An Analysis of the Developer-User Feedback Loop in "The Edge"

by

Peter Weng

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2003

Author . . . . . .
Department of Electrical Engineering and Computer Science
April 28, 2003

Certified by . . . . . .
Harold Abelson
Class of 1922 Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

An Analysis of the Developer-User Feedback Loop in "The Edge"

by

Peter Weng

Submitted to the Department of Electrical Engineering and Computer Science
on April 28, 2003, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

As software becomes increasingly complex and development teams grow larger and larger, developers find themselves more disconnected from the users of the software and their varying needs. This is primarily because developers lack the time to develop relationships with their users in order to fully understand what they are looking for in the software. This thesis describes The Edge, a server/client platform for developing web service applications that monitor and present useful data to users.

The interesting component of The Edge is the user feedback mechanism built into the platform. It provides developers with first hand knowledge of how users are using the software by collecting important usage statistics. This document analyzes the developer-user feedback loop mechanism in The Edge and discusses how it may be used by developers to improve software by focusing on user needs.

Thesis Supervisor: Harold Abelson
Title: Class of 1922 Professor of Computer Science and Engineering

Acknowledgments

I would like to thank Eric Carlson, Dave Mitchell, and Jeff Baxter of Microsoft for helping develop the code for this project. I would also like to thank them for their hard work and invaluable advice and direction they provided me, without which this thesis would not have been a success. I would like to acknowledge the help, advice and encouragement of professor Hal Abelson not only during this project but throughout my MIT career. I would also like to thank Elaine for spending the time to proof read and make corrections on my thesis. Also, without the funding provided by the iCampus MIT-Microsoft alliance, this project could have never existed. Finally I would like to thank my mother for her unconditional love and support for me in everthing I do.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Software has quickly become an essential part of businesses as well as our everyday lives. As computers get faster and more powerful, software will continue to expand their feature set and become more and more complex which widens the gap between developers and users. In order to keep software focused on the needs of users, there must be an efficient yet thorough process for the developers to obtain feedback from users.

However there are a few challenges in managing a piece of software and keeping it tightly coupled with its user base. First is the inherent challenge in finding out what the users want, what they would prefer changed, and how they use the software. Currently, this is done through user feedback, which typically involves monitoring message boards for user comments, distributing paper or electronic surveys to a potential user base, or simply making a guess at what features users might want in the next release. Once enough knowledge is gathered about what users really want in the product, this information has to be communicated effectively to developers and product managers.

The Edge is a research project that takes a novel approach to solving this problem. At its core, the Edge is simply a piece of software that provides a shell to host web service "parts" that monitor changes in data, such as stock quotes or local weather

data. The Edge is interesting in how well it mimics real software projects. In a typical software project, the first version of the project will be released to the public, adopted by a number of users, and then the development team will begin work on the next version. In the same manner, each of these web service parts, which are developed by users, may be consumed by a large number of users, and each of these parts will likely go through revisions.

Additionally the Edge provides a mechanism which allows combining each of these steps into a single programmatic process. The Edge has built user feedback mechanisms into the software which help developers not only quickly determine what features users rate highly and what missing features users want the most, but it also communicates this information programmatically and on-the-fly to the developer so there is no time lost in the communication process. This thesis examines how the idea of such a feedback mechanism can be useful in improving the efficiency of the software development process.

1.2 Description

XML Web Services have quickly become the standard way to pass data across the internet programmatically. As Web Services become more and more popular, the vast amount of data on the internet will become more and more accessible. The Edge takes a novel approach at bringing useful data to a client's desktop.

The Edge is a piece of software that provides a platform for web service development on a client desktop. It allows developers to build "Parts" which consume Web Services to monitor changes in data and present them to the user. This data can be anything the developer chooses. Some examples might be stock quotes, number of unread emails, or weather data.

The Edge can be thought of as a container for all these parts that run on the client desktop. Not only does the Edge shell provide some added UI functionality for the parts, most importantly it supports the parts by providing a platform which allows users to rate each part, submit bug reports, provides each part with a discussion area,

and gathers usage statistics for the part. The novelty of the Edge is the ecosystem that it creates. The added functionality supplied by the edge provides part developers with important information concerning how users feel about the part and promotes software revisions. As the cycle continues, parts will become more and more user-centric which increases use and popularity.

1.3 Overview

In this chapter I have presented the idea of a direct user feedback mechanism and motivation behind this project. I have also briefly described a user feedback mechanism which will form the basis of how developers can potentially analyze this data to determine specific needs of users and ultimately better prioritize feature development to cater to the user community. Chapter 2 will present a typical usage scenario which will help to anchor the rest of the discussion. Chapter 3 will discuss the design and implementation of the Edge as well as an in depth description of the user feedback mechanism provided by The Edge. Chapter 4 will discuss the type of data collected by The Edge, the methodology and the heuristics behind the analysis of this data, and how it can be used to improve software revisions. Chapter 5 will discuss related systems and processes for user feedback analysis and chapter 6 will present conclusions.

Chapter 2

Extended Example

In order to better understand the feedback mechanisms in the Edge, it may be useful to examine a typical usage scenario. Let us take for example a developer Eric writes a Weather part, which tracks the current temperature for the Boston area. Once Eric completes the part, he publishes it to the Edge site, thus making the part available to others.

A group of users install the part and begin to use it. Those in Boston love the part and rate it highly since it provides them with useful information. Others install the part, notice that it only provides weather information for Boston, and quickly uninstall it. They immediately go to the weather part webpage and leave comments for the developer, Eric, saying that the part would be more useful if the zip code information was configurable so that the weather data could be collected for other areas.

Now as Eric begins work on version 2 of the Weather part, he can review the feedback from his users to prioritize the features which are important to them. Eric first looks at the direct comments of the users and notices a message forum which discusses the need for an option allowing users to configure which zip code the weather data pertains to. Eric notices this forum is receiving a lot of activity, thus concludes that this feature is important to the user base. Additionally Eric takes a look at the usage statistics posted on the part webpage and notices that 80% of the users who installed the Edge in fact uninstalled it the same day. This leads him to conclude that

the majority of users might be installing the weather part, but are quickly turned off by the fact that the data provided is of no use to them since they are outside of the Boston area. With this new configurable zip code, hopefully the overall popularity of the weather part will increase.

After Eric finishes implementing the configurable zip code, he thinks about what other options might be interesting for his users. He's already looked at the direct feedback of the users, which comes in the form of comments and discussion postings. The Edge feedback mechanism also provides Eric with important data for him to infer what features might help to increase usability of his weather part. Eric goes to the part webpage and again examines the usage statistics provided by the Edge and notices that the majority of the users who have the weather part installed run it in the minimized state (running in the Edge command bar with only a single indicator value, in this case the temperature, visible). This gives Eric a clue that perhaps the weather part isn't providing enough information to the user to make it worth maximizing. Therefore Eric decides the next feature he will implement is to present more data to the user, such as barometric pressure, relative humidity, and precipitation.

