A Use Case Model for RAS in an MPP Environment

Suzanne M. Kelly
Sandia National Laboratories*
Scalable Systems Integration Department
PO BOX 5800
Albuquerque, NM 87185-0817
smkelly@sandia.gov

ABSTRACT

A use case model is an effective way of specifying how Reliability, Availability, and Serviceability (RAS) features would be employed in an operational Massively Parallel Processors (MPP) system. As part of a research project on RAS for MPPs, one such model was developed. A brief introduction to the use case technique is followed by a discussion of the developed model.

Keywords

RAS, MPP, use case, storyboard.

1.0 The Unified Modeling Language

The fundamental concepts in use cases existed in previous techniques such as story boarding and scenario development. The use case model itself has been evolving since Ivar Jacobson introduced it in 1992 [1]. In the mid 1990's, the use case model was incorporated into the Unified Modeling Language (UML) standard [2], which has been adopted as a standard by the Object Management Group (OMG) [3].

The UML is an object modeling language. It unifies the models of Booch [4], Rumbaugh [5], and Jacobson [1]. UML is not a method. There is no notion of process. It consists of (currently) 12 diagrams, of which the use case diagram is one. One can incorporate some or all of the UML notations and diagrams into their chosen software development process. Some of the diagrams are targeted to Object-Oriented analysis and design (OOAD), such as the Class Diagram and the Package Diagram. Others such as the use case diagram and the state transition diagram are applicable to non-OOAD development methodologies.

The purpose of standard diagrams is improved communication. If the notations are well understood by a broad community, the graphical view can provide volumes of information to the reader in just a page or two.

1.1 Use Case Concepts

The key concepts in a use case model are
- Use case
- Actor

A use case is a specific way of using the system by performing some part of the functionality. Each use case constitutes a complete course of events initiated by an actor [e.g. user] and it specifies the interaction that takes place between the actor and the system [1].

An actor is a representation of what interacts with the system. It may be a person, another system, or something else [e.g. cron daemon]. Actors represent someone or something that needs to exchange information with the system; but they are not part of the system [1].

Ovals represent use cases and stick figures represent actors. An arrow between indicates the direction of initiation, which is not necessarily the direction of data flow. These simple notations were selected so that sophisticated tools are not required to use the model. Both use cases and actors are assigned names. The author prefers to name each use case with a verb followed by an object. The initiating actor implies the subject.

1.2 Use Case Example

Figure 1 shows an example of a single use case for an ATM system:

[*] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

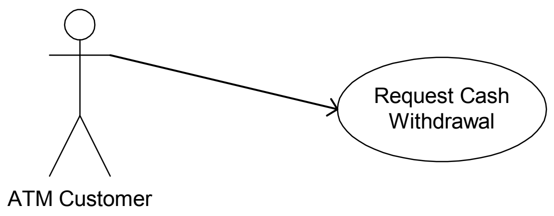


Figure 1: Single Use Case Example

In this trivial example, the actor is named the "ATM customer" and the use case is named "request cash withdrawal." This use case would be one of many use cases for an ATM system. Fleshing it out a bit further, we would have something like:

