SANDIA REPORT

SAND2013-10789
Unlimited Release
Printed December, 2013

Power/Energy Use Cases for High Performance Computing

James H. Laros III, Suzanne M. Kelly, Steven Hammond, Ryan Elmore, and Kristin Munch

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2013-10789
Unlimited Release
Printed December, 2013

Power/Energy Use Cases for High Performance Computing

James H. Laros III [#1], Suzanne M. Kelly [#], Steven Hammond [&],
Ryan Elmore [&], Kristin Munch [&]
[#] Sandia National Laboratories
[1] jhlaros@sandia.gov
[&] National Renewable Energy Laboratory

Power and Energy have been identified as a first order challenge for future extreme scale high performance computing (HPC) systems. In practice the breakthroughs will need to be provided by the hardware vendors. But to make the best use of the solutions in an HPC environment, it will likely require periodic tuning by facility operators and software components. This document describes the actions and interactions needed to maximize power resources. It strives to cover the entire operational space in which an HPC system occupies. The descriptions are presented as formal use cases, as documented in the Unified Modeling Language Specification [1]. The document is intended to provide a common understanding to the HPC community of the necessary management and control capabilities. Assuming a common understanding can be achieved, the next step will be to develop a set of Application Programing Interfaces (APIs) to which hardware vendors and software developers could utilize to steer power consumption.

Acknowledgements

The authors thank the reviewers of the first version of this document: David Jackson and Michael Jackson and Mary Smuin and Gary Brown, Adaptive Computing; Peter Bailey and Wei Huang and Bobbie Manne and Bill Brantley, AMD Research; Steven Martin, Cray, Inc.; Natalie Bates, Energy Efficient HPC Working Group; Suda Yalamanchili, Georgia Tech University; Nic Dube, HP; Mike Lang and Josip Lancaric, Los Alamos National Laboratory; and Mike Sheppy, National Renewable Energy Laboratory.

Contents

1 Introduction

Addressing anticipated HPC computational needs within reasonable power constraints requires significant advances in hardware power efficiency. To achieve the greatest efficiency from next-generation hardware at scale, software at many levels will need to coordinate and optimize the underlying hardware. While commodity pressures will drive useful innovations in this area that can be leveraged, our efforts are distinguished by our requirements at scale. The goal of this document is to identify the critical multi-level measurement and control requirements necessary to enable power and energy management of next-generation HPC platforms. We strive to cover a wide spectrum of needs, from facility to component. That said, our goal is also to bound or constrain the problem space, so that the reader obtains a firm grasp of the complete set of requirements. The ultimate goal, which is not covered in this document, is to evolve these requirements into a set of power-related Application Programming Interfaces (APIs) that can be implemented throughout an HPC system's software stack. While some APIs will likely be foundational and mandatory, not all APIs will need to be implemented at the same time.

The resulting API is not intended for a specific system. But rather, it is for general adoption within the HPC community. Community acceptance is always difficult to achieve, with no clear path to follow to ensure a definitive successful outcome. Our approach was to begin with requirements identification and send them out for review. The first review was done via email to a small set of reviewers. We incorporated feedback and produced this formal document. We are presenting the concepts contained herein at HPC workshops hoping to garner feedback and support. Concurrent with the workshops, we are drafting the API specification and selecting portions for reference implementations.

This paper documents our work in requirements identification. The authors elected to create use cases to model the ways power measurement and control capabilities will be used in HPC systems. The requirements are naturally captured in the use case documentation. Use cases were introduced in the 1990's as a down-to-earth technique for specifying the way in which a system would be used. They are considered a superior approach to identifying what a system will do. They are in contrast to a laundry list of specifications that are prone to mis- or broad interpretation by the reader. The term use case is no longer specific to its origin [2] that became part of the Unified Modeling Language (UML) Specification developed by Booch, Jacobson, and Rumbaugh [1]. UML later became specification ISO/IEC 19501:2005. This document employs Version 1.4 of the use case model within the UML standard. Our primary guidance document was [3].

Section 2 will present an overview of use cases and how we use them to express requirements and interfaces. Section 3 will specify the use cases for power monitoring and control within an HPC system. The Appendix provides some scenarios we drafted prior to creating the use cases. The scenarios cover multiple interactions, and therefore multiple use cases were derived from them. They are not a complete set of scenarios. We document them here as they were our first exploratory step before creating the use case diagram.

2 Use Case Diagrams

The following is a description of what a Use Case Diagram (UCD) represents in the context of this document. The term UCD has become quite overloaded. We will continue to overload the term as described below.

In this document a UCD is comprised of a small set of components (Actor, System, Arrows and one or more use cases). The components contained in a UCD are described in the following subsections. In the context of this document a UCD is intended to describe a very high level of interaction between a single Actor and a single System. Taken in mass, all of the Actor/System pairs will describe the high level view and scope of our system. The purpose of this exercise is ultimately to drive the definition of an Application Programming Interface (API). Taking this top-down approach will describe all of the things our system will be required to accomplish at a high-level. Additionally, this approach will capture information flow through the entire system. We hope to err towards a first pass that is too high-level since further decomposition of use cases that cover the complete scope will be more natural than omiting functionality altogether that when added can be disruptive and time consuming to the process.

2.1 Actor

Actors in UCDs can be thought of as users of the Systems depicted in the UCDs. It is very natural to think of Actors as people, in fact Actors are represented by stick figures in UCDs (see Figure 1). Actors are often people but Actors can, and frequently are in this document, entities that are also Systems to other Actors. For example, within the scope of this effort the operating system is a System to a number of Actors but is also an Actor in the UCD that captures the interaction between the operating system and the hardware. This becomes evident in the Top Level UCD (Figure 4) where all the UCDs covered in this document are combined.

2.2 System

Systems in UCDs can be thought of as the entity being used to accomplish something. Common examples of a System are a camera, or a telephone. A person (Actor) uses a camera (System) to take a picture for example. In UCDs the System in the Actor/System pair is represented by a box which includes the name of the System (see Figure 1). The system will contain the use cases that describe the interaction between a specific Actor and a specific System only. In many cases a System will become the Actor in a separate UCD covering that Actor/System interaction.

2.3 Arrow

Three different arrow types are used in the UCDs contained in this document. The arrow used to connect the Actor and each individual use case indicates the Actor/System interaction of that specific use case. The direction of the arrow is always towards the use case. Some use case models use the direction to indicate who initiates. That is not the case in our model. Two additional arrows are used within the System box to indicate subtle features of use cases. The *includes* arrow is used to show that the functionality of the use case is including additional information to accomplish its goals. Information necessary for a use case can simply be included in the use case description but in some cases we have chosen to use the includes arrow to highlight information flow through the overall system. The *extends* arrow is used to depict a use case that is an extension of a previously existing use case. For example, a typical activity on an HPC platform is run job. We use the extends arrow to recognize not only that this is an existing use case but to also indicate that we will only be covering the details of the extended use case (run power aware job) in our coverage.

2.4 Use Case

A use case, what appears in a single bubble in a UCD (see Figure 1), should be a goal oriented activity. That said, finding the correct level to address with a single use case can be difficult. The following example

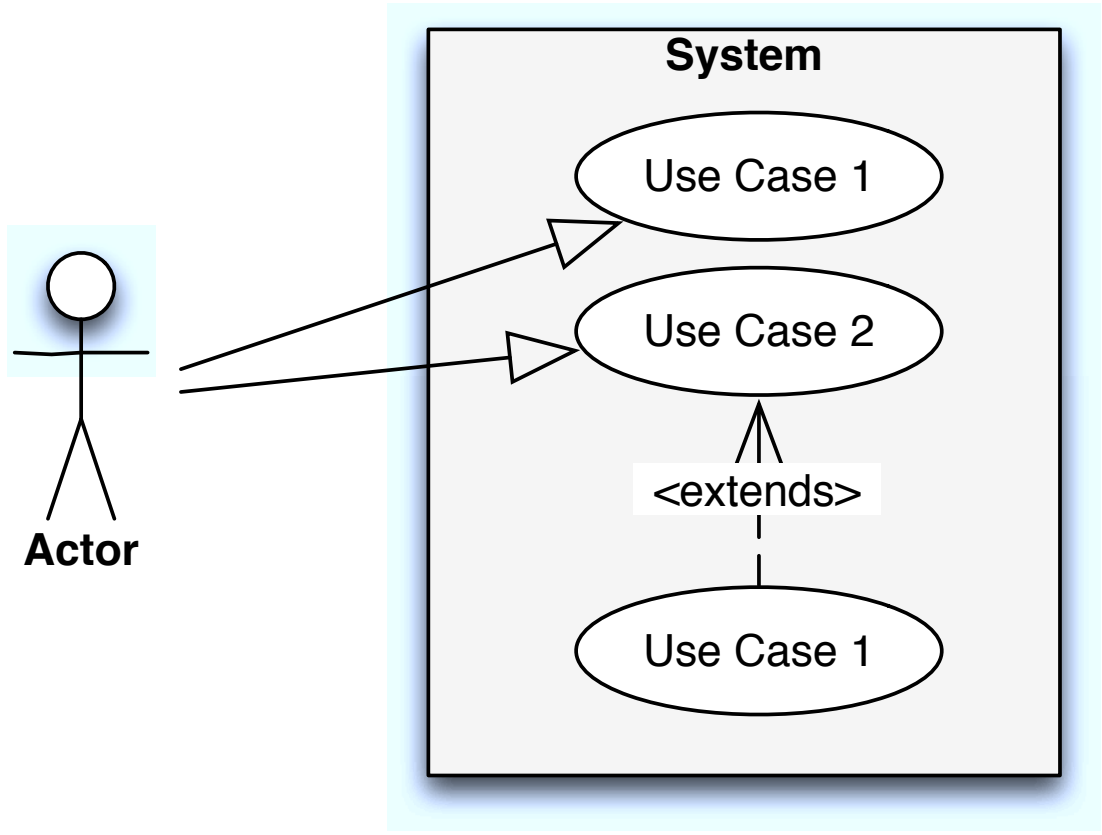

Figure 1. Example Use Case Diagram

helped us find what we hope to be the proper level when defining use cases in this document. Suppose we are describing the interaction between a person (Actor) and a telephone (System). To simplify things lets assume this is an old fashion land line phone, circa 1980's. Thinking about Actor/System interaction we could choose to include a use case for each of these activities: *pick up receiver, say hello, have conversation, say good-bye and replace receiver*. A UCD that describes this interaction would look like Figure 2.

The actions described above are all part of a sequence of interacting with a telephone, in this case answering the phone. The interaction can be more simply represented, from a person's perspective, by a use case named *answer phone*. One of the benefits of describing a use case at this level is that as technology evolves the use case remains the same. If we assume the person in our UCD is now interacting with an iPhone®the sequence of steps to answer the phone would be more like: *pick up iPhone, press answer box on screen, say hello, have conversation, say good-bye, press end box on screen*. Those of us old enough to have interacted with a phone in the 1980's who now use an iPhone likely still think of the interaction with the phone at the level of - answer phone - regardless of the different sequence we go through.

Another interaction (use case) between a person and a Telephone at the same level as *answer phone* is *place call*. A key distinction is that *answer phone* and *place call* have different goals. The sequence of placing a call is only slightly different than the sequence of answering a phone. One difference is the step of dialing the number of the person you want to contact. Both *answer phone* and *place call* share many of the

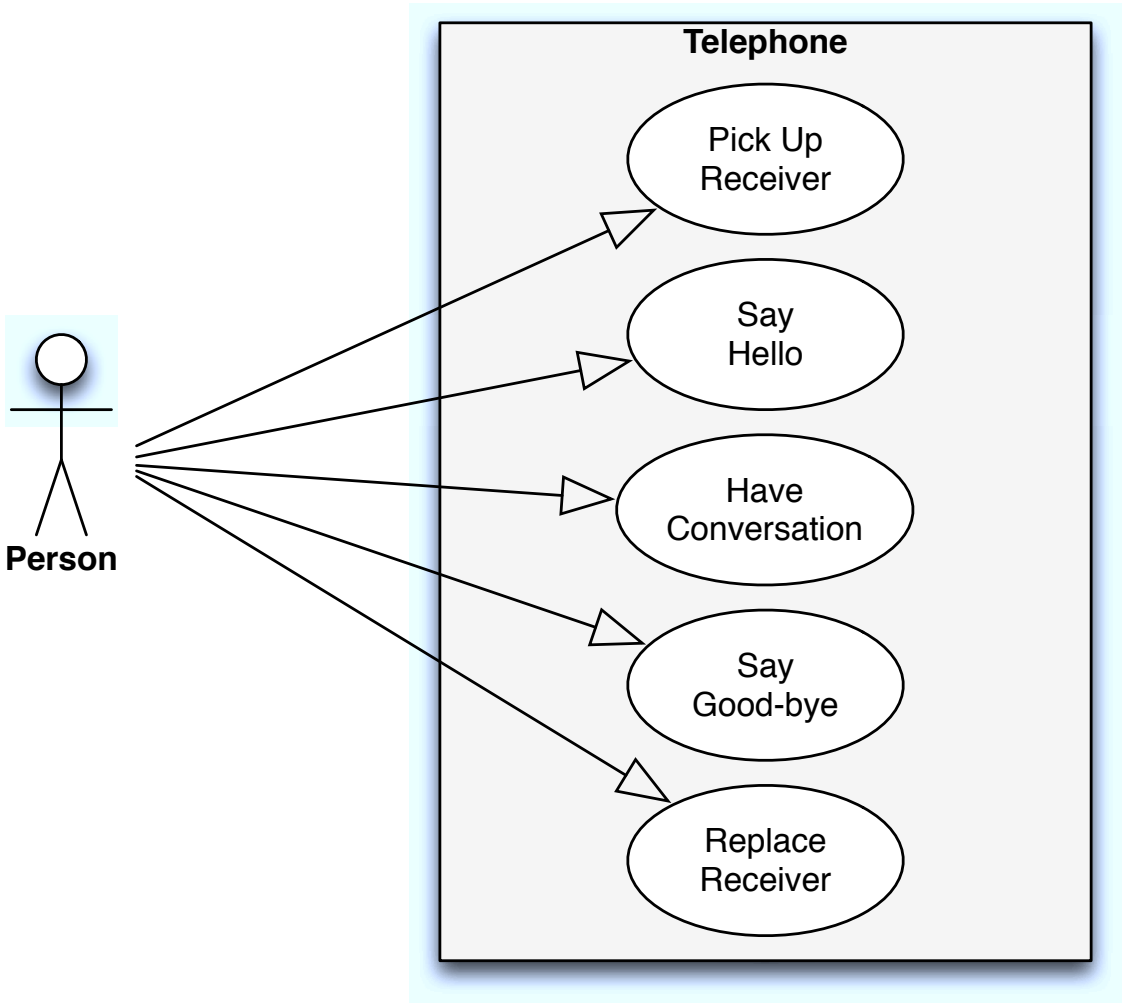

Figure 2. Person/Telephone UCD - Too Detailed

same activities. This is another indication that we are specifying use cases at the proper level. When the use case is documented this step will be enumerated in the flow of events (see Section 2.5). When the API is created only a single pick-up-receiver function will be necessary to implement for both *answer phone* and *place call*. This detail will emerge during the process of documenting the use case. Note that as the API evolves the pick-up-receiver function might be deprecated or extended to include pressing the phone button. Regardless, these are details that are best left to the development, and evolution, of the API. The UCD that we feel describes the correct level of interaction between a person (Actor) and a telephone (System) is represented in Figure 3. There may be additional use cases for this UCD. Our goal would be to include all of the interactions between a person and a telephone so that a complete API could then be generated from the UCD and use case descriptions.