As the next version is released to the public, Eric can begin to look at the data provided by the feedback mechanism to see if there is an increase in the number of users, or an increased utilization of the part for current users. As this cycle of users providing feedback and Eric improving the part continues, the software will become better and better and the feature set will more accurately reflect what the users want.

Chapter 3

Design and Implementation

3.1 Overview

The Edge contains three main components, the client, server and web community. The client component runs on a user's desktop from the windows taskbar. Its main function is to provide the user interface container for the parts the user chooses to install. While running, it can give important notifications to the user which is provided by the installed parts. Figure 3-1 shows the Edge client running on a user's desktop with three parts installed. The client provides the UI for minimizing and maximizing installed parts.

Figure 3-1: The Edge client with three parts installed.

The Edge however is more than just a container or a nice shell to house these parts. More importantly it also provides additional functionality to parts that allows part developers to gain in-depth knowledge of how well-liked their software is in the user community. The Edge also contains a backend server which is pivotal in data collection and analysis for providing developers with usage statistics. In addition, the server keeps aggregate logs of client installations and how they interact with the

Edge. This mechanism will be the basis of a tight feedback loop between Edge users and developers.

The last component is the user community web site which provides an easily accessible location for users to exchange ideas, share comments, and submit bug reports to part developers. For every part that is ever installed to the Edge, a web community is automatically generated which provides message forums, part announcements, and usage statistics for that specific part. Figure 3-2 shows an example of an auto-generated web community for the Lab Status part. The user community web site is central in the promotion of Edge parts and serves as the connection between developers and users in the feedback loop.

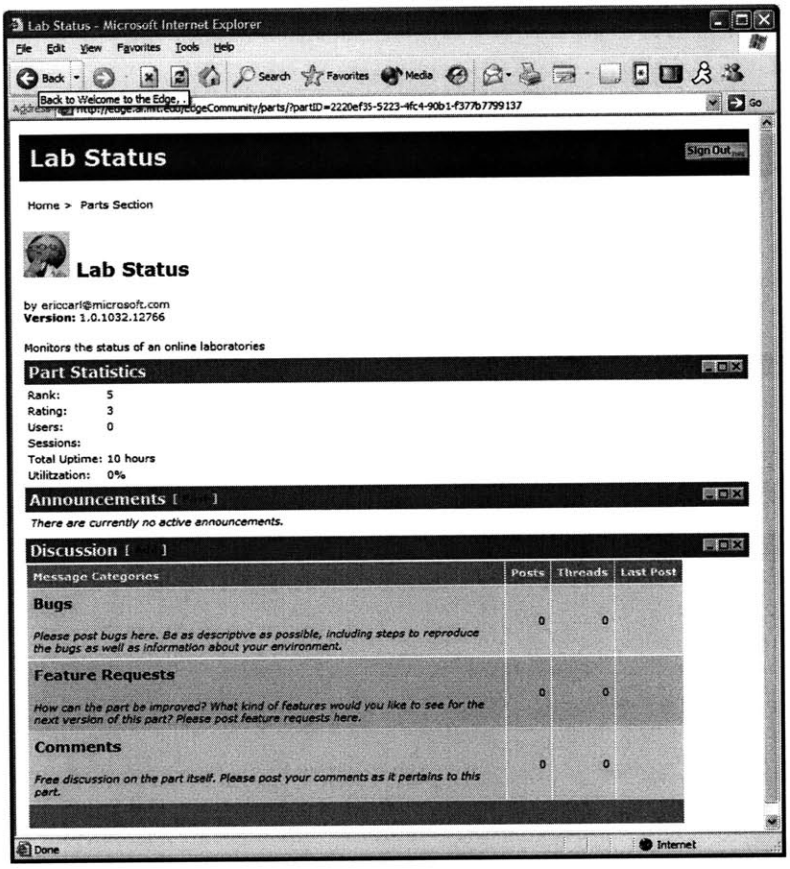

Figure 3-2: Auto-generated Lab Status part page.

The following section will discuss each of these components. While only the Edge community and various tools used for data processing in the Edge server were devel-

oped by myself, it is important to know about the client to get an understanding of the entire Edge system. Therefore the Edge client is being discussed only to serve as a backdrop in understanding the entire feedback mechanism.

3.2 Edge Ecosystem

The Edge ecosystem consists of an ongoing feedback loop which promotes improved software at each turn of the cycle. Figure 3-3 gives a graphical representation of the lifecycle which will be described in more detail below. The following section will discuss the lifecycle of a part as it progresses through the ecosystem.

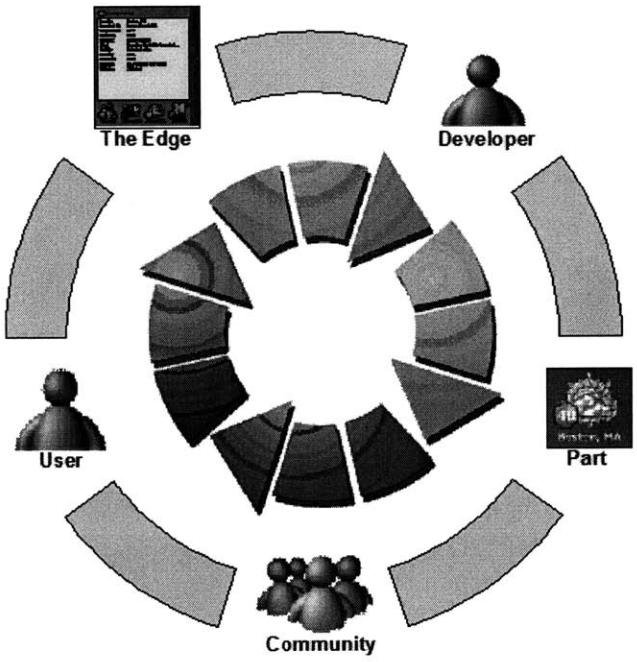

Figure 3-3: The Edge Ecosystem

3.2.1 Developer

The lifecycle starts with a developer writing a client part using the Edge SDK, such as Eric implementing the Weather part in Chapter 2. The Edge SDK is responsible for mapping part authors to parts and also handles the assignment of an unique identifier

(GUID) to each part. Once the developer is finished writing the part, it is compiled and loaded into the Edge client.

3.2.2 Community

As soon as the part is loaded for the first time, a community webpage is automatically generated and a link to the URL is provided via the Edge client where users may interact with the developer as well as view statistics about the part. The community webpage provides a mechanism for the part author to publish the part to the rest of the community, at which point the part becomes discoverable by other users.

3.2.3 Users

Users may discover parts via the part catalog in the Edge client or by searching on the main Edge community webpage. When a user finds a part they like, they may download and install the part into the Edge client. At this point the user may rate parts and provide comments, feature requests, and bug reporting.