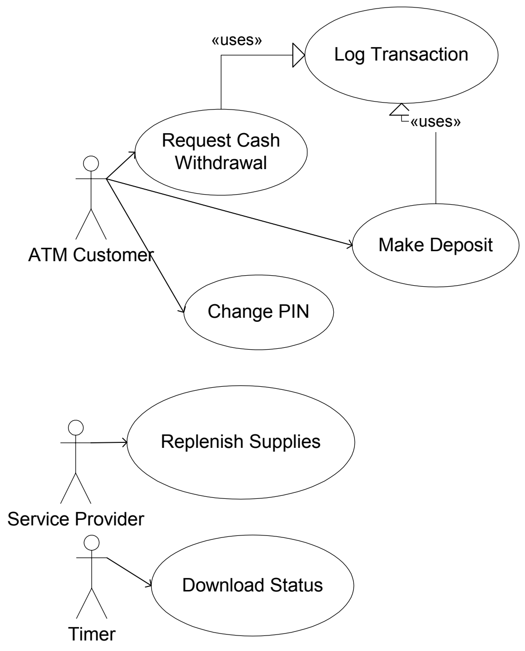


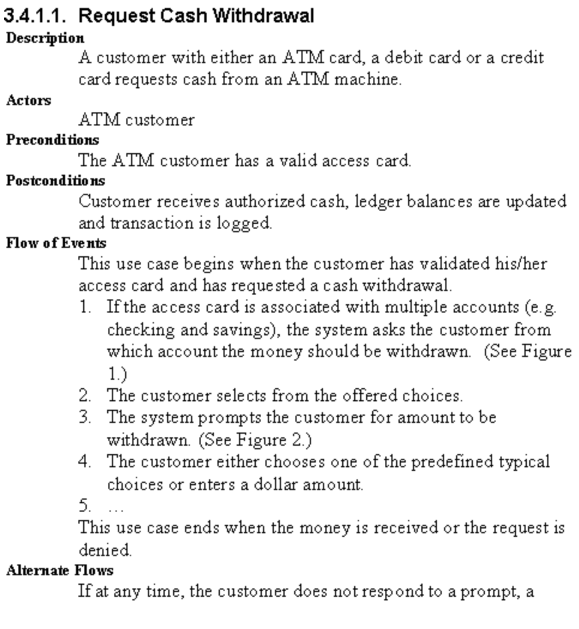
Figure 2: Simple Use Case Diagram

The entire collection of use cases is called the use case diagram. Each use case is accompanied by use case documentation. At a minimum, there is a description and one flow of events, or scenario. The author's preferred use case documentation template is based on one in [6] and consists of:

- Description
- Actors
- Pre & Post conditions
- Detailed Flow of Events
- Alternate Flows
- User Interface
- Data Requirements

Provide one or two sentences describing the use case. Identify the actors that are involved. Mention any conditions that must pre-exist. If the end of the use case will have reached a key state, identify it as a post-condition. The flow of events, also called a scenario, gives a step-by-step description of the interaction between the system and the actor(s). The typical scenario is first described in detail. Alternative flows may provide documentation on error conditions or less likely scenarios. The user interface section is often a graphic showing what the actor(s) will see if a GUI (graphical user interface) is involved. Lastly, the data items used in the use case are enumerated. When generating use case documentation, one assumes the necessary data is available—almost as if it is floating in space, ready to be retrieved from or added to. How data is actually maintained is not the focus of use cases.

An example may again prove illuminating. The following text contains the beginning portions of what would be included in the documentation for a use case.


3.4.1.1. Request Cash Withdrawal

Description
A customer with either an ATM card, a debit card or a credit card requests cash from an ATM machine.

Actors
ATM customer

Preconditions
The ATM customer has a valid access card.

Postconditions
Customer receives authorized cash, ledger balances are updated and transaction is logged.

Flow of Events
This use case begins when the customer has validated his/her access card and has requested a cash withdrawal.

1. If the access card is associated with multiple accounts (e.g. checking and savings), the system asks the customer from which account the money should be withdrawn. (See Figure 1.)
2. The customer selects from the offered choices.
3. The system prompts the customer for amount to be withdrawn. (See Figure 2.)
4. The customer either chooses one of the predefined typical choices or enters a dollar amount.
5. …

This use case ends when the money is received or the request is denied.

Alternate Flows
If at any time, the customer does not respond to a prompt, a

Figure 3: Example Use Case Documentation

1.3 The Value of Use Cases

Hopefully the previous description has given the reader an appreciation for the purpose of use cases. They can be an excellent communication vehicle between software developers and software users. They make minimal use of computer science terms. They define system behavior in a way that can be understood and appreciated, rather than the traditional laundry list of requirements. Once the system is developed, the test cases can be based on the use cases themselves. Also the user documentation, or at least the table of contents, falls out naturally from the use cases. Much of the graphical user interface is drafted. And many of the data items for a data repository or database are identified.

There are limits to the value of use cases. They only define the customer-visible portion of the system. Much of the system is still a black hole:

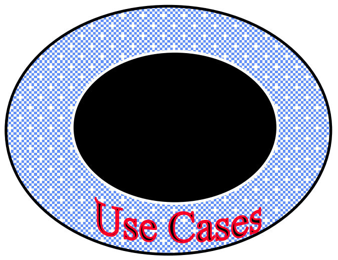


Figure 4: Use Cases present external interfaces only

As such, they provide minimal information for system architectural design. The UML offers 11 additional diagrams to help with those aspects of system development.

