viable software, namely, the ability to support architectural adaptation and evolution toward supervisory-adaptive runtime control. An example of implementing adaptive features in the Groove system is presented.

Suppose the users of the system request a dynamic interface that adapts to their usage patterns. In particular they want a user interface that gets simpler over time as

features that are infrequently or never used disappear and those that are frequently used are retained and promoted. For example, within a tool set, if a user infrequently uses a particular tool it will begin to fade away eventually disappearing from the tools set’s view. The tool is still there it is just not shown. Of course, the user can find tools that have been “disappeared” and bring them back onto the tool set’s tabbed menu. The expected benefit from this is a simplified, decluttered interface that reduces cognitive overload and presents a stream lined user experience.

Also, a major consider is that Groove is inherently a multi-user system. Accordingly the design must consider supporting the preferences and behaviors of multiple users across adaptive shared spaces. This will result in coevolutionary interface feature filtering within the shared spaces. Advanced capabilities might include support for spawning sub-spaces based on usage and analysis of inter-space relationships leading to merging of shared spaces.

The following sections describe an approach to implementing the capabilities described above. The mappings developed in the last section will be used to help answer the questions about adaptive software originally posed. The goal is to migrate aspects of human control knowledge into the software as indicated in Figure 2. The architectural drawings in Figures 3-6 show where this control knowledge must be placed: in the control and regulatory circuits.

The Transceiver’s Administrative sub-system provides a shell or browser that permits the user to access shared spaces administer the system. The user interface features of the Transceiver that will be under adaptive control are described. The main menu items for using the system are shown across the top. They are the Forward, GoTo and Backward navigator buttons, Home, New Space, My Spaces, My Contacts, My Account, Groove Services and Help. The navigator buttons let the user move backward and forward between previously visited tools. These are views of shared spaces or administrative function. If the user presses the GoTo button a pop down menu gives access to most of the functions at the top of the browser as well as the user’s shared spaces. The exceptions are New Space, Groove Services and Help. There are small buttons on each side of the GoTo button that “remember” the last menu option selected. That is, they record the last action and provide drop down menu selections that correspond to the Backward and Forward (arrows) buttons. In the central frame, the buttons labeled “Start a ...” provide the user a one-click way to create a new space. Also in that frame is a display of the user’s current spaces, “My Spaces” that permits the user to go directly to a space. Finally, Groove places an icon in the system tray (the task bar at the bottom of the Windows screen). The menu items listed there are Instant Groove, Open Transceiver, Account Preferences; New Account, Suppress Notification, Work Off line and Shutdown

Groove. These are the same options as on the Grove Services button on the opening window.

Given the above description of the user interface and the goal to implement adaptive features, the next step is to develop requirements for the desired adaptive behavior and models on which this behavior will be based. Answering the fundamental questions about adaptation will help in this:

1. What forces are acting on the adapting system? In the case of the adaptive user interface, the human is in the role of both the controller and the environmental disturbance as was mentioned in the introduction to these sections. Unlike autonomous system acting on behalf of the user where only high-level goals are specified, here the human is still in the loop. Also, as Groove is a multi-user system, the actions of other users will influence the evolution of the system.
2. What parts of the environment, embedding super-system and embedded sub-systems is the system adapting to? The environmental disturbances are the user, other users in shared spaces and developers that update or inject new components into the system.
3. What structures are undergoing adaptation? The structures are the Transceiver main window and its menu items including the system tray menu.
4. What are the mechanisms of adaptation? The mechanisms of adaptation are the adaptive and supervisory tuners added to the system. The adaptive tuner will implement a model of the Transceiver interface and the supervisory tuner will permit the user to specify preferences that governing the adaptation.
5. What part of the history of its interaction with the environment does the system retain? It is possible to store every interaction (e.g. menu item selection, window resize) of the user. This is the raw data needed by the adaptive tuner to implement the adaptive interface features. A trajectory of each user interface aspect or feature touched during each session can be stored.
6. How are different adaptive strategies compared? There are two approaches, one is to implement the adaptive features and test on users, and the other is to run the system as a simulation to search out the range of possible evolutionary developments. Both can be used. Doing a simulation first is a good approach as it provides immediate feedback and may uncover errors.