3.2.4 Feedback

The user feedback is presented to the developer at which point the developer may fix bugs, develop additional features, or incorporate user comments into future versions. As the developer completes improved releases, the author may publish the new version, users reward the developer with better ratings and increased usage, and the lifecycle repeats itself.

3.3 Edge Client

3.3.1 Overview

The Edge client is a piece of software that runs in the windows taskbar. The client provides a nice user interface for housing Edge parts that the user may choose. The

Edge client shell is designed to minimize the amount of real estate taken on the user's screen, but at the same time present enough information for the user to quickly understand changes in data presented by the parts.

The client also provides users the ability to quickly view highest ranked and most popular parts. By providing a catalog of parts to peruse, it allows users to immediately find new or popular parts and install them with a single mouse click. This helps widen the breadth of parts the user is exposed to which keeps usage statistics more fresh and accurate while keeping the development cycle moving.

3.3.2 Edge States

Because the main purpose of the Edge is to constantly provide the user with real-time data and notifications, it must be constantly running on the client's desktop for it to be effective. With that in mind, the Edge was designed to minimize the amount of real estate it requires of the desktop. However, minimizing real estate also minimizes the amount of effective data that can be presented to the user. The Edge gets around this problem by providing three different states that the user may choose to run in.

Hidden State

The hidden state occupies the least amount of real estate among the three states. In fact, it occupies virtually no real estate, and at the same time provides virtually no information to the user. The hidden state occurs when the Edge runs in the windows taskbar in the lower right hand corner of the user's screen.

In this state the Edge icon, or the Edge connection light, is displayed in the taskbar in either a green or red color. When the connection light displays green, the Edge is connected to the Edge server and able receive data from the web services of the various parts as well as send data to the Edge server about the usage characteristics of the current client instance. Otherwise the connection light displays red. In this state the Edge is unable to provide any visual information to the user other than whether it detects a live connection or not. Figure 3-4 shows a connected Edge client

running in a hidden state.

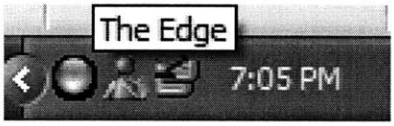

Figure 3-4: A hidden Edge client running in the taskbar

Minimized State

The minimized state allows the user to get a quick view of all parts installed. The minimized state only provides each part's indicator value to be viewed. The part indicator value is a single value that part developers deem as most important. It is typically the value that will generate notifications upon change. Again, an example of a value that is shown in the indicator value would be the current temperature for a weather part, or the NASDAQ index for a stock quote part.

In the minimized state, the Edge client runs in a rectangular box, known as the "parts view", in the lower right hand portion of the users screen. This mode gives a good balance between minimizing real estate and displaying significant data of the parts installed to the user. Figure 3-5 shows a minimized Edge client running with three parts installed. Additionally the Edge client is built to take advantage of the Microsoft .NET GDI+ opaque windows form. The user interface of the Edge client is transparent so that even in the minimized state, the little real estate the Edge client occupies is in essence "shared" so that the user can still see what is in the background behind the Edge. This design allows users to run the Edge in this state without completely overtaking that portion of the screen.

Figure 3-5: Minimized edge client with the three parts showing indicator values

Maximized State

The maximized state provides the most information to the user, but unfortunately also takes up the greatest amount of space on the users screen. The maximized state runs in a large rectangular box, known as the "details view", in the lower right hand corner of the user's screen. This mode is a combination of all three states, as the connection light, the parts view, and the details view are all visible in the state.

When a user clicks on one of the installed parts shown in the parts view, the details view shows additional information which is defined by the part. Going back to our canonical weather part example, while the indicator value shown in the parts view might be the current temperature, the details view might show additional information such as barometric pressure, relative humidity, or extended forecasts just to name a few.

The details view is also transparent. That is, even though the details view takes up the most amount of screen space, it still allows programs or widgets shown in the background to be seen so there is no need for the user to minimize the Edge client just to view background programs. Figure 3-6 shows the a maximized Edge client which displays the connection light, parts view, as well as the details view of the currently selected part.

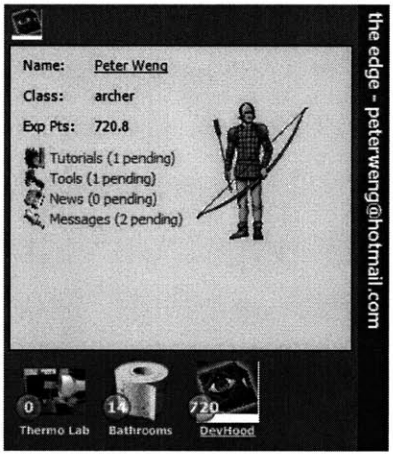

Figure 3-6: A maximized edge client showing details of the DevHood part

3.3.3 Data Collection

The Edge client was not only designed with the user interface in mind, it also has the hidden responsibility of data collection. In order to understand the justification behind some of the data collected by the Edge, it is necessary to understand the different Edge states described in section 3.3.2. In the following section I will describe not only the mechanisms built into the Edge for data collection, but also give an introduction to the type of data collected. These statistics will be further discussed in chapter 4.

The data collection mechanism built into the edge automatically tracks user behavior for each individual client. The user behavior is captured by the client, tracking which elements the user most often uses, and which client states the user is most often in. For example, one type of data used to determine Edge popularity is the utilization, which is a percentage of how much time the Edge is in use. Since the Edge can be running in any of three states, with each state being a different value as to how much the Edge is actually "in use", a formula is used to determine overall utilization with weights being applied to the time the client is in each of the three states.

Because some types of data are an aggregate value and others require multiple data applied to a formula, rather than building this logic into the client, it was designed to simply collect single data values and send those to the server where the logic exists to perform analysis. The data values are sent to the server via a session object that is responsible for collecting event information from the client.

Each Edge client is associated with a GUID that identifies that specific session. After a part is loaded by the user to the client, the client calls on a session object which handles the processing of events captured by the client. The session object writes the type of event, in this case a PartLoaded event along with event arguments, PartID and SessionID, to a session cache file.

The session cache file is used as a level of indirection in order to collect data in the event that the client is disconnected or if the user is adding parts offline via a dll file. The session cache file is kept entirely on the client's hard disk, therefore does not

require an internet connection in order to collect data. The client is set to attempt to persist the data contained in the session cache file every 30 minutes. The persistence is done through a web services architecture provided by the Edge server. It provides asynchronous web methods that allow the client to send the contents of the session cache data file to the server via remote procedure calls. When the server receives the data it can perform appropriate analysis and persistence, which is described in more detail in section 3.4.