2.0 RAS Study

The previous tutorial set the stage for describing an effort undertaken at Sandia National Laboratories. Sandia has a strong commitment to high performance computing (HPC) and in particular, to massively parallel processor systems. These systems are made up of tens of thousands of hardware components. Just due to the volume, failures happen on a regular basis. The software components running on these systems tend to be equally sophisticated and are rarely trouble-free. These facts make RAS difficult to achieve. The Sandia study looked at RAS features and how to employ them in an MPP to achieve good reliability, availability, and serviceability. The results of that study are documented in [6].

2.1 Definition of RAS

There is a community of professionals that apply specific meaning to the terms comprising RAS. The terms and their definition follow:

Reliability: the likelihood a system or component will sustain full functional operation over its lifetime. This is sometimes referred to as fault avoidance.

Availability: the likelihood a system is operational at any given time. This is sometimes referred to as fault tolerance.

Serviceability: the measure of a system's ability to sustain repairs to faulty components. This is referred to as fault identification and repair.

How one measures these individual attributes can vary from computer system to computer system. Reliability is often the most perplexing. One hears of MTBI (mean time between interrupts) and MTBF (mean time between failures), but the definitions and application of interrupts and failures is not standard. Availability is measured in percent of time the system is operational. But that too can vary depending on one's determination of when a system is "up." Serviceability is measured in MTTR (mean time to repair). It has a strong influence on the availability statistics, but again, the determination of the time between "repaired" and "up" may vary considerably, due to issues such as long boot sequences.

2.2 Use Case Model for RAS in an MPP

One of the products of the Sandia RAS study was the use case model. It was used as a communication and analysis tool for gleaning the unique features that a RAS system needed to provide for a MPP. The model looked at providing RAS features for both hardware and software components.

We began by identifying all the actors in the RAS system. These are all the persons and "things" that interact with the RAS system.

Asynchronous Event – an event that happens at any time
Manager – a person responsible for ensuring the system meets its RAS goals.
Operator – a person trained to monitor specific system-generated observable events and to follow a set of procedures based on the observable events.
Synchronous Event – an event that happens at a predetermined time.
System Hardware Administrator (SHA) – a person trained to monitor system hardware logs and resolve hardware problems.
System Software Administrator (SSA) – a person trained to install and configure software

components, monitor system logs, and resolve problems. This person will usually be the one to differentiate hardware and software problems.
System Software Programmer (SSP) – a person engaged in on-going software engineering that results in updates to the operating system(s) and other low-level service programs.
User – a person running and/or developing applications on the system.

Note that one person may fill one or more of the human roles. For example, the system software administrator may be the only operator of the system. Figure 5 shows the inheritance of roles envisioned for the actors.

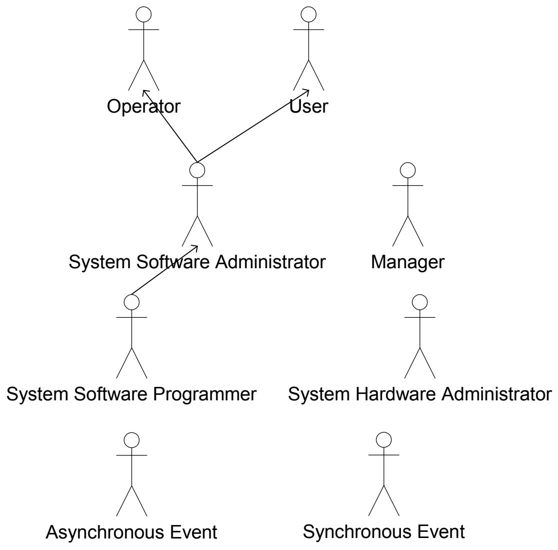

Operator
User
System Software Administrator
Manager
System Software Programmer
System Hardware Administrator
Asynchronous Event
Synchronous Event

Figure 5: The RAS Actors

Once the actors were identified, we brainstormed what they needed from a RAS system. These became the use cases. The entire use case diagram is given at the end of this paper. The short description of each use case was then written and is repeated here.

2.2.1 Use Cases Initiated by the User

Determine status of system resources – A user attempts to quantify the status of the MPP system resources to determine if the MPP system is available to run compute job(s).