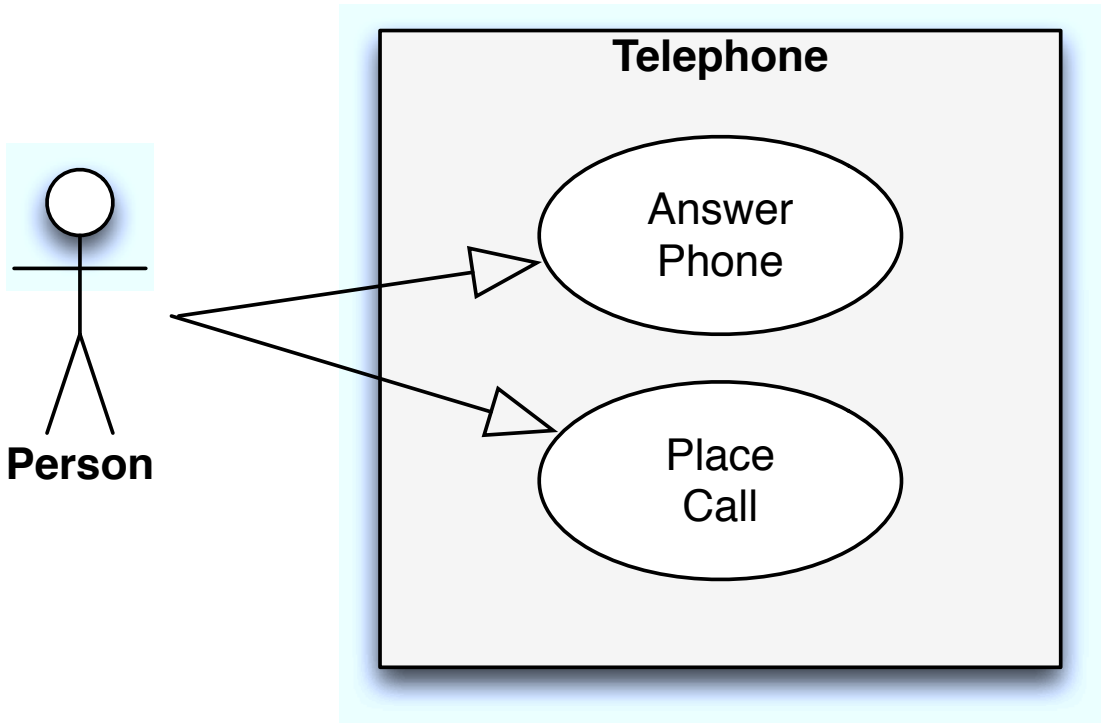

Figure 3. Person/Telephone UCD - Correct Level

2.5 Generic Use Case Text

The Unified Modeling Language allows considerable latitude in the text documentation for each use case. Numerous templates are available on the Internet suggesting what to include in the text. Most templates include a description section and a flow of events. What follows is the text format we adapted for our use case diagram.

Actor	Enter the Actor name here (from the UCD) (Section 2.5.1)
System	Enter the System name here (from the UCD) (Section 2.5.2)
Use Case	Enter the name of the Use Case here (from the UCD) (Section 2.5.3)
Description	Enter the description of the Use Case here (Section 2.5.4)
Trigger	Enter the trigger of the Use Case here (Section 2.5.5)
Flow of Events	Enter the flow of events, or the happy state, here. This does not account for failures. (Section 2.5.6) 1. Step 1 text 2. Step 2 text
Alternative Paths	Enter the alternative paths here, these extend the Flow of Events steps (Section 2.5.7) 1a. Alternative Flow of Events step 1 2a. Alternative Flow of Events step 2 2b. Alternative Flow of Events step 2
Frequency	Enter one or multiple Frequency descriptions here (Section 2.5.8)
Input Data	Enter input power data fields (Section 2.5.9)
Output Data	Enter output power data fields (Section 2.5.10)
Pre Condition	Add condition here (Section 2.5.11)
Post Condition	Add condition here (Section 2.5.12)
Power API	Enter yes or no here (Section 2.5.13)

2.5.1 Actor

The Actor name in the Actor/System pair represented in the UCD, e.g. person in the person/telephone UCD example (Figure 3). See Section 2.1.

2.5.2 System

The System name in the Actor/System pair represented in the UCD, e.g. telephone in the person/telephone UCD example (Figure 3). See Section 2.2.

2.5.3 Use Case

The name of the specific use case that will be described, directly from the bubble in the UCD, e.g. answer phone in UCD example (Figure 3). See Section 2.4.

2.5.4 Description

Free form description of what this use case is about. This will be refined over time to be more consistent as the document evolves.

2.5.5 Trigger

An example or examples of what triggers or initiates the use case, i.e. when the phone (System) rings, it triggers the person (Actor) to answer the phone.

2.5.6 Flow of Events

List of events that comprise the use case. Again, finding the correct level can be challenging. In the use case answer phone a list of events would include the sequence listed in Figure 2. While this level of detail is claimed to be too specific for individual use cases it is just about right for sequence of events. Note that the flow of events is considered the *happy path*, meaning the flow when everything goes right. The flow of events will be a numeric list, i.e. 1, 2, 3, through N. Too many steps in the flow of events might possibly indicate that the use case is too high level.

2.5.7 Alternative Paths

The alternate path is used in conjunction with the flow of events to describe what happens when the flow deviates from the *happy path*. For example, if something goes wrong in step 1 in the flow of events the alternate path 1a will describe what happens down this path followed by 1b, 1c, 1d through 1x.

2.5.8 Frequency

The frequency is how often the use case might occur, i.e. the actor (person) answers the phone (system) at the frequency the phone rings. Typically, frequency is largely out of the control of the actor, or the actor does not initiate the interaction which can be useful information in designing the API. Alternatively, place call is initiated by the actor (person). This could be in response to a missed call or just because the person desires to reach another person.

2.5.9 Input Data

A list or description of the input data needed for the use case. This information can be very important in designing the API. The flow of information through the system can also be seen here. Questions like, where did this input data come from will help to ensure that other interactions provide the appropriate data flow, or output data. *An important task of the next draft of this document is to flesh out these input and output data fields for inclusion in the API.*

2.5.10 Output Data

A list of possible output data produced by the interaction. Again, this information can be very important in the design of the API. As with input data it is often important to trace the output data to input data in associated use cases.

2.5.11 Pre Condition

A condition or conditions that must be true before the use case is initiated, i.e. the receiver must be on the cradle before someone calls and the phone is answered by the actor (person). This primarily addresses the happy path.

2.5.12 Post Condition

A condition or conditions that are true after the use case is completed, i.e. the receiver is on the cradle after the actor (person) answers the phone and completes the conversation. This primarily addresses the happy path.

2.5.13 Power API

Yes or No to indicate if this use case will eventually be implemented as part of the power API. To illustrate the boundaries of the system we will be covering use cases that will not become part of the API or in some cases use cases that describe conversations where information or input data originates but will not be implemented as part of the power API.

3 Power API Use Case Diagrams and Text

The following sections cover the individual High Performance Computing System (HPCS) Actor/System pairs that define the primary interactions that have been identified. These pairs along with their use cases will subsequently be used as the foundation of the Power API specification.

3.1 Top Level Use Case Diagram

The Top Level Use Case Diagram (UCD) combines all UCDs that appear in this document simplified into Actor/System pairs. See Figure 4. One of the uses of the Top Level UCD is to understand the flow of high level scenarios and understand the interactions or interfaces necessary to accomplish the scenario. The Top Level UCD uses two different icons to represent whether an entity is an Actor or a System. The expanded UCDs can be found in later sections.

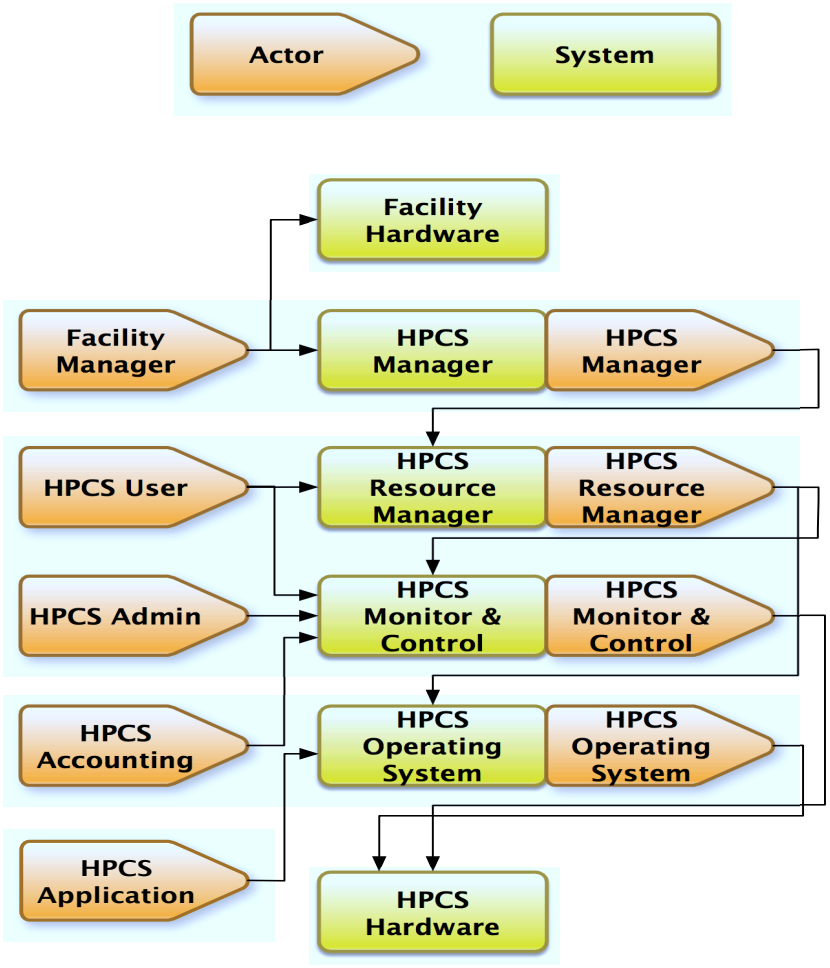

Figure 4. Top Level Use Case Diagram representing the culmination of all Use Case Diagrams covered.

3.2 Combined Use Case Diagrams

The following sections will be addressing each system and actor pair. While unfortunately an eye chart, we offer the following combined diagram of the use cases. You may find it a useful reference while reviewing the subsequent sections.