As described above, the session data file is central on the client side in terms of the data collection mechanism. The actual data collected by the Edge client is defined to help the server perform the analysis to predict user behavior and give insight to developers on how to improve their software. The actual data that is written to the session cache data file is actually just a capture of the triggered events of the Edge client. The following events are captured by the Edge client:

Edge Events

- New installation (date, user id) - occurs when an Edge client is newly installed on a machine. This translates to a new session on the server.
- Load client (session id, date) - occurs when a previously installed Edge client is started on the machine.
- Unload client (session id, date) - occurs when a previously installed Edge client is terminated on the machine.
- Change state (session id, state type, date) - occurs when the user changes the state of the current Edge client session. In other words, when the user minimizes, maximizes, or hides the Edge client.

Part Events

- Load part (part id, date) - occurs when a new Edge part is loaded into the current Edge client session.
- Unload part (part id, date) - occurs when an Edge part is removed from the current Edge client session.

- Rate part (part id, user id, rating, date) - occurs when an Edge part is rated by the current user.

- Add comment (part id, user id, comment, date) - occurs when the current user leaves a comment for the developer of an active part from the current Edge client session.

Each of these captured events is used by the Edge server to predict user behavior and analyze popularity of parts and edge clients among other things. Section 4.1 will discuss in more detail the formulas used to draw conclusions.

3.4 Edge Server

3.4.1 Overview

The Edge server is the portion of the Edge system that handles the processing of data that is captured by the client. The server holds all the logic as to how to analyze the data to provide useful information back to the developers as to the current state of their part.

The Edge server is transparent in the sense that there is no user interface, nothing concrete or substantial that a user could see. Rather it is a black box that is there to simply gather data, process it, and spit it back out in a form that will educate developers as to how their software is currently doing in the community and its perception among its users.

In this section I will describe the design and implementation of this black box, as well as discuss the logic and formulas used to analyze the data and its importance to developers.

3.4.2 Data Processing

The design of the Edge server involves two layers. The first is the web services layer, which gives the server the ability to receive data captured by the client as well as

the ability to post data to the user. With the standardization of web services and the popularity of the TCP/IP protocol, a web services architecture was used for this layer. It provides an extremely simple and effective way for the client to easily talk with the server to asynchronously pass data to it. This layer exposes web methods to the client so that it may perform remote procedure calls to let the server know of triggered events on the client. Web methods are also exposed for the web community to extract data and make them viewable to the users. This design abstracts the data store and logic away from the client by simply providing methods for the client to call. The client does not need any knowledge of what type of database or the formulas the server uses for analysis.

The web methods simply act as an abstraction layer that provides access to the backend data store. My implementation for the backend is a Microsoft SQL Server database. The main advantage of using a relational database management system (RDBMS) such as MS SQL Server is that the data is stored in a structured and organized fashion. It has a nice feature which enforces cross references between rows in separate tables. This is known as the "consistency" test for an RDBMS [4]. For a system like the Edge where most of the statistics calculated are within the scope of a single specific part, this consistency feature becomes a huge advantage.

The second layer is the analysis layer. This layer contains all the logic behind the processing of the data received from the client. The goal of this layer is to try to make assumptions of the overall popularity of a part based upon information provided by the client. However, it's almost impossible to get this type of analysis perfect the first time. In order for any formulas to work, it will inevitably have to be tweaked to more accurately measure the overall success of a part and put a numerical value to how well the user community has embraced it. It is for this reason the analysis layer exists. Rather than building the logic into the Edge clients, the analysis layer places all the logic, formulas, and data in a single point for processing where version updates of the Edge client softare would not be required for tweaking the analysis process. Section 4.2 will provide more of a discussion into the justifications behind the analysis process.

3.5 Edge Community

3.5.1 Overview

The Edge community is the component which allows analyzed data to be communicated not only to developers but also the users. Its main function is to serve as a point of interaction between developers and users and provide the link between them. The feedback mechanism in the Edge helps to analyze how well a developer's software is being embraced by the community, but without the Edge community, the developers would never have access to this information, nor would the users have access to information about the parts.

3.5.2 Data Posting

The Edge community serves as a portal type application for the Edge system. Its two main areas are the Edge main page and the auto-generated parts pages. Each area provides useful statistics information to both developers and users; however the parts pages are more important to the feedback mechanism as described in the following section.

Main Page

The Edge page has been designed to contain modularized components for each of the distinct features of the main page. The advantage of using these modularized component boxes is that each can be easily hidden or made visible to the user when necessary. Each of the important components will be discussed in the following section. Figure 3-7 shows a screen shot each of these components discussed.

The announcements component, as the name states, allows important Edge announcements to be conveyed to all Edge users. These might be messages about new versions or features coming soon, or service outages. Also important is the discussion component which is a typical message forum system where edge users may leave notes for developers or discuss with other users about certain features they like, or they feel

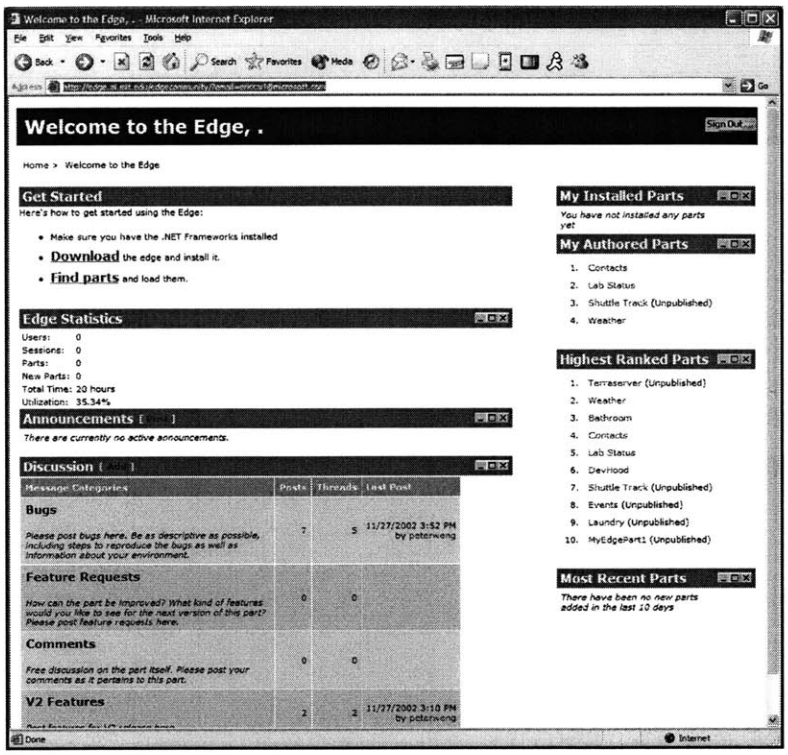

Figure 3-7: The Edge Main Page

are missing.