Determine status of job(s) that were or are running – A user wants to determine the status of compute job(s) they had previously submitted to the MPP to be run.

Review the logs of jobs(s) that were run – A user wants to review the STDOUT, STDERR, job summary, and any other logs associated with submitted and terminated job(s).

Utilize application checkpoint/restart capability – A user wants to make an application utilize the checkpoint/restart capability of the MPP system to maximize availability of the application by minimizing the lost work when having to restart an application from a checkpoint.

Utilize application monitoring capabilities – A user wants to make an application utilize the monitoring capability of the MPP system to detect and resolve problems in an automated way.

2.2.2 Use Cases Initiated by the System Software Administrator (SSA)

Determine the status of jobs – An SSA wants to determine the status of all jobs running, queued, and otherwise that the system knows about.

Manage user jobs – An SSA wants to manage any/all of the jobs running, queued, and otherwise that the system knows about.

Determine the status of system software components – An SSA wants to know the status of any/all system software components, i.e., daemons, service agents, operating systems, communication layers, file systems, etc.

Determine the status of system hardware components – An SSA wants to know the status of some or all system hardware components.

Restart failed hardware/software components – An SSA has determined that there are hardware / software components in the MPP system in a failed state and wants to attempt to fix the component(s) by restarting them.

Startup/shutdown/reboot system components – An SSA needs to startup or shutdown or reboot system components. This includes scenarios of booting the entire MPP system from scratch, rebooting the entire system, etc.

Run tests and diagnostics – An SSA wants to run tests and diagnostics on any of the MPP system's hardware or software components or on the MPP system as a whole.

Data mine current and historical information – An SSA wishes to collect information on a specific topic or question. The SSA may be interested in statistics such as uptime, reliability, repair time, hardware replacement rates, compute processor utilization, memory utilization, disk utilization, communication network utilization, user job characteristics, etc. The system provides some predetermined reports, but allows for what-if and what-about questions.

Review system logs – An SSA wishes to review any logs or event histories for the MPP system.

Manage disk space – An SSA performs maintenance on the disks and associated file systems.

2.2.3 Use Cases Initiated by the System Software Programmer (SSP)

Analyze post-mortem a system software failure – The system software programmer attempts to determine the root cause of a software problem.

Obtain verbose debugging information – A need has arisen which requires detailed debugging information. The additional debug information may reduce system performance/throughput.

Upgrade system software – The SSP has determined that an upgrade to system software is necessary. The revised software is in hand and must be tested locally and then installed for production use. The change may require a complete new boot disk or only a portion of the system software may be replaced.

2.2.4 Use Cases Initiated by the System Hardware Administrator (SHA)

Diagnose questionable hardware – An SHA has identified questionable hardware and wishes to run diagnostics on the hardware to ascertain if there is a failure of some sort.

Add/Remove/Replace hardware components – An SHA has identified hardware that needs to be added, removed, or replaced in the system.

2.2.5 Use Cases Initiated by the Operator

Receive audible/visible notification of problems – Some components may provide an audible and/or visible indicator when a problem is detected. This indicator may be necessary because the component does not have the capability to report problems in a more electronic fashion to a central location. Or the indicator may be a backup/duplicate mechanism to an electronic message.

Check if system is operational – A simple, intuitive interface gives the operator a clear indication that the MPP is operational or not. It may be possible to extract additional details about what the problem area might be.

Follow notification procedure – The operator has evidence that there is a problem with the MPP. The operator will follow a prescribed procedure to notify the responsible party.

2.2.6 Use Case Initiated by the Manager

Retrieve performance statistics – A manager wishes to collect information on a specific topic or question. The manager may be interested in statistics such as uptime, reliability, repair time, hardware replacements rates, or resource utilization trends. The system provides some predetermined reports, but allows for what-if and what-about questions.

2.2.7 Use Cases Initiated by a Synchronous Event

Perform proactive system diagnostics – On a configurable schedule, some diagnostic tests are automatically run.

Backup selected files – Key static and dynamic system files are copied to physically separate media for safekeeping.

2.2.8 Uses Cases Initiated by an Asynchronous Event

Asynchronous event causes failure of system software service – An unexpected event caused a software service/daemon to fail or hang. The event could be a hardware glitch, invalid input, a toxic combination of valid input, a race condition, or something else.