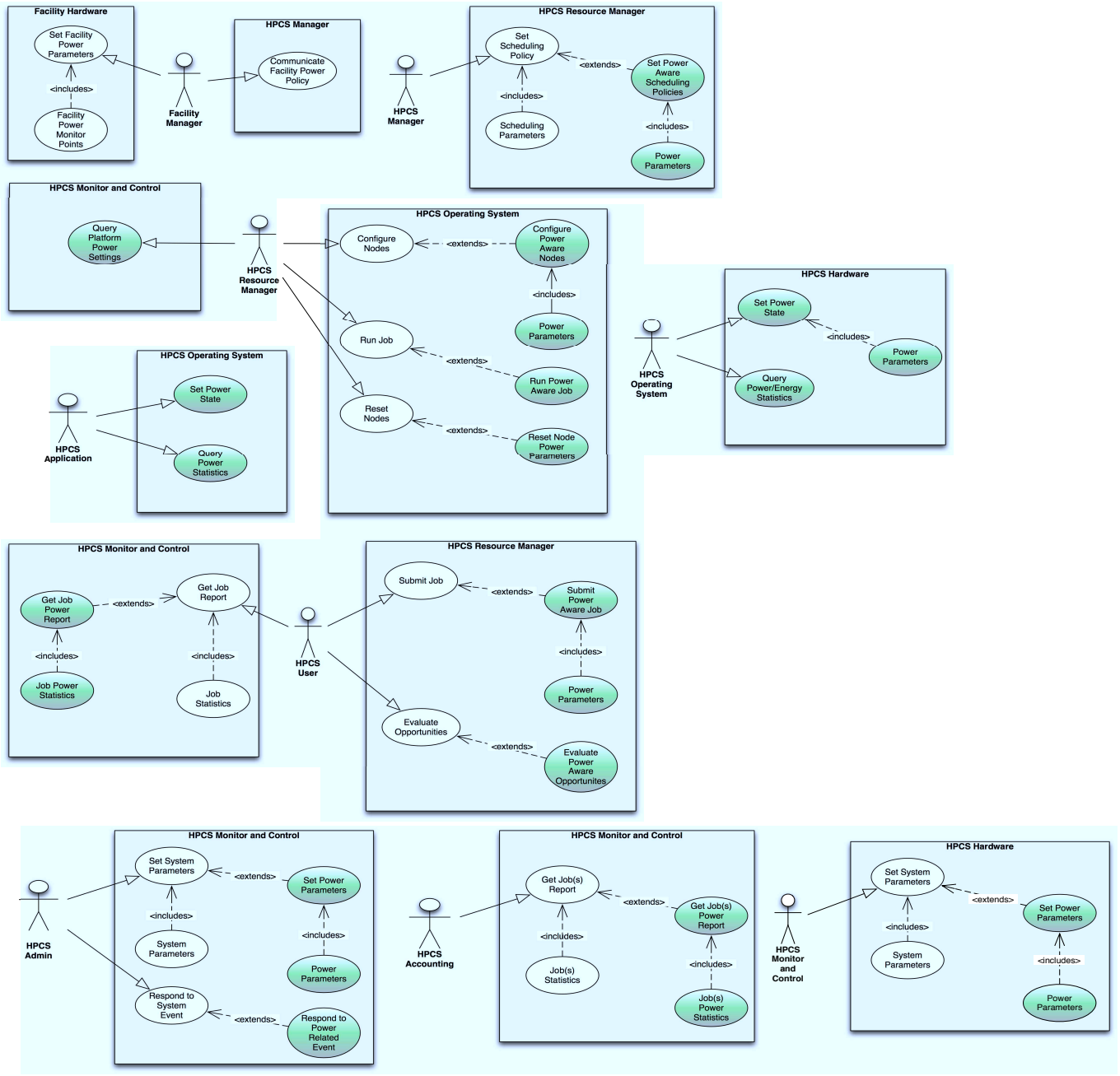

Figure 5. Combined Low Level Use Case Diagram collecting all Use Case Diagrams covered.

3.3 Actor: Facility Manager
System: Facility Hardware

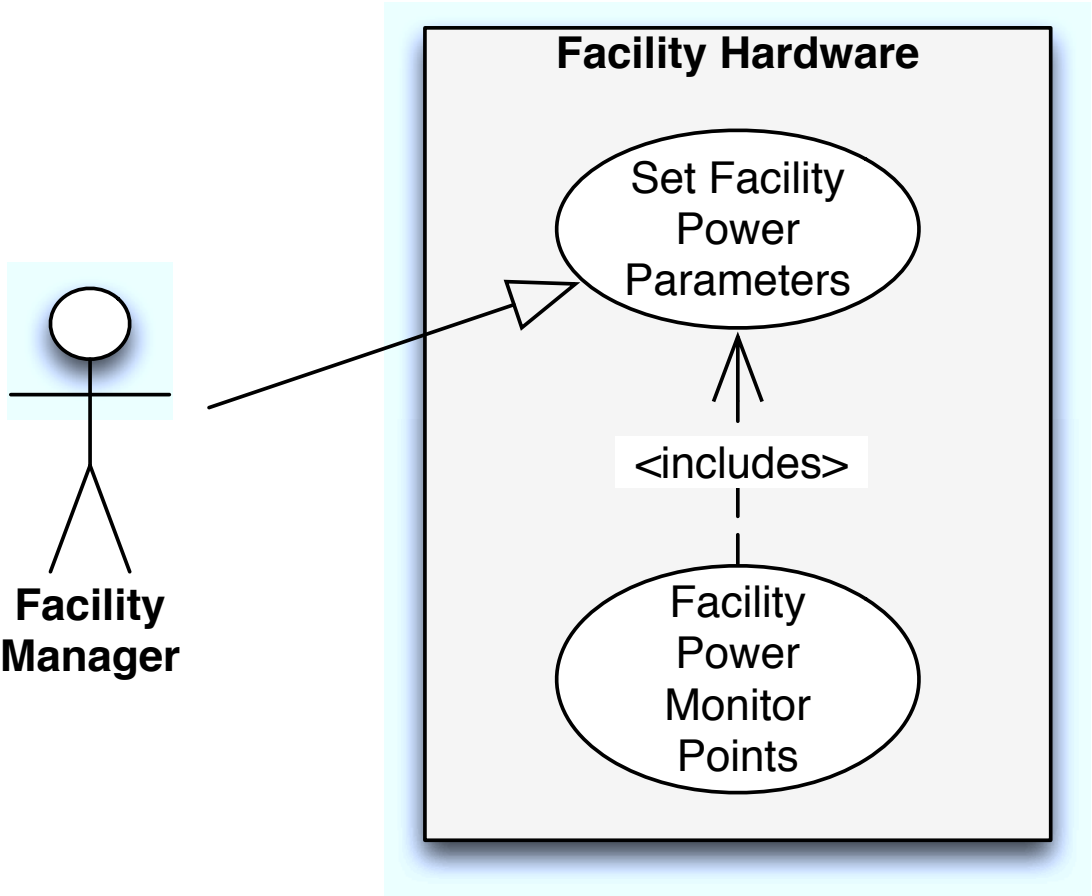

Figure 6. Facility Manager ⇒ Facility Hardware Use Case Diagram

3.3.1 Use Case: Set Facility Power Parameters

Actor	Facility Manager
System	Facility Hardware
Use Case	Set Facility Power Parameters
Description	This use case is intended to represent the interaction of the Facility Manager with the Facility Hardware at a high level. This interaction includes collecting information that will be used as part of the conversation with the HPCS Manager. Facility changes will be implemented based on the dialog with the HPCS Manager regarding computational requirements (for example). This interaction illustrates one of the boundaries of the scope of this project and is important in understanding the source of many of the parameters that will drive the use of the underlying resource. This interface will not be included in the API being developed for the HPCS. Facility Hardware is the facility mechanical, electrical and power infrastructure supporting the High Performance Computing System(s) (HPCS), including pumps, fans, and cooling apparatus.
Trigger	Feedback from HPCS Manager or from other external events that requires modification of Facility Hardware
Flow of Events	Resulting from communications with HPCS Manager (see Section 3.4) or compilation of data from external sources including upcoming environmental conditions, commercial power supply constraints and costs, etc. 1. Facility Manager and HPCS Manager agree upon energy policy 2. Facility Manager implements change to Facility Hardware 3. Successful Facility modifications
Alternative Paths	3a. All or portions of the Facility modifications unsuccessful 3b. Determine specific cause of failure 3c. Correct failure 3d. Re-implement change
Frequency	This might be done daily or following standard procedures or perhaps once or twice a week.
Input Data	New power policy parameters derived from environmental data, facility or commercial power outages (for example).
Output Data	Current power policy parameters
Pre Condition	Power parameters are at state A
Post Condition	Power parameters are updated to state B
Power API	No

3.3.1.1 Notes The primary purpose of this Actor/System interaction and use case is to document the important role of the facility. As power becomes a more important factor in the ultimate resource management of individual platforms and the site in general, information regarding the cost of power, availability and even permissible range of power fluctuation (not an exhaustive list) will be critical information that will be taken into account when scheduling job execution priorities. We document this interaction since this is an important source of information that affects scheduling and other factors related to platform resource management. Information will also flow from the HPCS Manager to the Facility Manager which will affect facility settings. We do not intend to completely cover the details of facility management at all sites. We consider this one of the boundaries in the scope of our coverage.

3.4 Actor: Facility Manager
System: HPCS Manager

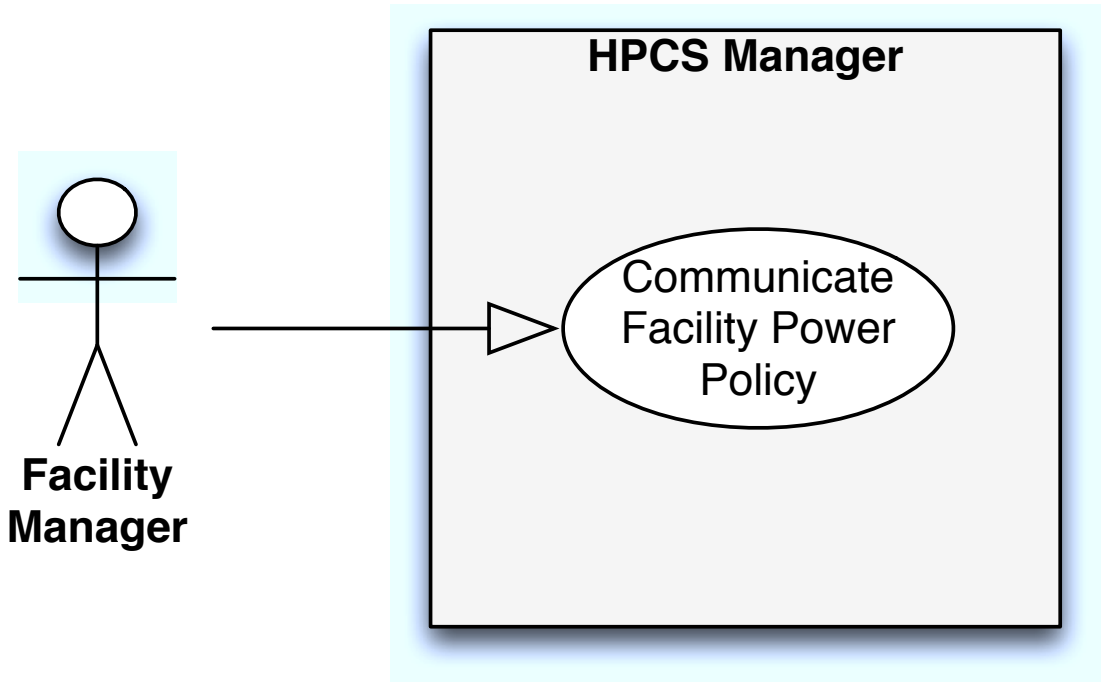

Figure 7. Facility Manager ⇒ HPCS Manager Use Case Diagram

3.4.1 Use Case: Communicate Facility Power Policies

Actor	Facility Manager
System	HPCS Manager
Use Case	Communicate Facility Power Policies
Description	The Facility Manager communicates (conversation or software) the power policy to the HPCS Manager. This is a two way discussion. The HPCS Manager also communicates priorities to the Facility Manager which the facility manager will then use when interacting with the facility hardware by applying appropriate settings to support future power and environmental needs (for example).
Trigger	1. To satisfy site power requirements the Facility Manager communicates new power parameters to the HPCS Manager. This is meant as an example rather than a complete list of the communications between this Actor and System.
Flow of Events	1. Facility Manager communicates policy change to HPCS Manager 2. HPCS Manager modifies parameters in the HPCS Resource Manager to reflect new policy (see Section 3.5). 3. Successful communication of policy changes
Alternative Paths	3a. Failure in communicating policy changes 3b. Repeat attempt to communicate policy changes until successful 3c. Successful communication of policy changes
Frequency	Possibly once or twice a week. This will be a site specific practice.
Input Data	Power policy parameters and values
Output Data	Agreed-to Power policy parameters and values
Pre Condition	Power policy set at A
Post Condition	Power policy set at B (assume impact on HPCS Resource Manager)
Power API	No

3.4.1.1 Notes This Actor/System pair is intended to document the communication that will occur between the Facility Manager and the HPCS Manager. The Facility Manager is both the source of site facility information that will ultimately affect platform resource management and the recipient of information from the HPCS Manager regarding platform requirements. This communication is vital to future platform power resource management.