Now, it is clear that a model of the Transceiver sufficient to implement the adaptive strategies is required. The model to support the adaptive user interface feature includes the following elements of the Groove Transceiver Opening Window:

1. Main Menu Items
 - a. Navigator buttons: Forward, Goto (and side buttons) and Backward
 - b. Home

- c. New Space
- d. My Spaces
- e. My Contacts
- f. My Account
- g. Groove Web Services
- h. Help

2. Central Frame
 - a. Graphic Shared Space Creation Menu
 - b. Shared Spaces List

3. Groove system tray menu.

The model will include the geometrical layout of the opening window and position of the above items. Each time the user clicks on one of the items a record will be created to store the event. The information about the event includes identification of the menu item accessed, the menu group it belongs to, date and time, a number for each Groove session (a number identifying the sequence of Groove program runs), device type and the item's position on the window. This set of event records provides the historical data for implementing the most frequently/infrequently used adaptive interface strategies. A function will be associated with each item that determines the state based on the historical data. An item can be in one of five states: Promoted, Bold, Normal, Fading or Invisible. The user will have the option, under My Account, to set the rate of evolution. (Perhaps this will take the form of a new "skin.") The user specifies the rate by a slider button and the minimum setting means no evolution. Each user-generated event causes the adaptive tuner to evaluate the state of the model and update the display. Next, a strategy is needed that relates the user set rate of change, Rate (R), and the user generated event data, Usage (U), to update the state of the menu items, Visibility (V) in the Transceiver window. Fuzzy control can be used to build the adaptive tuner and an example of that approach is now given.

Figure 7 shows three fuzzy set membership functions. Usage, $U(t)$, is a number calculated from the user-generated, stored, event data associated with each menu item. The top graph shows how the fuzzy membership function μ "quantifies the certainty" that $U(t)$ can be categorized linguistically as being Never, Infrequent, Frequent or Always. This categorization is called "fuzzyfication." It is up to us to define the function $U(t)$. For example, $U(t)$ for a menu item can be calculated based on the number of times the time has been accessed, and the minimum and maximum number of times all menu items of the group have been accessed to yield a number between -1 and 1. This number is then fuzzyfied into a linguistic value. Similarly, a derivative, or time rate of change, of $U(t)$ based can be defined based on the historical event data. Given the $U(t)$ and $U'(t)$ and their fuzzy membership functions a rule base is built and used to update the state of the menu items. The rule based is a set of rules of the form: **If** Premise **Then** Consequence. For example:

If Usage is Average **and** Change in Usage is Zero
Then Visibility is Normal.

The rule base will take the form of a look up table of numbers; there may be up to 25 (5^2 : Never ... Always by VerySlow ... VeryFast) rules in the table. More than one rule may be "on" at a time. This is the case when $\mu_{premise}(U(t),U'(t)) > 0$. Determining which rules are on is called "matching." After matching, the next step is called "inference" where the conclusions of all rules are determined. In this case, the result will be a linguistic value specified according to the bottom graph, Invisible... Promoted. Finally, the linguistic value is "defuzzified" into a "crisp" number that is used to set the visibility value of the menu item. The process is

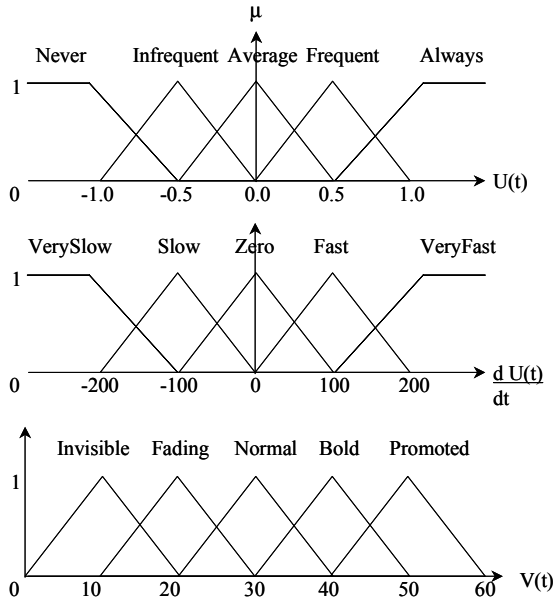


Figure 7 Fuzzy Set Membership Functions

illustrated in Figure 8.