Asynchronous event hangs/panics operating system – An unexpected event caused the operating system or one or more processors to hang or fail. The event could be a hardware glitch, invalid input, a toxic combination of valid input, a race condition, or something else.

Asynchronous event causes recoverable error – A hardware component detects an error. The operation is retried and succeeds.

Asynchronous event faults hardware with hot spare – A hardware component has a problem that can be fixed with a hot spare. The hot spare is put into service.

Asynchronous event faults hardware that can be isolated – A hardware component has a problem that is not critical to system operation. The component is isolated for subsequent repair.

Asynchronous event faults hardware that is a single point of failure – A hardware component fails that paralyzes the system sufficiently that no useful work can be done.

Asynchronous event causes environmental failure – An external, but necessary support service fails. The most likely examples are power or cooling.

Asynchronous event results in unknown event – The service processor collects one or more error reports. However, none of the predefined rules point to any specific problem.

Notify system software administrator of problems – A problem has been detected with the system and the responsible party needs to be notified.

2.2.9 Use Case Documentation

The final step in developing a complete use case model is to prepare the full documentation for each use case. Section 1.2 presented the documentation template used in the RAS study. The full documentation can be found in [7]. One sample is provided below.

Asynchronous event causes environmental failure
Description
An external, but necessary support service fails. The most likely examples are power or cooling.
Actors
Asynchronous event
Preconditions
None
Postconditions
An alert is generated and the system may be shut down.
Flow of Events
This use case begins when the environmental support service fails.

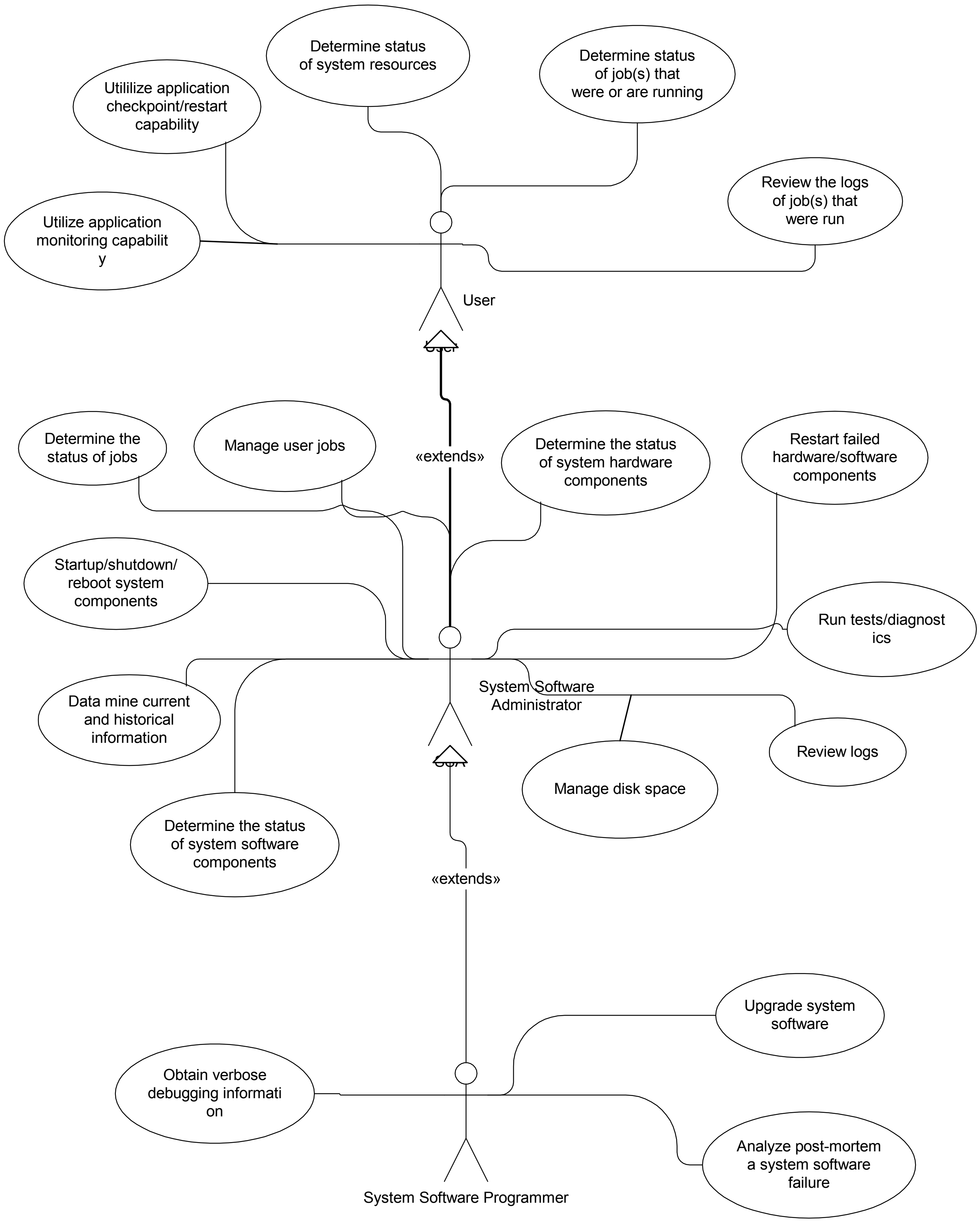
1. A system sensor is triggered.
2. If the triggered sensor is detecting loss of power and the alternate primary power source is still functioning, all power is derived from the alternate. The recovery action is reported to the service processor (SP).
3. If the triggered sensor indicates that all power is lost, the UPS automatically switches to battery and sends the recovery action to the SP.
4. Disks should also switch to battery when all AC is lost.
5. If a temperature alarm or multiple alarms are triggered, each report to the SP.
6. The SP processes the alarms. In the case of temperature alarms, the SP directs the appropriate fans/blowers to increase speed.
7. The SP generates an alert for immediate service using the "notify system software administrator of problems."
8. If the temperature is above a configurable limit or the system is operating exclusively on UPS, the system performs a graceful shutdown using the "shutdown the system use case".
9. If the temperature is above a possibly different configurable limit, the SP performs an automatic power off of the MPP and then possibly itself.