3.5 Actor: HPCS Manager
System: HPCS Resource Manager

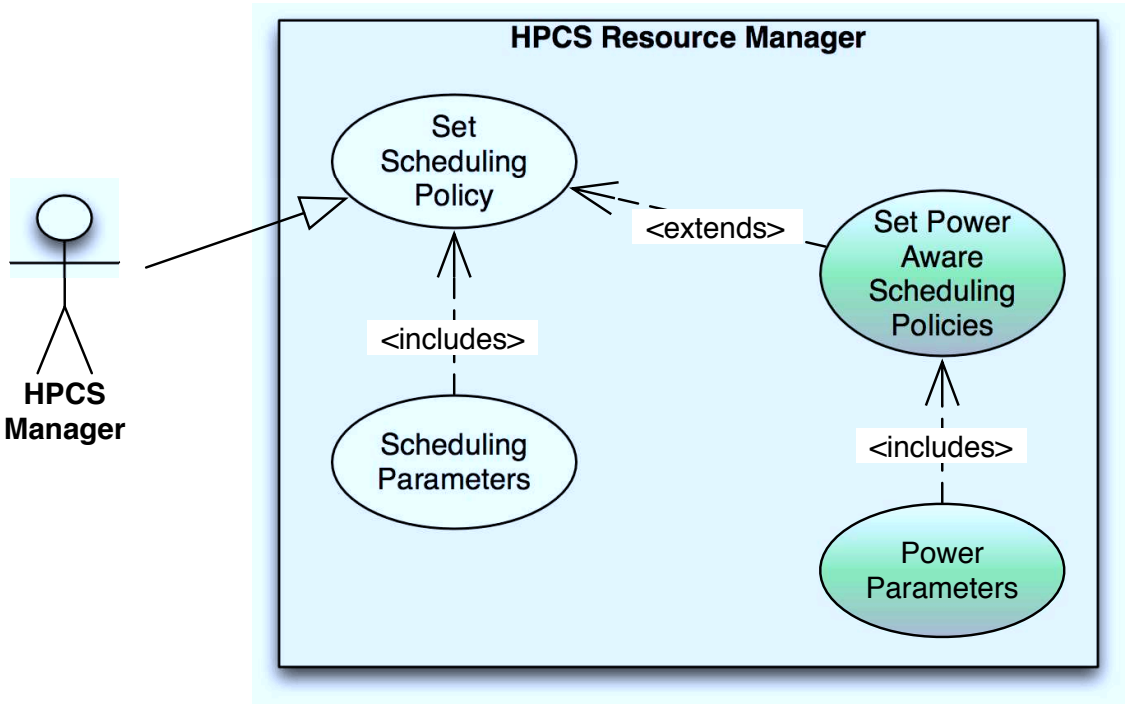

Figure 8. HPCS Manager ⇒ HPCS Resource Manager Use Case Diagram

3.5.1 Use Case: Set Power Aware Scheduling Policies

Actor	HPCS Manager
System	HPCS Resource Manager
Use Case	Set Power Aware Scheduling Policies
Description	The HPCS Manager sets the power related scheduler policies in the HPCS Resource Manager. The policies may be based on a yet-to-be-defined "fused" metric which would likely include parameters regarding power, time of day, node hours available, etc.
Trigger	1. Due to a change in the power policy, a change is required in the scheduling policies (see Section 3.4.1).
Flow of Events	1. HPCS Manager sets scheduling policies via the HPCS Resource Manager 2. HPCS Resource Manager returns policy settings to the HPCS Manager
Alternative Paths	2a. Return Failure 3b. Resubmit policy change 3c. Return success or failure
Frequency	Daily or as needed, likely site dependent.
Input Data	Power policy parameters and values
Output Data	parameters successfully configured
Pre Condition	Scheduler parameter set A
Post Condition	Scheduler parameter set B New power policy applied to HPCS Resource Manager
Power API	Yes

3.5.1.1 Notes For the purposes of this document we are combining a few functionalities like the batch scheduler and some traditional functionality of the runtime system into a System (in the Actor/System sense) called the HPCS Resource Manager. In the use case listed we are recognizing the importance of a batch scheduler in platform resource scheduling and utilization. The policies may be based on a yet-to-be-defined "fused" metric which would likely include parameters regarding power, time of day, node hours available, etc. This is one possible way schedulers might weigh all of the important variables that must be considered when scheduling a resource. For the purposes of this document power and energy are primary considerations but the concept of the fused metric is meant to consider many varying and sometimes conflicting considerations. Scheduling considerations can be prioritized by weighing them more heavily than others. Considerations that are not at all important could be given a weight of 0 so they do not affect the scheduling calculation. While not the primary purpose of this effort, the fused metric concept will be fleshed out further in subsequent versions of this document. It is important to recognize the flow of information in this use case. The *Power Parameters* that are supplied by the HPCS Manager are obtained in part from communications with the Facility Manager.

3.6 Actor: HPCS Resource Manager
System: HPCS Monitor and Control

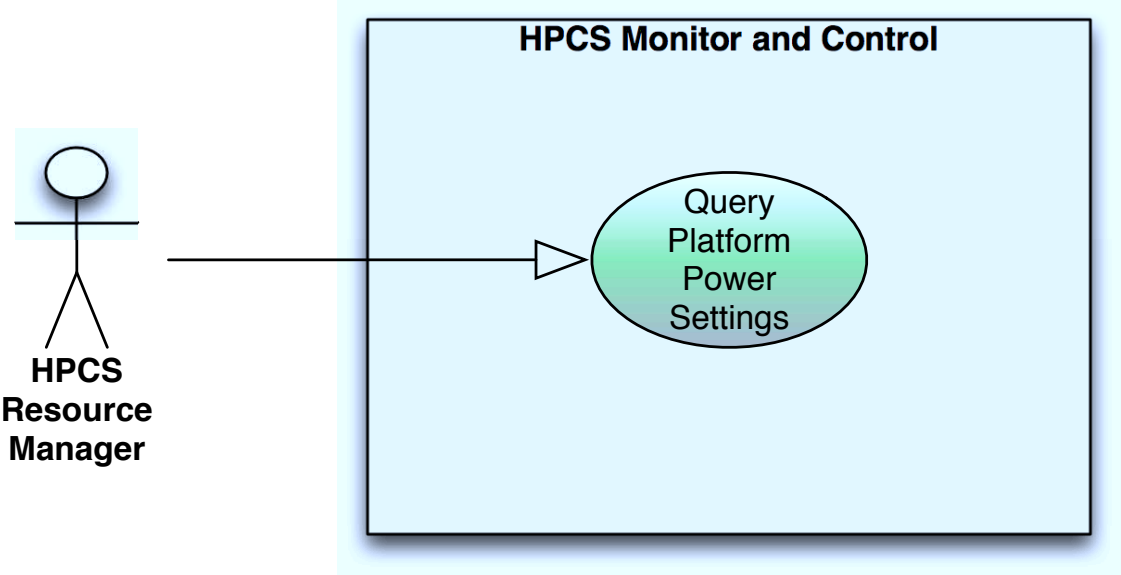

Figure 9. HPCS Resource Manager ⇒ HPCS Monitor and Control Use Case Diagram

3.6.1 Use Case: Query Platform Power Settings

Actor	HPCS Resource Manager
System	HPCS Monitor and Control
Use Case	Query Platform Power Settings
Description	The HPCS Resource Manager interfaces with the HPCS Monitor and Control system to determine platform power settings. For example, only the Monitoring Control system may know about a constraint on a power maximum for a cabinet or the entire system. This information will be used in determining which power aware jobs can be launched on which hardware to maintain the overall power policies that have been set for the current operating environment.
Trigger	1. HPCS Resource Manager performs this query on a set interval or possibly before making a determination of what job to run next
Flow of Events	1. HPCS Resource Manager (scheduler) requests platform power settings 2. HPCS Monitor and Control system successfully returns current platform power settings
Alternate Paths	1a. HPCS Monitor and Control system is unable to return current platform power settings
Frequency	On pre-set schedule or before running a job (for example)
Input Data	Query settings such as current system power cap, worst case uncapped power, target system min/max power
Output Data	Current platform power parameters
Pre Condition	None
Post Condition	None
Power API	Yes

3.6.1.1 Notes It is likely that the HPCS Monitor and Control system has the platform power settings cached and does not need to query the HPCS Hardware upon each individual request. When the platform power parameters are changed or upon an asynchronous event that changes them the cached copy of current parameter settings will likely be updated.

A note on runtime systems: More and more intelligence is being integrated into a software component referred to as a "runtime system". This conceptual component can be implemented in several domains. It can be a root-level daemon, linked as a library to the application, and/or embedded in the operating system. The resource manager, application, and operating system sections of this document apply to the pieces that might be implemented in each domain.

3.7 Actor: HPCS Resource Manager
System: HPCS Operating System

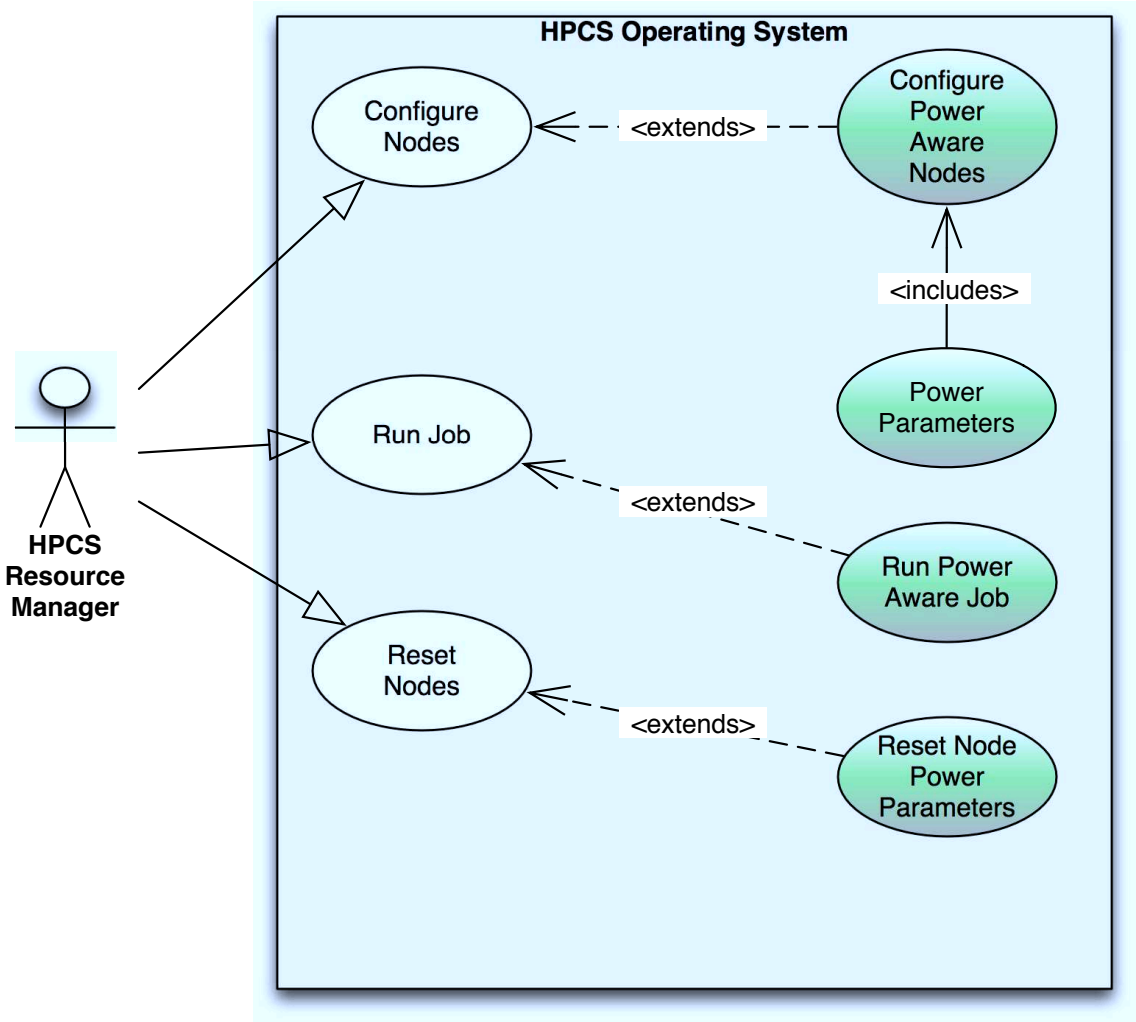

Figure 10. HPCS Resource Manager ⇒ HPCS Operating System Use Case Diagram

3.7.1 Use Case: Configure Power Aware Nodes

Actor	HPCS Resource Manager
System	HPCS Operating System
Use Case	Configure Power Aware Nodes
Description	The HPCS Resource Manager interfaces with the HPCS Operating System to set Power configuration parameters on nodes prior to running a Power Aware job (this could also be done dynamically during job execution). For example, the CPU frequency could be changed up or down. Or the power of an attached SSD could be turned on or off. This use case extends the generic use case of **Configure Nodes** in Figure 10 by configuring nodes to run Power Aware jobs. Additional Power Parameters will be communicated to the HPCS Operating System to be used during job launch.
Trigger	1. The scheduler portion of the HPCS Resource Manager identifies a job to start. (see Section 3.14.1)
Flow of Events	1. HPCS Resource Manager (scheduler) requests configuration of Power parameters. 2. HPCS Operating System successfully configures nodes in preparation for job launch
Alternate Paths	1a. HPCS Operating System returns failure 1b. Depending on Policies job may or may not be launched
Frequency	For every Power Aware job launched
Input Data	Power Parameters
Output Data	Power Parameters successfully configured
Pre Condition	Operating System able to receive requests for Power Aware Parameters
Post Condition	Power Aware Parameters are set
Power API	Yes

3.7.1.1 Notes The use case Configure Power Aware Nodes is addressing the potential that parameters that will affect power and energy may be set prior to job launch. A straight forward example is that a specific application has been analyzed and found to run efficiently at a lower frequency P-state. Prior to job launch the scheduler would request (of the operating system) that all nodes allocated to this job to be changed to this P-state. The job will then be launched on these nodes and upon completion the nodes will be returned back to a default configuration which will be addressed later by the use case Reset Nodes depicted in Figure 10.

It is also possible that the resource manager will dynamically change these parameters during job execution. The scenario of the utility company requesting an immediate reduction in power consumption is not completely addressed in this document.

In order to make the use case realistic, we elected to hard-code the term "node" in the title. The final API may elect to specify a more generic hardware object in the function call. The important message is that power management must be coordinated across multiple allocatable units, which are typically nodes. While there will be exceptions, such as for client/server applications within one job, the typical HPC scenario is a single homogeneous application. All nodes, including GPUs should be coordinated. At this time, the level of hardware exposure in the API is unclear.