Certain components on the Edge main page also serve a role in the Edge ecosystem by promoting the adoption of parts. The ranked parts component lists the top ten parts by order of how well they are ranked by the community. This component allows users to quickly view popular parts and install them in their Edge client thereby effectively promoting "good" software and increasing activity of popular parts. The recent parts component is another one that promotes user activity. This component displays the top ten newest parts born into the Edge ecosystem. These parts are sorted by date so that new parts will be given a jumpstart at receiving user activity, thus allowing developers to begin collecting data from its user base.

Other components are customized to the currently logged in user. The installed parts component allows the current user to have quick links to the parts that the user has installed in their Edge client. This information is available because the feedback mechanism sends this information to the Edge server for each client. Thus the Edge

server is able to provide this customized information to the community web page. The authored parts component works in a similar fashion, though providing different information. The author parts component displays the list of edge parts that the current user has developed, providing quick links to part pages that may be used by the developer to perform administrative functionality on the part.

The last and probably most intriguing component is the statistics component. This component displays the statistics of all Edge clients ever installed. The statistics that are displayed are the number of active users, the number of sessions, the number of new parts and active parts, the total amount of time the Edge has been running, and the percent utilization of the Edge. These statistics are obtained from the Edge server as described in section 3.4.

Part Pages

What's unique about the Edge community is that for each new part that is added, a dynamic webpage is automatically generated for the part. This part page provides each part with an area where its users can interact and obtain further information about the part. Just as the Edge main page was designed, the part pages also contain different components that display this information to the users. Figure 3-8 shows the components of the part pages.

Like the Edge main page the part pages also contain an announcements and discussion forums component. Both the announcements and discussion forums are limited to the domain of that particular part. That is, each part has its own discussion forums separate from all other parts.

The part pages also contain a statistics component similar to that of the Edge main page. Specifically, the statistics components displays the rank, rating, number of users of the part, number of active sessions of the part, the total uptime, and the overall utilization of the part. Each of these statistics is described in more detail in section 4.1.

Additionally the part pages provide administrative functionality for developers. When a developer is visiting a part page of their part, they are provided with function-

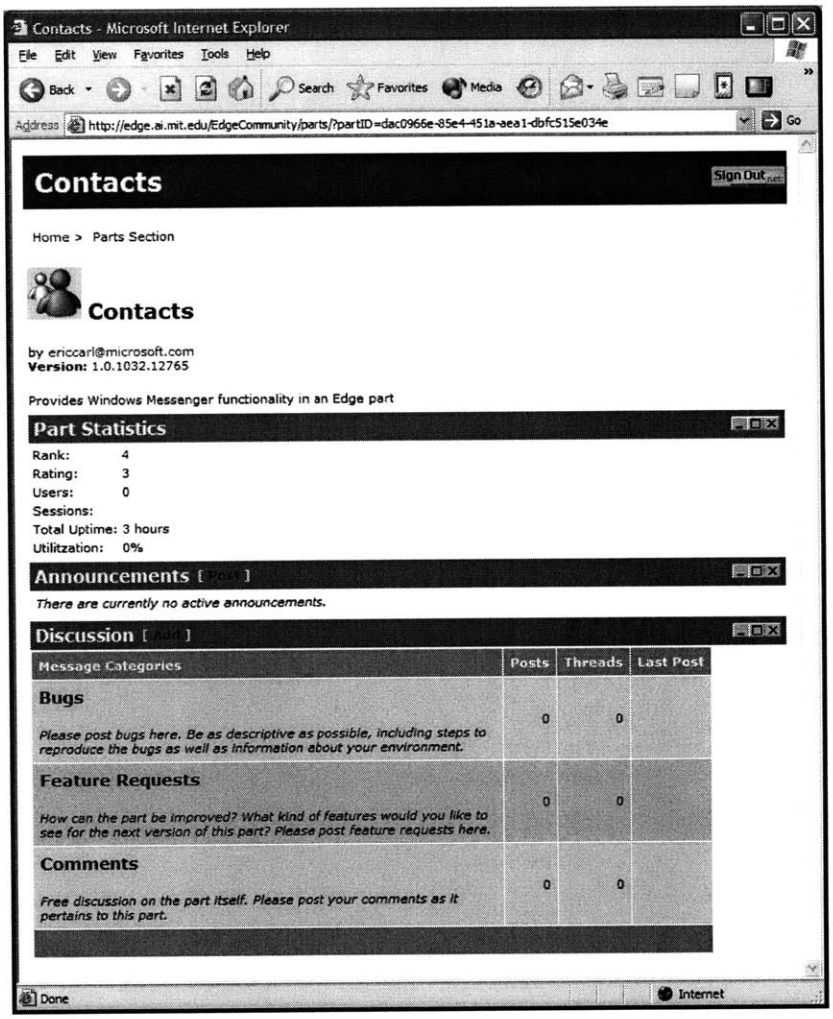

Figure 3-8: The Contacts part page with the customization options for the part author

ality to post announcements or create additional message forum categories specific to their part. Also, if a part is unpublished, or not yet made available to the public, the part page allows the developer the ability to publish the part. The developer does this by specifying a URL location in the part page where the dll file can be downloaded to load the part into the Edge client.

Chapter 4

Data Analysis

4.1 Feedback Statistics

As described above, the Edge system consists of three components, the client, data server, and web community. While the client is responsible for the data collection, the latter two components handle the backend of the feedback mechanism. This is the area of my main contribution.

The Edge will be able to track data for individual parts. There are two main categories of data which the Edge will track, that is direct user feedback and inferred feedback. Together these statistics provide the tightly coupled feedback mechanism the Edge uses to help developers improve their software.

The Edge allows users to rate parts as well as leave comments for the developers. While the user comments cannot be used for statistical analysis, the ratings of parts provide a very accurate means of evaluating the success as well as the worth of each part. These ratings directly correlate to whether or not users in the community like or dislike the part.

In addition to direct user feedback, the Edge also tracks usage data. While this usage data cannot be directly translated into user feedback, it can be used by the developer to determine usage patterns for its part and more importantly infer how successful the part is doing in the community.

The data analysis process begins with calculating specific statistics that will be

used to form the basis of the feedback mechanism. These statistics are categorized and defined as follows:

Edge Statistics

- Active Sessions - The number of distinct sessions which have been created in the Edge over the last 24 hours. This statistic measures the current activity of the Edge.

- Active Users - The number of distinct users who have used the Edge in the past 30 days. This metric defines the growth of the Edge in the last 30 days. Note the difference from active sessions. This statistic measures the growth of the user base whereas active sessions measures how many of those users are actively using it.

- Active Parts - The number of distinct parts that are contained in any session over the past 30 days.

- Newly Published Parts - The number of new parts that have been published to the Edge in the last 30 days. Note that this does not include revisions of old parts.

- Total Time - The total amount of time that the Edge has been run. That is, an aggregate over all sessions for all users.