Next the user adjustable rate of adaptation, R , is incorporated into the controller. One standard way is to tune the controller by introducing "gain" on the inputs, $U(t)$ and $U'(t)$, and/or on the output $V(t)$. In this approach three constants are derived from R : g_0, g_1, h . Using these numbers the input becomes: $g_0 * U(t)$ and $g_1 * U'(t)$; and the output becomes $h * V(t)$. The effect of adding gain terms is as follows: for $g=1$ there is no change; for $g < 1$ the membership functions are uniformly spread out – the rate of adaptation will slow down; and for $g > 1$ the membership functions are uniformly contracted and hence the system will adapt faster. For scaling the out using h , the inverse is true, i.e., if $h < 1$ the membership functions are contracted. R can be used to tune the membership functions themselves, not just their scale, but also their shapes.

Having completed the design and implementation of the adaptive tuner, the system is ready for test and evaluation. Analytical, simulation and human trials can be used. Fuzzy control theory provides many tools for analysis of the controller. In the case of Groove, simulation is possible based on profiles of human usage.

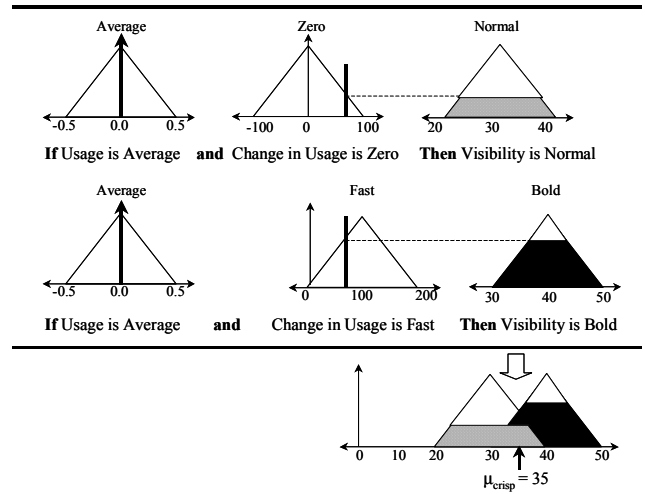


Figure 8 Graphical representation of fuzzy inference

Profiles can be obtained by collecting real usage data. A software surrogate for the human controller can be built that feeds the recorded interface events into an instance of the adaptive Transceiver. The analyst can then view the evolution of the interface and interact with it in a step-by-step fashion or run it to the end and view the final state.

5. SUMMARY AND FUTURE WORK

The adaptive user interface described can be thought of as an Agent-like assistant that watches the user and dynamically changes the control surface presented to him. The purpose is to make the overall performance of the system somehow better by adapting to the preferences of the user. This is one evolutionary step in advancing the Groove system. This paper shows how the VSA can provide a way forward based on the idea of software viability.

6. REFERENCES

1. Ozzie, R., *Groove*. <http://www.groovenetworks.com>, 2000.
2. Herring, C. and S. Kaplan. *The Viable System Architecture*. in *Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34)*. 2000. Maui, Hawaii.
3. Herring, C. and S. Kaplan. *The Viable System Model for Software*. in *4th World Multiconference on Systemics, Cybernetics and Informatics (SCI'2000)*. 2000. Orlando, Florida.
4. Beer, S., *The Viable System Model: Its Provenance, Development, Methodology and Pathology*. *Journal of the Operational Research Society*, 1984. **35**(1): p. 7-25.