3.7.2 Use Case: Run Power Aware Job

Actor	HPCS Resource Manager
System	HPCS Operating System
Use Case	Run Power Aware Job
Description	A job is starting to run that specified some type of power configuration. The Operating systems on HPCS components assigned to the job need to ensure the power specifications are satisfied throughout the life of the job.
Trigger	The scheduler portion of the HPCS Resource Manager identifies a job to start.
Flow of Events	1. An application is ready to execute and provides the OS on each computing resource with its power specifications. 2. The OS ensures it has the appropriate initial environment created in 3.7.1. 3. The OS adapts to additional requests for changes in power settings as long as they are within the specifications provided at job launch. 4. Job may request power information from the OS. 5. Job completes (success or failure is irrelevent to this use case.
Alternate Paths	2a. The specification is invalid or unachievable. 2b. An error message is written and OS initiates job abort. 3a. Invalid power change request is denied with an error return but job continues. 4a. Invalid power information request results in an error return, but job continues.
Frequency	For every Power Aware job launched
Input Data	Job Identifier and power configuration specification
Output Data	Any collected power data stored (implementation dependent).
Pre Condition	Use case 3.7.1 has completed.
Post Condition	Power aware job is complete.
Power API	Yes, for passing the power specifications to the OS.

3.7.2.1 Notes This use case could be considered the center of the Power API. All APIs exist to support this use case.

3.7.3 Use Case: Reset Nodes

Actor	HPCS Resource Manager
System	HPCS Operating System
Use Case	Reset Nodes
Description	Once the power aware job completes, the HPCS Resource Manager directs the operating systems associated with the latest job to re-configure to default settings.
Trigger	A job has completed running.
Flow of Events	1. HPCS Resource Manager requests re-configuration of power parameters 2. The computing resources may go into a very low power state since there is no active job.
Alternate Paths	1a. The requested re-configuration may fail. 1b. The resources may be marked unusable.
Frequency	For every Power Aware job launched
Input Data	Optional Power Parameters, in case there are no system defaults
Output Data	Power Parameters successfully re-configured
Pre Condition	Operating System able to receive requests for Power Aware Parameters
Post Condition	Power Aware Parameters are reset to default
Power API	Yes. It is likely the same API as when resources are initialized prior to job start.

3.7.3.1 Notes This use case could be combined with 3.7.1.

3.8 Actor: HPCS Operating System
System: HPCS Hardware

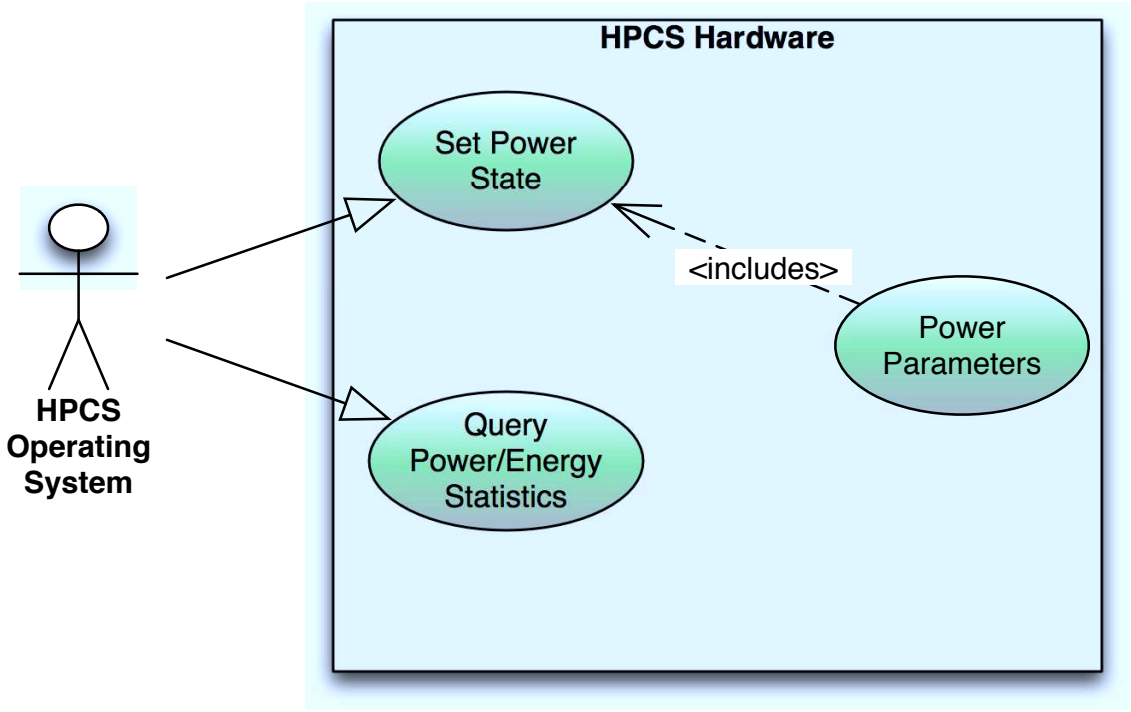

Figure 11. HPCS Operating System ⇒ HPCS Hardware Use Case Diagram

3.8.1 Use Case: Set Power State

Actor	HPCS Operating System
System	HPCS Hardware
Use Case	Set Power State
Description	The HPCS Operating System interfaces with the HPCS Hardware to set a power related state. A generic example of this would be setting the processor P-state which defines both frequency and input voltage parameters under which the processor will operate.
Trigger	1. HPCS Operating System is setting power state on behalf of HPCS Resource Manager. 2. HPCS Operating System is setting power state on behalf of HPCS Application (or library). 3. HPCS Operating System is setting power state independently based on some other criteria.
Flow of Events	1. HPCS Operating System requests power related state change 2. HPCS Hardware changes current power related state 3. HPCS Operating System returns success
Alternate Paths	1a. HPCS Operating System unable or unwilling to request power related state change 1b. HPCS Operating System returns failure 2a. HPCS Hardware fails (or is unable) to change power related state 2b. HPCS Operating System returns failure
Frequency	Prior to job launch Following job completion Throughout job execution Between job allocations (on idle nodes, for example)
Input Data	Hardware parameters that will likely be component-dependent. For processors, these might be frequency, voltage and/or operating system (P, C, S, etc state)
Output Data	Current/updated hardware settting
Pre Condition	Power State prior to request
Post Condition	Requested Power State is achieved
Power API	Yes

3.8.1.1 Notes It is important to note that the available power related parameters will be hardware specific. This use case uses CPU P-state as an example but it is anticipated that a range of power related parameters will be available on future platforms. It is also recognized that the CPU will likely not be the only hardware component that has power related settings exposed to operating system control. The HPCS Operating system will likely play the role of mapping generic requests to more specific hardware parameters. For example, the HPCS Application should not have to know about CPU P-states. The HPCS Application may just request a High, Medium or Low Frequency. Alternatively, the HPCS Application could request a change based on a percentage of the default or current setting. Again, these are CPU specific examples, other components might require drastically different interfaces. These details will be sorted out and tested by implementing reference implementations of the specification.

A note on runtime systems: More and more intelligence is being integrated into a software component referred to as a "runtime system". This conceptual component can be implemented in several domains. It can be a root-level daemon, linked as a library to the application, and/or embedded in the operating system.

The resource manager, application, and operating system sections of this document apply to the pieces that might be implemented in each domain.

3.8.2 Use Case: Query Power/Energy Statistics

Actor	HPCS Operating System
System	HPCS Hardware
Use Case	Query Power/Energy Statistics
Description	The HPCS Operating System interfaces with the HPCS Hardware to query the current power/energy related statistics. This could include the current power/energy state, instantaneous current and or voltage, instantaneous power or accumulated energy (for example).
Trigger	HPCS Operating System is acting on behalf of a query from the HPCS Application HPCS Operating System is acting on behalf of a query from the HPCS Resource Manager HPCS Operating System is acting on behalf of itself if it is abstracting or storing any of the power/energy related statistics.
Flow of Events	1. HPCS Operating system requests a range of hardware specific power/energy statistics from the HPCS hardware 2. HPCS Hardware returns requested statistics
Alternate Paths	2a. HPCS Hardware does not return request statistics 2b. HPCS Operating may repeat request of statistics, return failure or some defined condition
Frequency	The frequency of this interaction could vary greatly depending on where the request is initiated from. In the case of the application (3.10.2) or a user (3.13.1), it may request multiple data points at a sampling rate.
Input Data	List of Power/energy statistics requested
Output Data	Power/Energy statistics requested.
Pre Condition	Components have generated and reported the power/energy related data
Post Condition	HPCS Operating system has current HPCS Hardware power/energy statistics requested.
Power API	Yes

3.8.2.1 Notes We assume that a number of different components could make power/energy statistics available to the operating system. As an example case, the HPCS Operating system interfaces with the CPU to obtain whatever power/energy related statistics are available. The HPCS Operating system may abstract these statistics to return them to higher layers of the software stack to provide them in a more standard and portable way. For example, an energy counter is made available as part of the HPCS Operating System API. If the HPCS Hardware provides this capability the HPCS Operating system may simply query this statistic and return it to the requestor. Alternatively, if the HPCS Hardware only exposes a power or separate current and voltage statistic, the HPCS Operating system might calculate and store this counter in a register, and return it upon request.

A note on runtime systems: More and more intelligence is being integrated into a software component referred to as a "runtime system". This conceptual component can be implemented in several domains. It can be a root-level daemon, linked as a library to the application, and/or embedded in the operating system. The resource manager, application, and operating system sections of this document apply to the pieces that might be implemented in each domain.

3.9 Actor: HPCS Monitor and Control System: HPCS Hardware

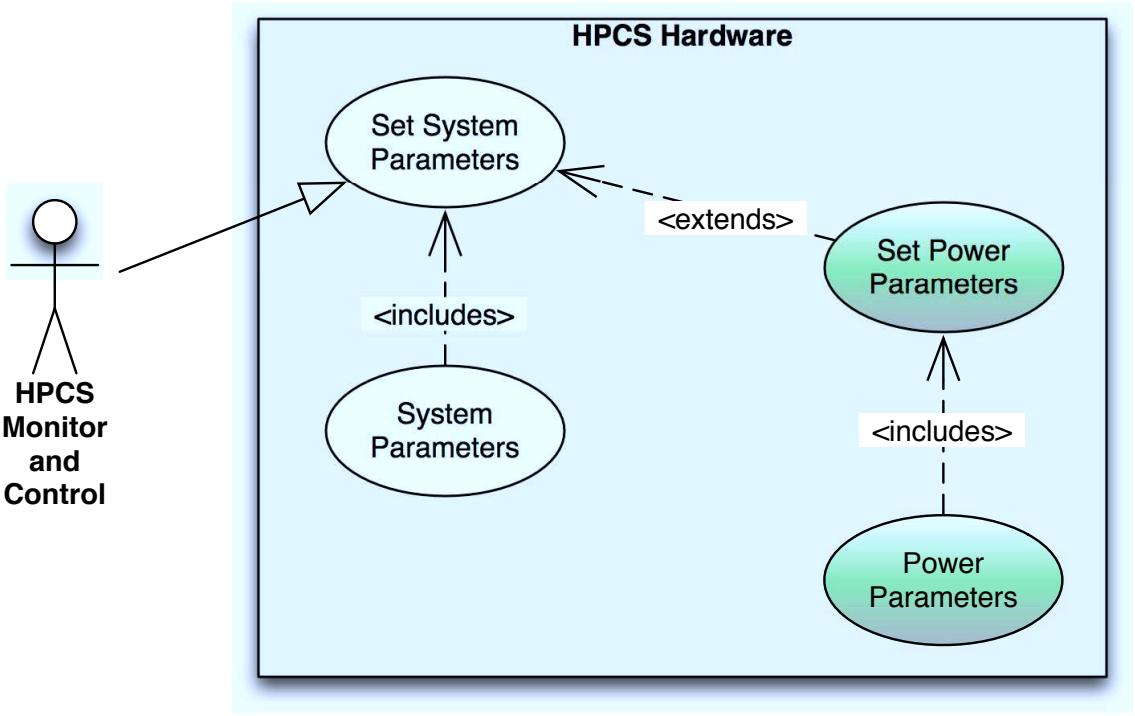

Figure 12. HPCS Monitor and Control ⇒ HPCS Hardware Use Case Diagram

3.9.1 Use Case: Set Power Parameters

Actor	HPCS Monitor and Control
System	HPCS Hardware
Use Case	Set Power Parameters
Description	The HPCS Monitor and Control System is responsible for setting power parameters on the HPCS Hardware. This could take the form of strict power limits on system (power caps), racks, cages, or nodes.
Trigger	The HPCS Admin requests a power parameter change. (see Section 3.12.1)
Flow of Events	1. HPCS Monitor and Control requests certain power parameters be set on the HPCS Hardware. 2. The parameters are successfully set on the HPCS Hardware. 3. Return success as well as current power parameter settings.
Alternative Paths	3a. Return Failure 3b. Resubmit parameter change request 3c. Return success or failure
Frequency	This could happen as frequently as necessary to manage system power and energy consumption.
Input Data	Power and energy parameters and values necessary to meet resource management targets.
Output Data	A hierarchical view of power parameters by system, racks, cages, or nodes.
Pre Condition	Power parameters are at state A.
Post Condition	Power parameters are updated to state B.
Power API	Yes

3.9.1.1 Notes A common example of what this Actor/System interface would be used for is to set platform hardware power caps. These power caps would likely be requested by the HPCS Administrator but could alternatively, or additionally, be requested by the HPCS Manager. Depending on system architectures the HPCS Monitor and Control system may or may not directly access platform components such as the CPU.

Platform-level power capping could be implemented via software, such as by the resource manager performing job-level capping. That use case is described in Section 3.7. Alternatively, or additionally, some set of components could be configured separately via this use case. With this use case, it would be possible to have power capping without suppport from the resource manager. This use case also applies when there are environmental or power feed issues for some set of components.

Setting a hard power threshold may not be practical. Power may only be measured at some frequency. Usage exceeding the threshold may happen between sampling periods. The API may require a duration or other concept to allow for variability in the system.