- Utilization - I consider the Edge to be running in three states, hidden, minimized, and maximized. See section 3.3.2 for more information about Edge states. The utilization value of the Edge is a percentage of how much of time the Edge is in use, with weights applied to each of the 3 states. Thus the total utilization formula is (maximized + 0.5*minimized) / (maximized + minimized + hidden).

Part Statistics

- Online Users - The number of distinct users who have run the Edge in the last 24 hours which included this part.

- Active Users - The number of distinct users over the past 30 days who have run this part.

- Total Uptime - The total amount of time that this part has been running in any session.

- Utilization - The percentage of time that this part has been active in relation to other parts within this session. This statistic is taken over the last 30 days in order to avoid carrying dead weight with inactive parts.

- Average Rating - The average rating of this part taken over all active users. A part rating is provided by users of the part.

- Total Comments - The number of total comments left for the developer of the part.

- Ranking - The ranking is a single numerical value that is meant to get an overall picture of the popularity of the part. It is a combination of utilization, rating, and number of users actively using this part. The formula for calculating ranking is simply utilization * average rating.

The part statistics are at the core of the data analysis of the feedback mechanism. By examining the changes in ranking over time, matching the date with the revision history of the part, one can find whether there is a correlation between the popularity of the part among the user community and the improvement of a part by the developer. This correlation of course can be attributed to the feedback mechanism promoting the improvement of parts by providing developers with direct user feedback about their part.

With the above statistics, it is easy to find usage patterns of certain parts. For instance I can answer questions such as "Do users typically have my part active? Do they have my part hidden?", "How many users are using my part?", or "How do users rate my part?" These types of questions are important in first evaluating how good a part is as well as a way to track improvement of a part over time.

Since the ranking statistic captures the overall popularity of the part by combining usage statistics with direct user ratings, it is a good metric for determining the improvement of a part. By relating this with the revision history of the part, I can conclude whether part rankings help to initiate developer revisions which cause higher rankings thus concluding an improvement on the software. In other words, if a part's lifetime were broken down into time periods matching the revisions of the part, by examining the change in rankings over part revisions I can conclude whether the software has exhibited an improvement over revisions.

Additionally to show that user feedback has in fact promoted developer revisions of a part we can look at how the volume of comments left by users as well as the number of ratings, whether good or bad, correlate with the timing of revisions. For instance if developer revisions seem to follow when the total number of comments grows large or the number of ratings increases, this provides evidence that the feedback loop is in fact stimulating developers to provide improved revisions of their software.

4.2 Feedback Analysis and Implementation

The part statistics are kept in a SQL table "PartRating". The part objects are contained in a table "Part". The columns of the part table contain the properties of the Part object. Some of the more important columns to this discussion include the following:

- PartID - this is a GUID which simply assigns a unique identifier to this part. Also the primary key for the table.
- Name - the string name of this part.
- Ranking - the integer value represents the ranking of this part. The ranking value was described in the previous section.
- Utilization - this floating point value represents the percentage of the amount of utilization by users of this part. This statistic is also described above.

The PartRating table stores the values of how each user rates the specified part. As discussed above, the consistency property of the RDBMS helps to check consistency in that each rating that is added to the PartRating table assures that it is for a valid, existing part. The reason for this is that the PartRating table has a references link from the partID column to the "Part" table. The PartRating table contains the following columns:

- partRatingNo - this integer is the primary key of the table and assigns a unique value to each particular rating.
- partID - this is the column that contains the reference link to the Part table. This determines which part is the recipient of this rating value.
- email - this columns contains the reference link to the Owner table. This link defines which Edge user was the rater for this rating value.
- rating - this integer value between 1 and 5 which corresponds to how well received the part was by the user. A value of 1 corresponds to a "Hate It" rating and a value of 5 corresponds to a "Love It" rating.

The ranking and utilization values are calculated using batch SQL query statements that update the Part table every two hours. The batch update queries for utilization work by first creating a temporary table, #utilization, to hold the utilization value during processing. Next the total amount of active time that all parts are calculated by summing the difference in the start date and end date of each session. This value is then stored in a temporary value @total. Finally the total time of activity in the last 30 days for each part is calculated individually and divided by the @total time of activity of all parts. This now becomes the utilization value for each part and is stored in our temporary table #utilization upon calculation. Finally each row of the #utilization table can be used to update the utilization column of the Part table to make the utilization values permanent. The following SQL query statement is used to calculate the utilization value. See appendix A for a listing of all SQL stored procedures used.

```
DECLARE @total float
select @total = cast(sum( datediff(second, starttime,endtime))
        as float )
from sessionpart

INSERT #utilization ( partID, utilization)
SELECT partid,
        cast(sum(datediff(second, starttime, endtime)) AS float)
        / @total AS utilization
FROM sessionpart                                                    10
WHERE sessionPart.startTime > dateadd( day , -30 ,
      CURRENT_TIMESTAMP)
GROUP BY partid
```

The batch update queries for ranking work in a similar fashion. Remember that ranking is meant to capture the overall popularity of a part. Utilization captures the inferred feedback of the part by examining how often it has been in use. Additionally there is the direct user feedback that is the rating values assigned by users saying how much they liked or disliked the part. The ranking value simply combines these two types of feedback into an overall popularity ranking. The SQL statement is as follows:

```
SELECT part.partiD
FROM part
LEFT OUTER JOIN partrating ON( part.partid = partrating.partid)
GROUP BY part.partID
ORDER BY MAX(utilization) *
          Coalesce(cast(Sum ( rating )AS float) /
          cast( count( rating) AS float), 0.0)
DESC
```

This SQL statement works by simply selecting the ranking value by multiplying the max utilization value of each part with the average rating of that part. Again

these values are stored in a temporary table #ranking which are then used to update the ranking column of the Part table making the calculated value permanent.

4.3 Developer-User Feedback Loop

Probably the most intriguing aspect of the Edge is the research involved in the Developer-User feedback loop. In the previous sections I have discussed the implementation and design of the feedback loop. In this section I will discuss the theory behind how the developer-user feedback loop can be an instrumental component in helping to improve software and reduce development time.

The benefits of the Edge feedback loop is of course limited to this application since the type of feedback collected is hard wired into the system, however this can be considered a proof of concept for how this type of mechanism's benefits can be realized in all types of software. What's interesting about the Edge is how well it resembles that of real world software application. The Edge consists of various parts, built by developers, which together make up the bulk of the Edge client. This is similar to a typical software project which is built by teams of developers each working on separate components which together make up the entire software product.

The feedback portion of the loop consists of course of feedback and usage statistics provided by the users of the various parts. While most software products might not have a community of users, they all have some market of users. These usage statistics can just as well be applied to any market while providing the same type of information.