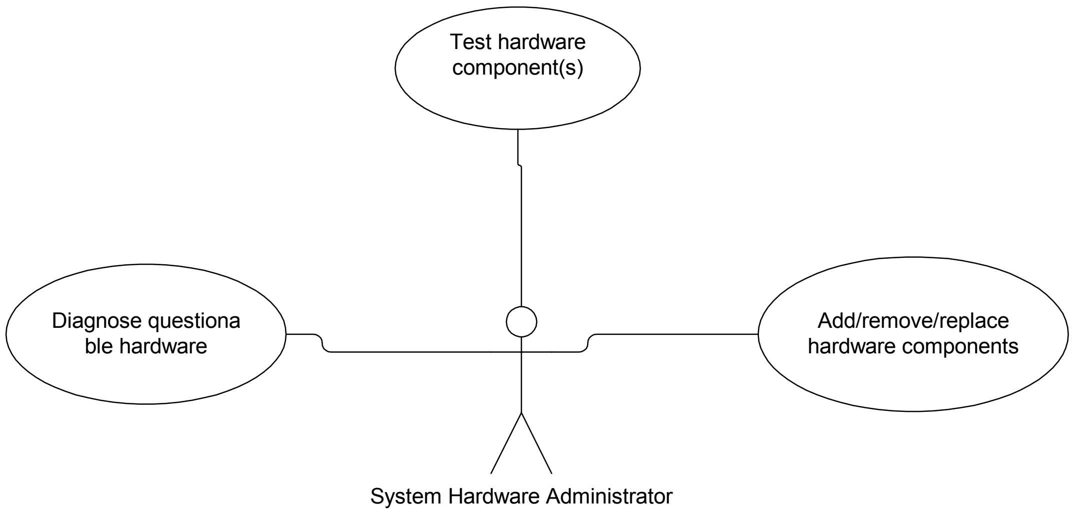
This use case ends when the alert is generated and the system is shut down (if necessary).