3.10 Actor: HPC Application
System: HPCS Operating System

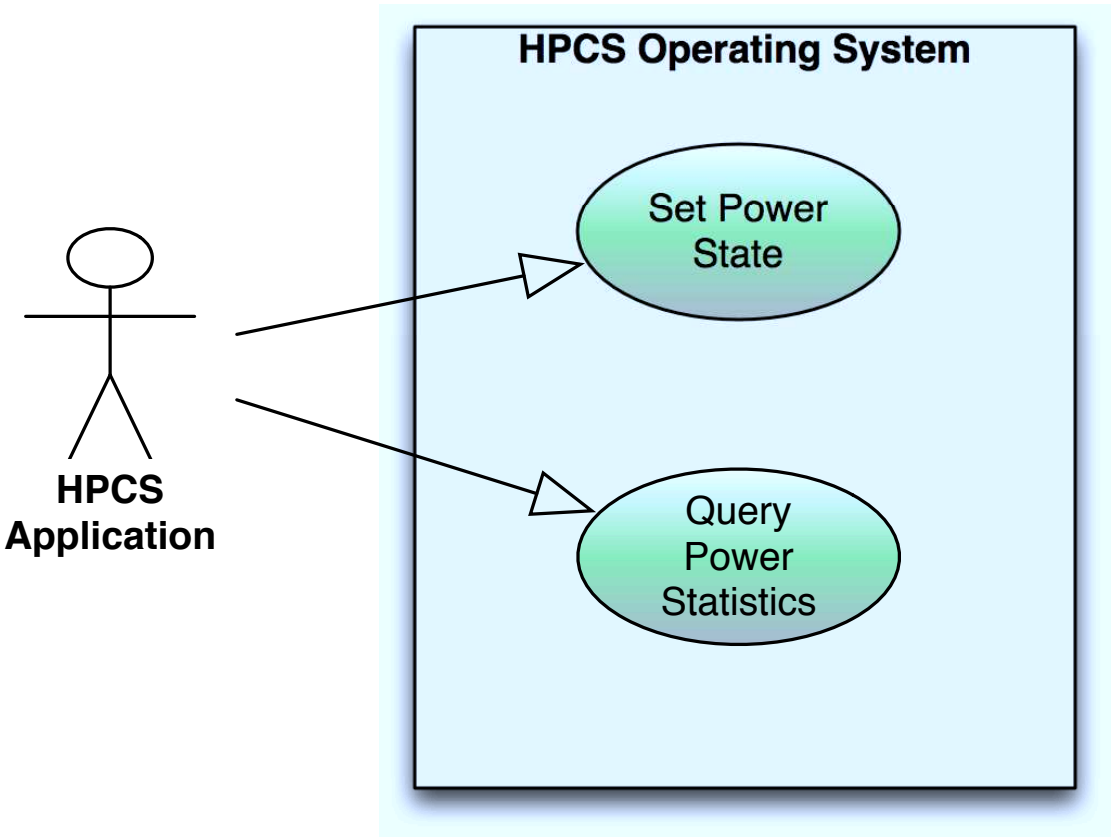

Figure 13. HPCS Application ⇒ HPCS Operating System Use Case Diagram

3.10.1 Use Case: Set Power State

Actor	HPCS Application
System	HPCS Operating System
Use Case	Set Power State
Description	While an application is running, one, some, or all of the processes may choose to actively manage its power or energy consumption on its node. For example, it may request to lower power state on the CPU while writing a checkpoint to disk or going into an I/O intensive phase.
Trigger	The application reaches a point in the code where it knows that the power state is either particularly important or not important.
Flow of Events	1. As a precursor to setting a new power state, the application requests information on available power states and the current power state of its components (see Query Power Statistics use case 3.10.2). 2. The operating system returns requested power/energy state and statistics. 3. If the current power state is not optimum, the application decides how much to raise or lower the power/energy state of one or more components. 4. The application requests that the power state be changed on one or more components. 5. The operating system returns success along with state information.
Alternative Paths	5a. The operating system returns failure along with state information. 5b. The Application can decide to resubmit request immediately or wait until a later point in execution.
Frequency	Could be measured in seconds, more likely in minutes
Input Data	In step 4, the application requests a power state change.
Output Data	In response to step 2, the available power states are provided in some generic, system portable fashion.
Pre Condition	Application is executing at power/energy configuration A
Post Condition	Application is executing at power/energy configuration B
Power API	yes

3.10.1.1 Notes The application will likely not have detailed knowledge of hardware specific power and energy settings. It is more likely that the application will be requesting an abstract notion of the power or energy statistics and act based on this information. The HPCS Application might request changes based on a percentage of the current state or in a stepwise fashion, for example high, medium or low. The application will also have limited information on the latency of this operation. The API could provide the latency in the response.

A note on runtime systems: More and more intelligence is being integrated into a software component referred to as a "runtime system". This conceptual component can be implemented in several domains. It can be a root-level daemon, linked as a library to the application, and/or embedded in the operating system. The resource manager, application, and operating system sections of this document apply to the pieces that might be implemented in each domain.

3.10.2 Use Case: Query Power Statistics

Actor	HPCS Application
System	HPCS OS
Use Case	Query Power Statistics
Description	While probably not necessary to use on every run, the application may have a "debug" flag that enables collection of power information. The HPCS Application queries power statistics to aid in making decisions about changing power states (see Set Power State 3.10.1)
Trigger	"Debug" flag or other notification tells the application to collect power consumption data. The application hits a pre-defined phase and requests power statistics before possibly setting a new power state.
Flow of Events	1. Application requests power statistics. The request could be for a single set of data points or the application could specify a collection interval and statistics that are requested. 2. Operating system returns statistics requested (single instance or on requested interval). 3. If interval query Application might direct collection to stop (alternatively might list number of samples up front)
Alternative Paths	2a. Operating system unable to satisfy single or one or more interval requests. It then exits.
Frequency	At discretion of the HPCS Application
Input Data	Statistics desired, collection interval (if not single request), number of samples etc.
Output Data	Statistics including time stamp, source (which component) etc.
Pre Condition	Job is running
Post Condition	Job is running, has updated Power/Energy statistics requested.
Power API	Yes

3.10.2.1 Notes It is typically very desirable that to take advantage of architecture specific features the application is not required to be modified. This interface along with Set Power State 3.10.1) might be implemented in a Library that is made available as a standard. This use case might be employed by the HPCS Application to simply keep power and energy statistics for some reason. This use case might also be leveraged to make decisions on a very dynamic basis in which the HPCS Application actively manages its power/energy use. We do not envision this operation to be coordinated across nodes.

A note on runtime systems: More and more intelligence is being integrated into a software component referred to as a "runtime system". This conceptual component can be implemented in several domains. It can be a root-level daemon, linked as a library to the application, and/or embedded in the operating system. The resource manager, application, and operating system sections of this document apply to the pieces that might be implemented in each domain.

3.11 Actor: HPCS Accounting
System: HPCS Monitor and Control

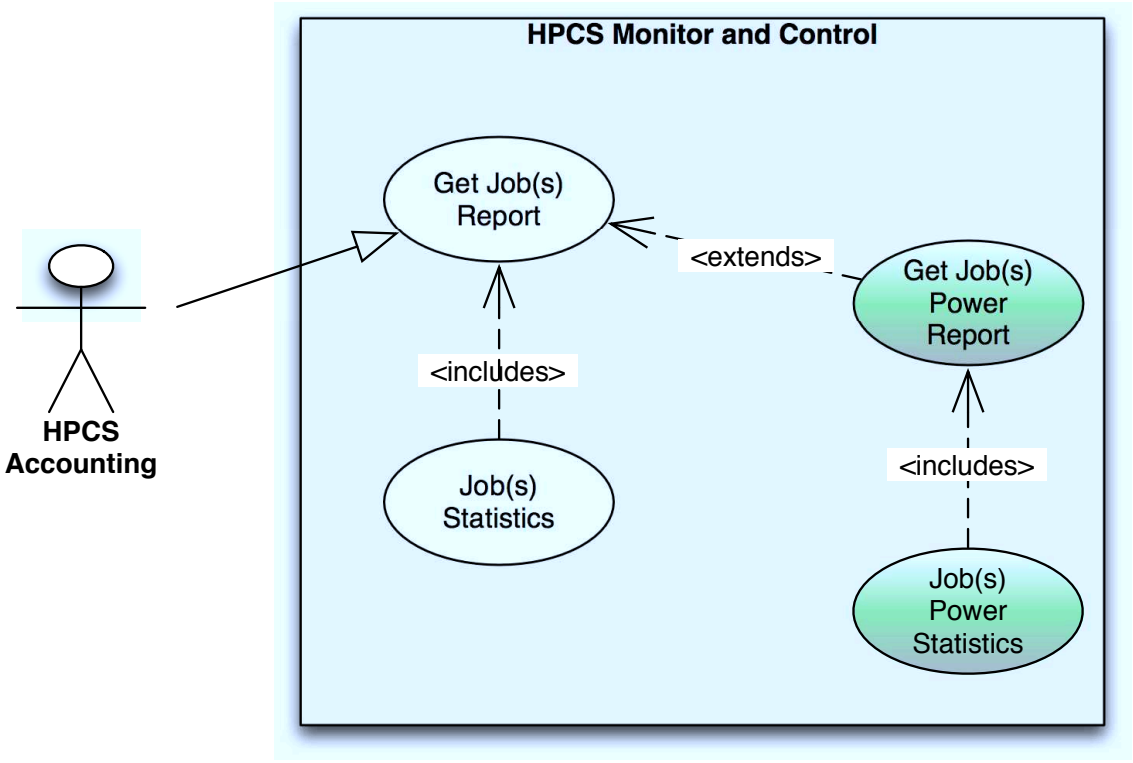

Figure 14. HPCS Accounting ⇒ HPCS Monitor and Control

3.11.1 Use Case: Get Job(s) Power Report

Actor	HPCS Accounting
System	HPCS Monitor and Control
Use Case	Get Job(s) Power Report
Description	The HPCS Accounting software interacts with HPCS Monitor and Control system in order to produce reports. This use case extends the existing Get Job Report use case to include power metrics.
Trigger	HPSS Accounting requests Job(s) Power Report.
Flow of Events	1. The HPCS Accounting software sends Job Report request to the HPCS Monitor and Control system. 2. The results of the successful query are returned to HPCS Accounting.
Alternate Paths	2a. Query is re-issued if it failed in step 2. 2b. Results of query are sent to HPCS Accounting.
Frequency	On scheduled basis, once per week for example, or impromptu request (likely not more than a few times a day)
Input Data	All (full report) LIst of specific job id's Time range Likely other useful report specifications.
Output Data	Full Power related job report Power related job report for specific job id's Power related job report for specified time range. Power related job report based on requested parameters.
Pre Condition	Jobs have run.
Post Condition	Report generated
Power API	Yes (possibly extends existing API)

3.11.1.1 Notes A differentiating factor of this use case is that the HPCS Accounting system will have access to information about any job that has been executed on the system, whereas a specific HPCS User will only have access to a job that was executed under their user id. The report generated might contain a range of data but will certainly include metrics like energy used over the duration of the application execution, min, max and average power and power/energy parameters used during application execution.

3.12 Actor: HPCS Admin
System: HPCS Monitoring and Control

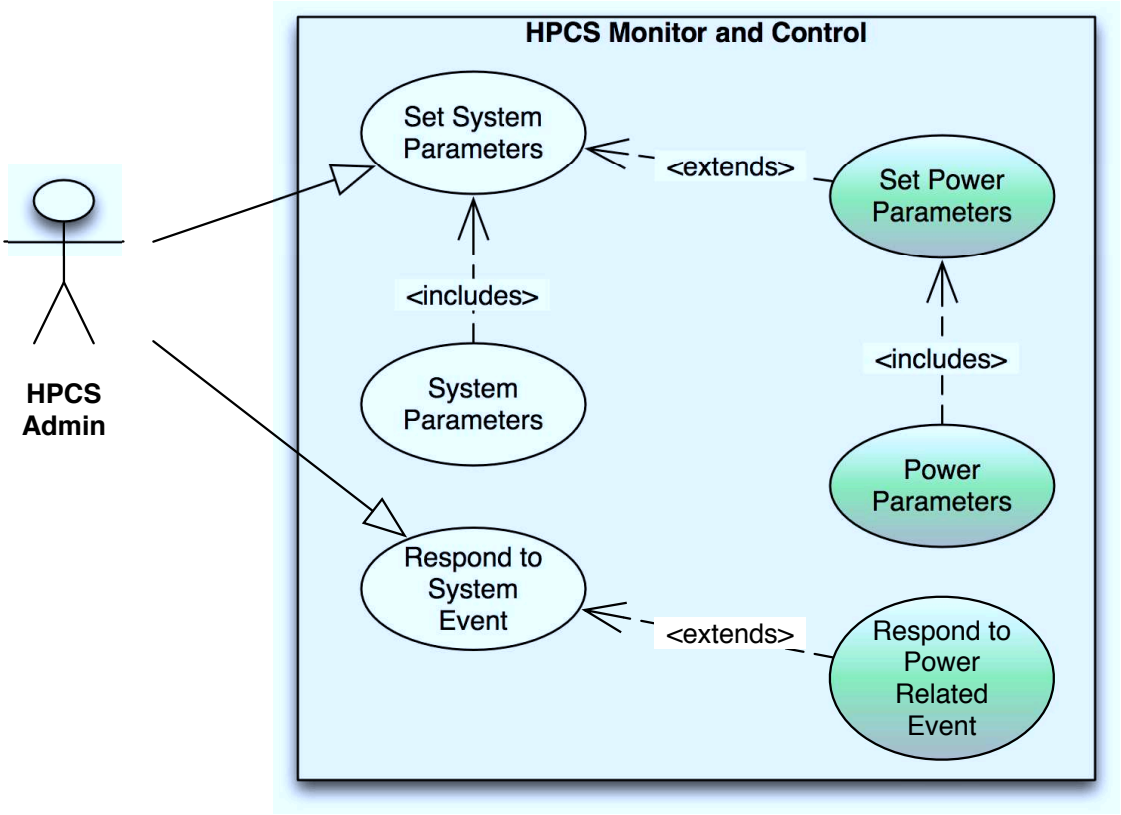

Figure 15. HPCS Admin ⇒ HPCS Monitoring and Control Use Case Diagram

3.12.1 Use Case: Set Power Parameters

Actor	HPCS Admin
System	HPCS Monitoring and Control
Use Case	Set Power Parameters
Description	Based on a new policy, set hardware controls to enforce maximum or minimum (power capping for example). Note that the new maximum can be either an increase or decrease from the prior maximum.
Trigger	A new power policy is to be implemented. The request is typically originated by the HPCS Manager.
Flow of Events	1. HPCS Admin invokes command to set new power maximum (or minimum). 2. Use case Set Power Parameters is invoked. See Section 3.9.1 3. HPCS Monitor and Control system returns success (along with the new parameter settings).
Alternative Paths	3a. HPCS Monitor and Control returns failure (along with the current parameter settings) and possibly a reason why the request could not be honored
Frequency	Likely not more than once or twice a day, might be intrusive or require the system to be quiesced
Input Data	Maximum system usage in Megawatts (if applied to platform) Specific maximums or minimums if applied to other platform granularities.
Output Data	Current parameter settings in maximum Megawatts (if applied to platform) Other parameter settings as appropriate based on granularity of request in appropriate units
Pre Condition	None
Post Condition	Platform (rack, cage or node) is operating under new power parameters.
Power API	yes

3.12.1.1 Notes When a new power policy is requested to be implemented (Trigger) this direction and information likely comes from the HPCS Manager. The HPCS Manager and the HPCS Admin might be one in the same person operating under different roles. Changing power parameters might cause jobs to be killed and others started depending on other system wide settings and policies. The common need for this use case at this time is setting power caps (and or minimums) for the platform at the granularity of the platform, rack, cage or node level.