The example provided in chapter 2 has already shown how the data provided by the feedback mechanism can improve "parts" in the Edge. Now let's take it a step further and discuss what kind of benefits this type of feedback mechanism can provide for other applications. Imagine a word processor application Macrosoft Word 1.0 with a built in developer-user feedback mechanism. Immediately the developers can be notified of bug reports sent by users, prioritizing the most common occurrence of bugs. Additionally when planning the next version of Macrosoft Word, developers and product managers can look at the data provided by the feedback mechanism,

looking at the inferred feedback by the users. Since the mechanism performs some user tracking they can look at data that will answer questions like "Do users prefer to use the pull down menu to save a document, or click on the toolbar icon?" In fact as Macrosoft Word enters into future releases you can even take the analysis a step further by matching the feedback data along with how the software has changed to determine how effective those changes were. For an example let's imagine Macrosoft Word having a button in the bottom right hand corner of the screen that when pressed prints the document. Now in version 2.0 the print button has been moved to the top left hand corner of the screen. The feedback mechanism has the ability to tell you how times that print button was pressed in version 1.0 and by how many different users. By comparing that data with the amount of times it was pressed in version 2.0 (now in its new location) could tell developers if that move made it more user-accessible or not.

While this is obviously a fictitious example, the theories behind the potential of such a user feedback mechanism are real. Chapter 2 already gave an example of the benefits of a feedback mechanism in a real software application. In fact even with the data collected by the Edge feedback mechanism, there are already benefits that can be realized. If Macrosoft Word collected the same type of data as the Edge, a program manager can already look at the total amount of time users use Macrosoft Word per day, or compare the number of installs to uninstalls over the period of a year. Each of these statistics are already giving insight into how the software might be improved. By providing an electronic data collection mechanism one can begin to see how software can be improved by directly incorporating information on how its user base uses the software into future revisions.

Chapter 5

Related Work

User feedback has long been considered the most effective way of improving any type of consumer products and software is no different. While the automated feedback mechanism in the Edge is a novel idea with no current system quite like it, there have been many attempts at finding the best way to build products to cater to consumers. This section will discuss some of these processes as they relate to software development.

5.1 User Driven Design

In this methodology, software designers often solicit user feedback via surveys or interviews in the very early stages of the development cycle in order to understand the needs of the user. The purpose of this is threefold:

1. To document software requirements that satisfy user needs
2. Analyze user tasks which can be improved or made more efficient through software
3. Prioritize feature sets to design extensible software

While this is an integral part of software development, any data collected is essentially just a prediction of how users will use the software. In addition it is generally

a time-consuming process where at best you may only survey a small subset of your potential users. For instance a car manufacturer may have to send survey collectors to a parking lot to collect data, where the bulk of the work will be in convincing drivers to spend time to fill out such surveys. There may also be a question of how valid the data collected is, since it is more an opinion of the user's perception of the software rather than an inference based on the user's interaction with the software.

5.2 Open Source Model

The open source development model has been a recent trend in software development that has shown success in many areas. One of the goals of open source projects is to give users of open source software (OSS) the ability to remove inadequacies and improve upon the project from the user's point of view. OSS works by providing users with the source code for software projects to allow users to customize the software to suit their needs.

This model of development does away with the need for user feedback by simply allowing users the ability to directly modify the software. It is most flexible in terms of allowing each user to have a customized version of the software that specifically fits their needs. However, this model is limiting in that users are requiring to have a fairly deep understanding of the source code, which in many cases is non trivial.

Chapter 6

Conclusions and Future Work

The Edge ecosystem and development cycle is a novel idea that at this time has no known counterparts. Not only will the user feedback and inferred feedback collected by the Edge provide developers useful information in gauging how well their software is embraced by the public, the additional ranking system provides an ego-driven environment that compels developers to respond to their users needs. Thus the results of such a system may have a potential positive impact on how developers can improve their software.

In this thesis I have described the current implementation of the Edge and the built-in feedback mechanism. I have shown how such a system could be of potential use to the improvement of existing software products in the market. However, up to this point this is still only a proof of concept. There is much work left to be done to make this a viable system. For example, the heuristics and metrics used to measure software popularity and to track user behavior need to be tweaked. The Edge was designed with this in mind, however more work needs to be done to actually look at massive amounts of real data and tune the analysis process to most accurately represent the users. This long term future work could significantly improve the results of such a feedback mechanism.

Additionally, at this point the feedback mechanism built is specific to the Edge in terms of the data collection and the analysis process. However such a system has benefits for all different types of applications. One of the downsides is the need to rebuild

parts of this feedback mechanism into each different type of software application. One possible extension to this project would be to build a standardized, extensible data store and web methods that would be accessible to the public. This would allow any developers who would like to have a similar feedback system in their software to only have to build the client side. That is once the data collection agents are built specific to that application, it would talk to the standardized data store and web services provided by the Edge. This would eliminate the need to have to host your own server to store this data. This would help in not only providing a central store for housing massive amounts of data but also help to encourage the adoption of similar feedback mechanisms by easing the development process for building these types of systems.

In this thesis I have described a prototype of a developer-user feedback system as it exists in the Edge. There are however many privacy issues with such a system that tracks user behavior and collects usage statistics. However, due to the research nature of this project these privacy issues were of no concern at this point. Regardless, the Edge has been shown to be a capable feedback mechanism that can be used for improving software. Future work notwithstanding, I believe this project has been a success, providing a useful starting point for anyone interested in incorporating similar developer-user feedback mechanisms into a wider variety of applications.