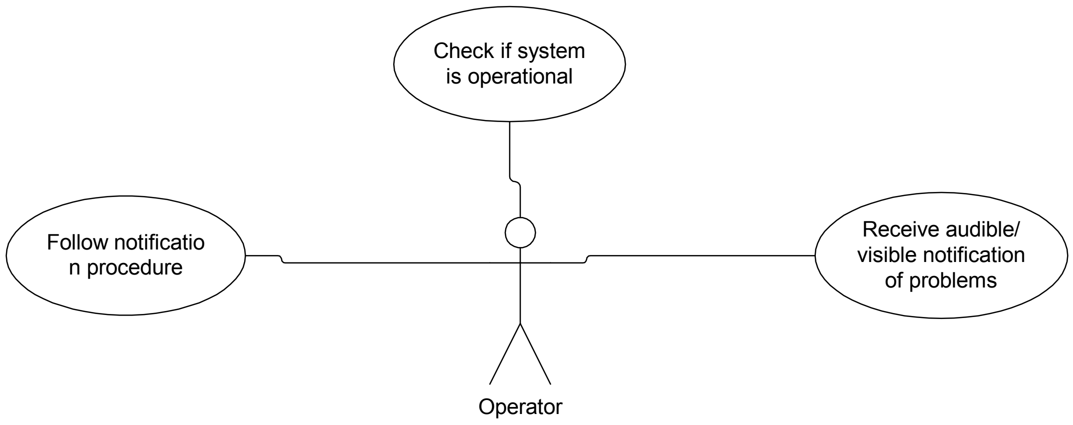
3.0 References

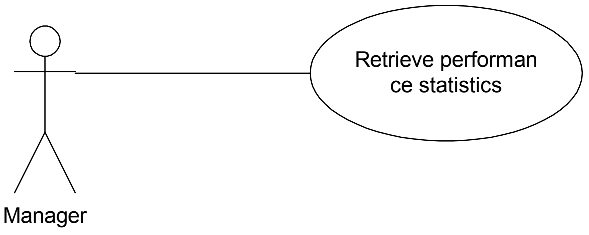
[1] Ivar Jacobson et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, 1992.
[2] http://www.uml.org.
[3] http://www.omg.org
[4] Grady Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, 1993.
[5] James Rumbaugh et al, *Object-Oriented Modeling and Design*, Prentice Hall, 1990.
[6] Geri Schneider and Jason P. Winters, *Applying Use Cases*, Addison-Wesley, 1998.
[7] Suzanne M. Kelly and Jeffry B. Ogden, An Investigation into Reliability, Availability, and Serviceability (RAS) Features for Massively Parallel Processor Systems, Technical Report SAND2002-3164, Sandia National Laboratories, October 2002.

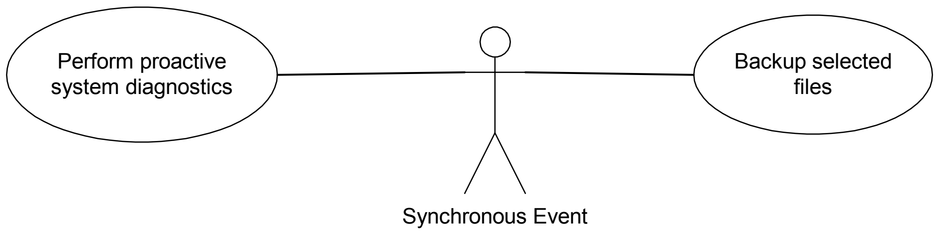
The Full Use Case Diagram of RAS for an MPP Environment is given on the following two pages. Ignore wrapping.


Determine status
of system resources
Utililize application
checkpoint/restart
capability
Determine status
of job(s) that
were or are running
Review the logs
of job(s) that
were run
Utilize application
monitoring capabilit
y
User
User
«extends»
Determine the
status of jobs
Manage user jobs
Determine the status
of system hardware
components
Restart failed
hardware/software
components
Startup/shutdown/
reboot system
components
Run tests/diagnost
ics
System Software
Administrator
Data mine current
and historical
information
Review logs
Manage disk space
SSA
Determine the status
of system software
components
«extends»
Upgrade system
software
Obtain verbose
debugging informati
on
Analyze post-mortem
a system software
failure
System Software Programmer


Test hardware component(s)
Diagnose questiona ble hardware
Add/remove/replace hardware components
System Hardware Administrator


Check if system is operational
Follow notificatio n procedure
Receive audible/ visible notification of problems
Operator


Retrieve performan ce statistics
Manager


Perform proactive system diagnostics
Backup selected files
Synchronous Event