3.12.2 Use Case: Respond to Power Related Event

Actor	HPCS Admin
System	HPCS Monitoring and Control
Use Case	Respond to Power Related Event
Description	Based on an asynchronous event like an out of bounds condition. Will usually result in querying the monitoring and control systems for additional detail.
Trigger	An environmental, out of bounds or hardware alarm has been raised.
Flow of Events	1. The admin receives and reviews the notice of a out of range power condition. 2. The admin queries the HPCS Monitoring and Control system for warnings or errors in the log data. 3. The admin queries the HPCS Monitoring and Control system for current power metrics. 4. The admin collects and synthesizes the important information.
Alternative Paths	TBD
Frequency	Daily, Weekly, Monthly depending on conditions and platform usage and stability.
Input Data	Email alert, page, or audible alarm, system logs.
Output Data	Power-related errors messages and metrics, possibly response suggestions
Pre Condition	An alaram has been raised.
Post Condition	Sufficient data is collected upon which to take action.
Power API	yes

3.13 Actor: HPC User
System: HPCS Monitoring and Control

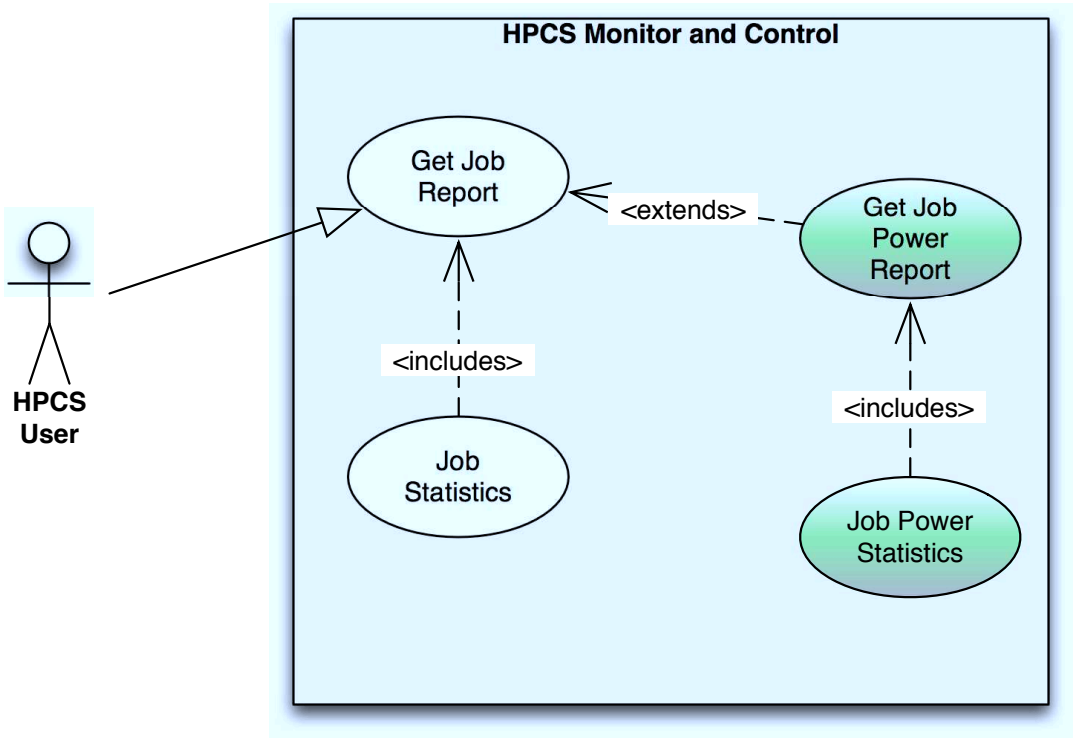

Figure 16. HPCS User ⇒ HPCS Monitoring and Control Use Case Diagram

3.13.1 Use Case: Get Job Power Report

Actor	HPCS User
System	HPCS Monitoring and Control (M&C)
Use Case	Get Job Power Report
Description	Whether it is an active or completed job, the user can query the monitoring system for job power-related statistics. The returned data could be in tabular/text (e.g. csv) or graphical form.
Trigger	The end user is interested in the power or energy statistics of their job. The concern could stem from various sources. Each user may have a maximum energy allotment for example. The user may be a library developer and is attempting to provide algorithmic solutions to application developers that minimize energy. The HPC center may require that jobs declare an upper bound on their peak power usage and this run is to identify a safe upper bound for future runs.
Flow of Events	1. User either invokes a GUI or enters a command to interact with the HPCS M&C. 2. User specifies job ID of interest, which is determined to be either running, completed, or invalid. An invalid result displays an error message and this step is repeated. 3. With actionable input, the HPCS M&C determines 1) when the job started, 2) ended (in the case of a completed job) and 3) the hardware components used exclusively by the job. 4. User provides more detail on desired information: a. time range b. sampling interval and/or bins of time c. whether aggregated consumption information is desired or for specific components, such as memory, CPU, NIC is requested 5. HPCS M&C processes the request and for active jobs, determines if data collectors need to be started. 6. Data is returned to user in requested format.
Alternative Paths	2a The input may also indicate use case termination, and if it does, the M&C stops any dynamic collection that had been started.
Frequency	Hundreds of times per day
Input Data	Job Identifier Output format desired time range of interest sampling interval or bins of time Aggregated consumption or identification of components to study
Output Data	energy in what units? The data may be arrays of components with minimum, maximum, average values.
Pre Condition	None
Post Condition	User has requested information
Power API	yes

3.13.1.1 Notes In this use case in contrast with the use case for the HPCS Accounting system, the User only has access to their job information. Shared hardware component usage is not accounted for in this

use case. The volume and longevity of the power data is an important implementation consideration. We do not foresee a common API to set these values.

3.14 Actor: HPCS User
System: HPCS Resource Manager

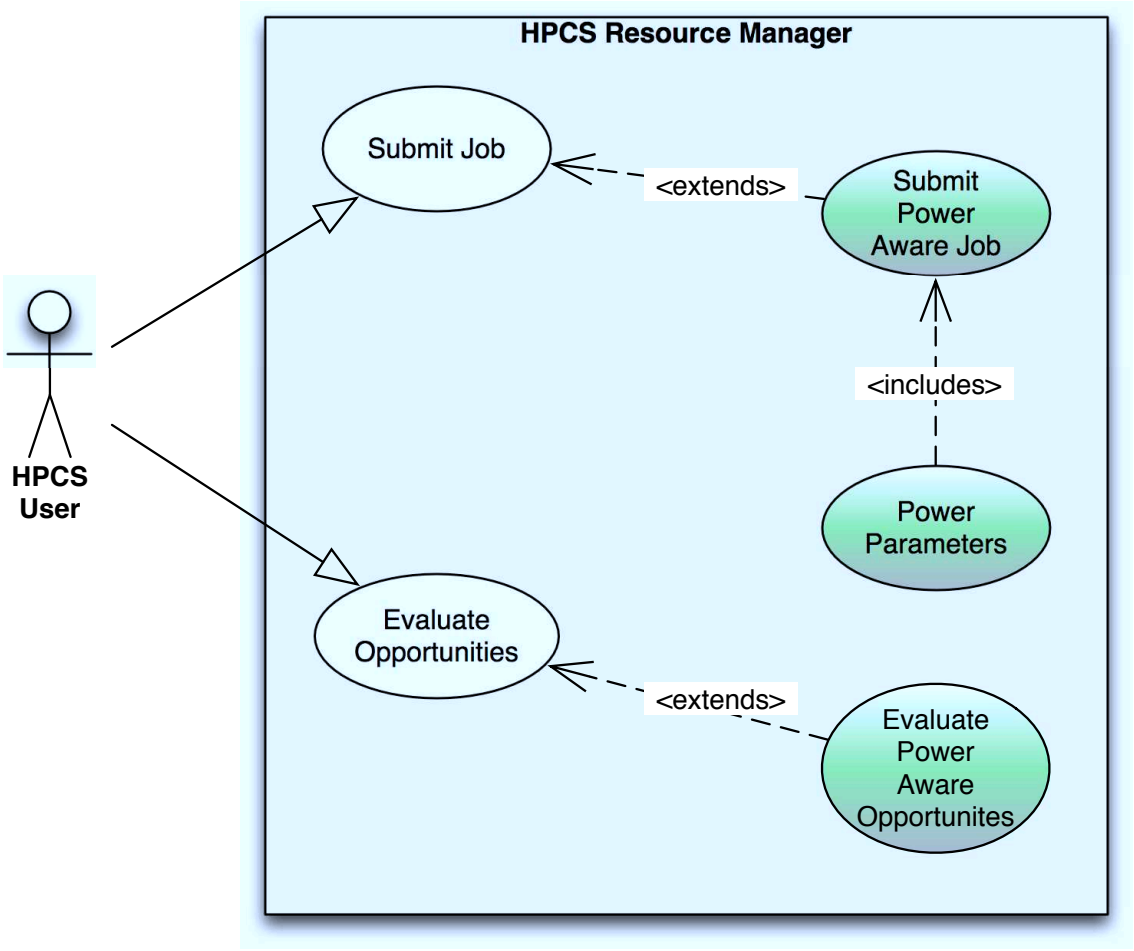

Figure 17. HPCS User ⇒ HPCS Resource Manager Use Case Diagram

3.14.1 Use Case: Submit Power Aware Job

Actor	HPCS User
System	HPCS Resource Manager
Use Case	Submit Power Aware Job
Description	The HPCS User submits a power aware job to the HPCS Resource Manager. For example, user wants to submit their job in a more energy efficient way to improve performance or get a better turn-around time through the system which could depend heavily on how quickly the job is scheduled (based on the fused metric).
Trigger	HPCS User decides to submit power aware job
Flow of Events	1. HPCS User submits a power aware job to the Resource Manager (job which includes power parameter information) 2. HPCS Resource Manager returns success of the submission to HPCS User
Alternative Paths	3a. Return Failure 3b. HPCS User may resubmit the job
Frequency	Many times a day depending on the use of the platform, capacity vs. capability for example
Input Data	Power aware job parameters and constraints
Output Data	Submission success code possibly with estimate of when job will execute based on submission parameters
Pre Condition	HPCS User has a power aware job ready to submit.
Post Condition	HPCS User job successfully submitted
Power API	Yes

3.14.1.1 Notes The HPCS Resource Manager will start the job based on the policies set through the interaction with the HPCS Manager. This use case only accounts for job submission.

3.14.2 Use Case: Evaluate Power Aware Opportunities

Actor	HPCS User
System	HPCS Resource Manager
Use Case	Evaluate Power Aware Opportunities
Description	Obtain a high level, possibly followed by a semi-detailed level of information regarding the power usage of the system. With power evolving into a managed resource (in addition to nodes) it may be possible to exploit available cycles.
Trigger	User desires to submit a job and is flexible about adapting to available power resources.
Flow of Events	1. User interacts with the HPCS to request power resource information. 2. System responds with summary information. 3. User may desire to probe for additional information that would allow a job to run sooner rather than later.
Alternative Paths	None
Frequency	Perhaps a dozen times per day.
Input Data	User request, perhaps with options for lower level detail
Output Data	Summary and job-level detail of power consumption
Pre Condition	None
Post Condition	Increased understanding of power status of the HPCS.
Power API	Yes

3.14.2.1 Notes We see this use case as equivalent to the "show backfill" capability currently available on some resource management systems. The "user" is quite generic in this case. In addition to the end-user, it could be an administrator simply assessing the available power capacity.

A Brief Power Aware HPC Scenarios

A.1 Dynamic Frequency Scaling

The local forecast shows that a facility's large-scale photovoltaic (PV) array's ability to produce power is compromised due to an approaching weather system set to arrive within the next hour. The facility manager would like to scale back power being used by the HPC system so that the overall campus or facility does not reach its peak power level. We explain this scenario in more detail in Section B.1

The Facility Manager communicates the facility demands or requirements using specific power parameters to the HPCS Manager. The HPCS Manager will set power-related scheduling policy using the HPCS Scheduler.