Appendix A

SQL Stored Procedures

```
CREATE Procedure UpdateBatchStatistics
AS
-- now the utilization
--drop table #utilization
create table #utilization
(
        partID uniqueidentifier not null,
        utilization float not null
)

declare @total float
select @total =
        cast(sum( datediff(second, starttime,endtime)) as float)
from sessionpart

insert #utilization ( partID, utilization)
select partid,
        cast(sum(datediff(second, starttime, endtime)) as float)
        / @total
as utilization
```

```
from sessionpart
where sessionPart.startTime > dateadd( day , -30 ,
      CURRENT_TIMESTAMP)
group by partid

update part set part.utilization = #utilization.utilization
from #utilization
inner join part on ( part.partID = #utilization.partID)

create table #ranking                                          30
(
        rno int identity(1,1) not null,
        partID uniqueidentifier not null
)
insert #ranking (partID)
select part.partiD
Coalesce(cast(Sum ( rating )as float) /
cast( count( rating) as float)
0.0)
                                                               40
FROM part
left outer join partrating on( part.partid = partrating.partid)
group by part.partID
order by max(utilization) * Coalesce(cast(Sum(rating) as float)
        / cast( count( rating) as float), 0.0) desc



update part set
       part.ranking = #ranking.rno,
       rankingUpdateTime = CURRENt_timeSTAMP                   50
```

```
from #ranking
inner join part on ( part.partID = #ranking.partID)

drop table #ranking
drop table #utilization
go


--statistics support
CREATE PROCEDURE GetEdgeStatistics                                    60
        @activePeriod INT,
        @minimizedWeighting INT,
        @activeSessions INT OUTPUT,
        @activeUsers INT OUTPUT,
        @activeParts INT OUTPUT,
        @newlyPublishedParts INT OUTPUT,
        @totalTime bigint OUTPUT,
        @utilization float OUTPUT
AS
                                                                      70
SELECT @activeSessions = Count( distinct email )
FROM EdgeOwner INNER JOIN Session
ON EdgeOwner.EdgeID = Session.EdgeID
WHERE startTime > dateadd(hour, -24, CURRENT_TIMESTAMP)

SELECT @activeUsers = Count( distinct email )
FROM EdgeOwner INNER JOIN Session
ON EdgeOwner.EdgeID = Session.EdgeID
WHERE startTime > dateadd(day, -1 * @activePeriod,
    CURRENT_TIMESTAMP)                                                80
```

```
SELECT @activeParts = Count( distinct ( cast (
        partID as varchar(40))))
FROM SessionPart
WHERE startTime > dateadd(day, -1 * @activePeriod,
      CURRENT_TIMESTAMP)

SELECT @newlyPublishedParts = Count(*)
FROM Part
WHERE DatePublished >                                          90
      dateadd(day, -1 * @activePeriod, CURRENT_TIMESTAMP)

SELECT @totalTime =
        coalesce(SUM(datediff(second, startTime, endTime)),0)
FROM Session



--drop table #times
select a.actionNo, a.eventTime as startTime,
        b.eventTime as endTime,                                100
        datediff(second, a.eventTime, b.eventTime) as timeDiff
into #times
from edgeevent as a
inner join
edgeevent as b  on( a.sessionID = b.SessionID)
where b.eventTime =(select min( eventTime  )
from edgeevent as c
where c.sessionID = a.sessionID
and c.eventTime > a.eventTime)

                                                               110
```

```
insert #times ( actionNo, startTime, endTime, timeDiff)
select  a.actionNo ,
        max(a.eventTime) as startTime,
        max (b.endTime) as endTime ,
        datediff( second, max(a.eventTime), max(b.endTime))
from edgeevent as a inner join session as b
on ( a.sessionID = b.sessionID )
and a.eventTime =
(select max(eventTime) from edgeEvent as e
where e.sessionID = a.sessionID)                                    120
group by  a.sessionID, a.actionNo
order by 2 desc

declare @maximized float
declare @minimized float
declare @hidden float

select @maximized = cast(sum( timeDiff)as float) from #times
where actionNo= 2
select @minimized = cast(sum( timeDiff) as float)from #times        130
where actionNo= 1
select @hidden = cast(sum( timeDiff) as float) from #times
where actionNo= 0

select @utilization = coalesce((@maximized + @minimized /2)
       / (@hidden + @maximized + @minimized),0.0)


GO
                                                                    140
```

```
create PROCEDURE GetPartStatistics
                @partID uniqueIdentifier,
                @activePeriod INT,
                @onlineUsers INT output,
                @activeUsers INT output,
                @totalUptime BIGINT output,
                @utilization float output,
                @averageRating float output,
                @ranking int output                                    150

AS

SELECT @onlineUsers = Count( distinct email)
FROM EdgeOwner
INNER JOIN Session  ON EdgeOwner.EdgeID = Session.EdgeID
INNER JOIN SessionPart
ON Session.SessionID = SessionPart.SessionID
WHERE SessionPart.PartID = @partID
AND SessionPart.startTime > dateadd(hour, -24,                         160
    CURRENT_TIMESTAMP)

--number of active edge clients
SELECT @activeUsers = Count( distinct email)
FROM EdgeOwner
INNER JOIN Session  ON EdgeOwner.EdgeID = Session.EdgeID
INNER JOIN SessionPart
ON Session.SessionID = SessionPart.SessionID
WHERE SessionPart.PartID = @partID
AND SessionPart.startTime >                                            170
```

```
dateadd(day, -1 * @activePeriod, CURRENT_TIMESTAMP)


--total uptime
SELECT @totalUptime =
        coalesce(SUM ( datediff(second, startTime, endTime)),0)
FROM SessionPart
WHERE PartID = @partID

SELECT @utilization =                                                    180
        coalesce(utilization, 0.0) from part
where partID = @partID

SELECT @averageRating =
        coalesce( sum(rating) / count(rating), 0.0)
FROM PartRating
WHERE partID = @partID

SELECT @ranking = coalesce(ranking,0) from part
where partID = @partID                                                   190

GO
```

Bibliography

[1] Liliana Ardissono and Robin Cohen. Extending the role of user feedback in plan recognition and response generation for advice-giving systems: An initial report. In *Canadian Conference on AI*, pages 109–120, 1996.

[2] Brian D. Davison. Web traffic logs: An imperfect resource for evaluation. In *Proceedings of the INET'99 Conference*, 1999.

[3] Brian D. Davison. HTTP simulator validation using real measurements: A case study, 2001.

[4] Philip Greenspun. *Philip and Alex's Guide to Web Publishing.* Morgan-Kaufman Publishers, 1999.

[5] H Wagner. Tracking the navigation behavior of web communities. Master's thesis, MIT, 2002.

[6] David M. Hilbert and David F. Redmiles. Collecting user feedback and usage data on a large scale to inform software development.

[7] David M. Hilbert and David F. Redmiles. Agents for collecting application usage data over the Internet. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 149–156, New York, 9–13, 1998. ACM Press.

[8] David M. Hilbert, Jason E. Robbins, and David F. Redmiles. EDEM: Intelligent agents for collecting usage data and increasing user involvement in development. In *Intelligent User Interfaces*, pages 73–76, 1998.

[9] Hillol Kargupta, Ilker Hamzaoglu, and Brian Stafford. Scalable, distributed data mining - an agent architecture. In *Knowledge Discovery and Data Mining*, pages 211–214, 1997.

[10] Hillol Kargupta, Brian Stafford, and Ilker Hamzaoglu. Web based parallel/distributed medical data mining using software agents, 1997.

[11] H Lieberman and D Maulsby. Instructible agents: Software that just keeps getting better. *IBM Systems Journal*, 35(3), 1996.

[12] D Martin and S McIlraith. Bringing semantics to web services. *IEEE Intelligent Systems*, pages 90–93, January 2003.

[13] James E. Pitkow and Colleen M. Kehoe. Emerging trends in the WWW user population. *Communications of the ACM*, 39(6):106–108, 1996.

[14] James E. Pitkow and Margaret M. Recker. Using the Web as a survey tool: results from the second WWW user survey. *Computer Networks and ISDN Systems*, 27(6):809–822, 1995.

[15] R. Toward and W. Information. IEEE Computer.