A.2 Demand Response Signals from Utility Providers

Many large-scale utility companies offer incentives for businesses, institutions, and even households to reduce their power demand for a short time period in order to help reduce the overall demand on the grid. Given the extremely large amount of power supporting an HPCS facility, the HPCS manager may be asked to scale back the power demand on the system during these time periods, e.g. using ideas similar to A.1, powering down idle nodes, etc.

As an example, Xcel Energy (via their contractor EnerNOC) pays NREL to reduce its campus load for approximately one hour per year. This process is currently initiated by Xcel via a phone call to NREL's Xcel point-of-contact. In the future, this signal will be sent automatically through software such as OpenADR (developed by LBNL) and Voltron (developed by PNNL). The details of the conversation between a facility and its utility provider are communicated to the Facility Manager. The Facility Manager communicates the facility demands or requirements using specific power parameters to the HPCS Manager. The HPCS Manager will set power-related scheduling policy using the HPCS Scheduler.

A.3 Change Energy Recovery

An HPC data center generates useable heat that may offset heating loads in other parts of the facility or campus. If the Facility Manager is told (e.g. using a predictive model) that the weather is going to be colder than expected, the HPCS facility can be operated at its maximum capacity to produce extra waste heat. This may involve a reduction in water flow rates through a liquid-cooled system.

At a facility such as NREL's new Energy Systems Integration Facility (ESIF) waste heat is being used on the NREL campus to preheat outside air for ventilation, heat return/mixed air, and melt snow on certain walkways.

A.4 Micro-grid Demand Management

Micro-grid environments are becoming increasingly common. There would be no choice but to control the load of the HPCS system in order to meet the demand of other systems within a micro-gridded ecosystem. In particular, a sophisticated software layer would likely act as the Facility and HPCS Managers in order to regulate the generation, distribution, and storage of the micro-grid's energy in concert with the HPCS demands.

A.5 Shifting Power Source

In the case of a power grid failure, a facility may switch (if available) to on-side diesel generators in order to satisfy their power demands. As a result, the Facility and HPCS Managers will need to communicate and implement their power requirements as they manage toward the generator's power cap.

A.6 Mission or Time Critical Computing Need

There is a mission or time critical computing need which requires a large workload increase. The scheduling of this type of job should involve the Facility Manager because the facility may be required to provide colder water for some period of time.

A.7 Computation with an Energy Budget

Complete a calculation with an *a priori* determination of acceptable runtime performance versus power consumption. The HPCS user only interacts with the runtime system after the user's job has been allocated resources by the scheduler. The HPCS user depends on the runtime system to initialize the energy consumption environment based on the user's parameters specified at job submission. This initialization is done before or as part of application launch. Once begun, the runtime system can receive commands that direct the runtime system to reduce job energy consumption due to a system-wide power reduction or threshold reached. At a later time, the runtime may be directed to resume to the original job parameter specifications. At job termination, the user reviews the energy consumption provided in the final job output.

A.8 Real-Time Node Energy Management

Once an application is executing, it may choose to obtain information on energy consumption at various points in the code. This would come from the operating system on each node. The application would do the aggregation and analysis based on its pre-determined areas of interest. Additionally, the application may choose, for example, to request a lower power state while writing a checkpoint to disk. When the I/O is complete, the application would request that the previous power state be restored.

A.9 User-accessible Power Analysis Tool

As an alternative to instrumenting individual codes (or libraries), the user may wish to employ a power monitoring tool. This tool would likely be very similar to existing performance analysis utilities. Rather than analyzing cache misses or MPI waits, it would provide information on the power usage of a running application. As suggested by one reviewer of this document, the tool could provide a "frequency sensitivity" metric. This metric would indicate if this code, or section of code, could take advantage of a higher or lower processor frequency setting.

A.10 Predictable Applications

Some HPC facilities may have a relatively small set of applications that run frequently (e.g. prediction of tomorrow's weather). As applications are run, the resource manager could collect the power consumption information. If there is a unique application identifier, the consumption information could be accumulated in a data base. When the same application is submitted again, the resource manager could query the data base for power requirements. This scenario is in contrast to requiring the user to determine the power requirements for a job.

B Extended Power Aware HPC Scenarios

B.1 Campus/Facility Power and Energy Management

B.1.1 Scenario

A typical utility company will charge commercial or industrial customers for their energy usage in addition to a *demand* charge. As an example, Xcel Energy's demand charge is based on the maximum average power used during any 15-minute interval in the billing cycle. This demand charge is calculated at a significantly higher rate than the usual energy rate. Therefore, a simple strategy for reducing a facility's utility bill is to reduce their peak power usage. HPCS facilities such as a national lab, academic institution, commercial data center, etc., are in a unique position in that they might be able to mitigate the cost associated with the peak charges by managing the power in their HPC data centers using demand-response models. In this section, we will describe a demand-response scenario and discuss how it fits in to the proposed Power API.

Suppose that an HPC facility's power monitoring software determines that the peak power is likely to be unusually high in the next hour. At a campus similar to the National Renewable Energy Lab in Golden, CO, this could be due to a facility's large-scale photovoltaic array being compromised by cloud cover. The following set of actor/system pairs are impacted in this scenario.

A The HPCS Admin (actor) and HPCS Monitor and Control (system) pair (see 3.12) is used to provide input into the predictive model of peak power usage. An HPCS system is traditionally one of the largest energy consumers/systems on a campus and, thus, necessitates consistent monitoring. Results of this monitoring and analysis are used as inputs into any predictive model involving campus power consumption. In is worth noting that the HPCS Admin in this scenario would likely be software and not an actual person.

B The Facility Manager will initiate a conversation with the HPCS Manager (3.4) in order to discuss if adjustments to the HPC system are possible. Adjustments in this case may include lowering p-states, suspending the submission of new jobs, etc. Although this interaction is not technically a part of this Power API, we feel that this conversation is sufficiently important to be included in this document.

C If it is determined that an action on the HPC system is appropriate to reduce power, the HPCS Manager will interact with the HPCS Resource Manager in order to initiate this change (3.5).

D The implementation details of this change (e.g., reducing p-states across processors) are communicated through the system via the HPCS Resource Manager to the HPCS Operating System (3.7) followed by the HPCS Operating System translating the instructions to the HPCS Hardware (3.8).

E The HPCS Admin will follow up with the HPCS Monitor and Control System (3.12) in order to verify that the system is working as expected. In this case, the HPCS Admin will likely be a person.

B.2 Increase Application Efficiency

B.2.1 Scenario

Application Efficiency will be determined by an as yet undefined metric (fused metric) that will likely include performance, energy, priority of job, amortized node expense and time-of-day (to list a few) as variable and/or weighted parts of eventual fused metric (goodness value). In concept, this is similar to Energy Delay Product (EDP) but more inclusive and targeted to a particular sites needs. This scenario, for the purposes of simplification, will focus on performance (wall clock execution time) and energy (combined total energy used by a single application on all nodes used by the application for the duration of application execution).

A Execute a large-scale production scientific computing application (application) using the default system environment and parameters. This first run is a productive run that produces productive results. This

run also establishes the baseline energy and performance characteristics of this application (Requirement 1) (Note: other factors might affect the application such as scale and input problem requiring separate or additional analysis).

B Analyze the energy and performance data to determine what available tuning parameters might be applied to follow on executions (Requirement 2).

C Execute SAME application (keep as many factors, number of nodes, problem type etc. the same as possible) with tuning parameters applied (Requirement 3). The energy and performance results from this run will be judged relative to the baseline execution (Requirement 1,2). Note, this is also a productive run producing useful production results.

D Analyze (Reference B), (Requirement 2).

E Additional component level analysis to determine if additional tuning can be productively applied (Requirement 4).

F Additional executions of the SAME application until range of productive tuning parameters are established (Requirement 1,2,3,4).

B.2.2 Notes

This scenario focuses on applying tuning parameters to hardware power management capabilities, e.g. affecting the frequency and subsequent energy use of the CPU. This could be accomplished in a number of ways including setting P, C or S states, or any other architectural mechanism exposed for this purpose. We value any opportunity to affect energy used by ANY component if it can be leveraged to increase the energy efficiency of our applications, and/or to operate the system within its externally allocated and variable power budget. The cycle outlined is a general high-level scenario. The process is repeated using production applications so all time consumed for analysis is the result of productive runs. The output of the analysis is an understanding of the effect on performance and energy of tuning parameters that can be applied to this application at this scale for this problem. The knowledge gained by this analysis can be used to simply run the application more efficiently (using the proposed fused metric for example) or to implement intelligent power capping of the overall platform (described in a separate scenario).

B.2.3 Requirements for Increasing Application Efficiency

1. To obtain an energy profile, the amount of energy used by a single application on all nodes used by the application, the minimum requirement is a node level measurement capability. Since an application will be executed on a large number of nodes and there is no expectation that these nodes will coincide perfectly with a set of cabinets a node level measurement capability, rather than cabinet level, is required. The frequency of data sampling per node should be greater than or equal to, one sample per second to obtain enough fidelity for analysis and comparison with subsequent runs. It is expected that at the node level the data samples will be DC measurements, discrete current and voltage values.

2. This analysis step implies that the measurement data be made available for analysis, at a minimum, after application execution. Also implied is a transport mechanism to coalesce the data to a single location. The transfer of data from the points of measurement must scalable and efficient to be of utility. Further, this also implies tools are available for analysis such as generating a energy total for all nodes involved in the application and possibly visualization capabilities to analyze characteristics of the energy profile of the application over the duration of the run. These tools will help determine the most productive tuning parameters to apply to subsequent executions. An out-of-band tightly integrated Reliability Availability and Serviceability (RAS) subsystem could be leveraged to accommodate many of these capabilities.

3. To affect the energy used, tuning parameters must be exposed, for example, to the OS, run-time, launch mechanism, scheduler or application library. For example, before application execution all cores of all nodes that will host the application are set to a lower frequency state. After execution, the nodes are reset to default values. This implies an ability to control CPU frequency, for example. This is more of a static approach to CPU frequency tuning.

4. Initial tuning parameters may be applied statically. Multiple static tuning configurations may be applied and analyzed based solely on composite or node level analysis. It may be determined that to achieve additional application energy efficiency, or to achieve any relative to baseline, dynamic tuning methods must be applied. Component level measurement is required to determine where energy is being used within a node. For example, intense compute, communication or IO phases can be observed with component level measurement. Dynamic tuning can be applied to allow the CPU to run at very high frequency during computationally intensive phases. During heavy communication or IO phases the CPU can be set at lower energy saving frequencies. Other components can be tuned if the capability is available and exposed.

B.3 Power Capping

B.3.1 Scenario

Power Capping, minimally defined as the ability to prescribe the instantaneous [1] power draw, energy use (over time) or power/energy fluctuation (including rate of change and magnitude). This scenario will suggest that two methods of Power Capping should be applied in conjunction to maximize the use of the underlying resource while protecting against accidental violations of Power Cap parameters. This scenario will assume that the facility manager has defined, for example, the sites power, energy and cooling parameters for the period of time covered in this scenario and the Platform manager has defined any other parameters that will be used to define Power Capping levels such as other local policy considerations used in the fused metric defined in Scenario B.2. The two approaches are: 1) hardware Power Cap – defined by setting a physical limit to the amount of instantaneous power that the platform, cabinet, node or component is limited to, or rate of change limitation, 2) Power aware scheduling – defined as intelligently scheduling jobs to maintain a mix of power consumption (or energy use) that complies with site policies.

A System Administrators set platform Power Caps as directed by Facility and Platform management (Requirement 1).

B Users schedule applications (over a period of time) that have been previously analyzed (as in Scenario B.2)(Requirement 2).

C Scheduler launches applications with tuning parameters necessary to keep overall platform within Power Cap parameters (Requirement 3).

D Scheduler does not adequately launch application mix to maintain power/energy use within Power Cap parameters (Requirement 4).

B.3.2 Notes

This does not account for a dynamically changing Power Cap, jobs launched based on what caps were at the time of launch.

B.3.3 Requirements for Power Capping

1. This activity requires an integrated ability to configure the platform as a whole, at the cabinet, node and or component level to gate the power and or energy consumption at a hardware level. Efficiently

[1] The term instantaneous is being used loosely here. The implementation may be over some number of samples, for example.

configuring the platform to accomplish this implies that this is an activity that can be accomplished while the platform is operational (it will not be practical to only have the option of accomplishing this configuration at boot time for example). As described in Requirement 2 - Scenario B.2, a RAS system could be leveraged to efficiently accomplish this activity.

2. Implies there is a way for either the user to specify the range of power/energy tuning parameters and associated profile information or this information is available by some other means to the scheduler.

3. In addition to the requirements described in Scenario B.2, which enable individual applications to be analyzed from a power and performance perspective, this activity requires an ability for a scheduler to use the specific power and energy characteristics and tuning parameters as part of the fused metric calculation mentioned in Scenario B.2 to determine how (what tuning parameters) and when (when this application given the known power/energy profile) each application can be scheduled to run to most efficiently use the resource while remaining within the prescribed Power Cap parameters.

4. This implies that the power parameters configured in step A act as a fail-safe preventing the platform, cabinet, node or component from violating the Power Cap parameters.

References

[1] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[2] Ivar Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[3] Geri Schneider and Jason Winters. *Apply Use Cases: A Practical Guide, Second Edition*. Addison-Wesley, 2001.

DISTRIBUTION:

1	MS 1319	James A. Ang, 1422
1	MS 1319	Ron B. Brightwell, 1423
1	MS 0899	Reports Management sanddocs@sandia.gov, 5936
1	MS 0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories