Partial Acquisition

Prashant Jain and Michael Kircher

{Prashant.Jain,Michael.Kircher}@mchp.siemens.de

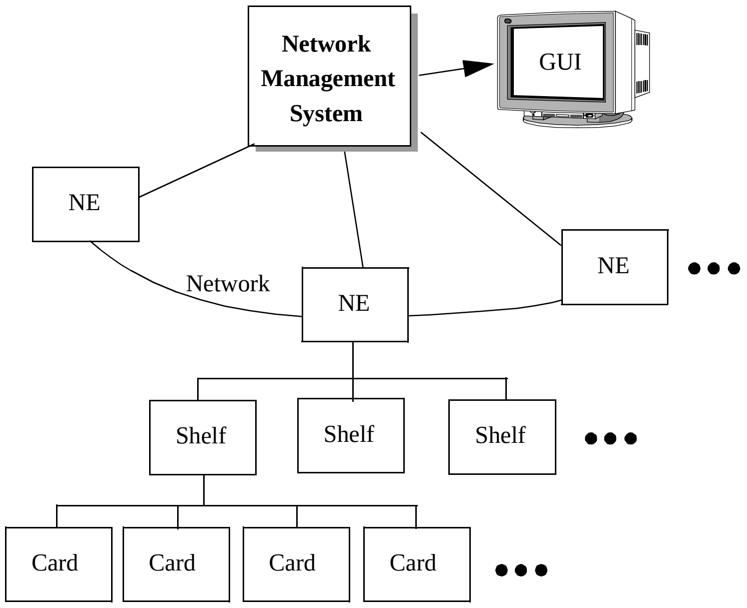
Siemens AG, Corporate Technology

Munich, Germany

Partial Acquisition

The Partial Acquisition pattern describes how to optimize resource management by breaking up acquisition of a resource into multiple stages. Each stage acquires part of the resource dependent upon system constraints such as available memory as well as availability of other resources.

Example Consider a network management system (NMS) that is responsible for managing several network elements (NEs). These NEs are typically represented in a topology tree. A topology tree provides a virtual hierarchical representation of the key elements of the network infrastructure. The NMS allows a user to view such a tree as well as get details about one or more NEs. Depending upon the type of the NE, its details may correspond to a large amount of data. For example, the details of a complex NE may include information about its state as well as the state of its components.



The topology tree is typically constructed at application start-up or when the application is restarting and recovering from a failure. In the first case, the details of all the NEs along with their components and sub-components are usually fetched from the physical NEs. In the latter case, this information can be obtained from a persistent store as well as from the physical NEs. However, in either case obtaining all this information can have a big impact on the time it takes for the application to startup or recover. This is because completely creating or recovering an NE would require creating or recovering all its components. In addition, since each component can in turn be comprised of additional sub-components, creating or recovering a component would in turn require creating or recovering all its sub-components. Therefore, the size of the resulting hierarchical topology tree as well as the time it takes to create or recover all its elements can be hard to predict.

Context Systems that need to acquire resources efficiently. The resources are characterized by either large or unknown size.

Problem Highly robust and scalable systems must acquire resources efficiently. A resource can include local as well as distributed objects and services. Eager acquisition [Kirc02] of resources can be essential to satisfy resource availability and accessibility constraints. However, if these systems were to acquire all resources up front, a lot of overhead would be incurred and a lot of resources would be consumed unnecessarily. On the other hand, it may not be possible to lazily acquire [Kirc01] all the resources since some of the resources may be required immediately at application startup or recovery. To address these conflicting requirements requires resolution of the following forces:

- Acquisition of resources should be influenced by parameters such as available system memory, CPU load, and availibility of other resources.
- The solution should work for resources of fixed size as well for resources of unknown or unpredictable size.
- The solution should be scalable with the size of the resources.
- The acquisition of resources should have a minimal impact on system performance.

Solution Split the acquisition of a resource into two or more stages. In each stage, acquire part of the resource. The amount of resource to acquire in each stage should be configured using one or more strategies. For example, the amount of resource to acquire in each stage can be governed by a strategy that takes into account available buffer space, required response time, as well as availability of dependent resources. Once a resource has been partially acquired, the system can start using it assuming there is no need to have the entire resource available before it can be used.

Patterns such as Eager Acquisition and Lazy Acquisition can be used to determine *when* to execute one or more stages to partially acquire a resource. However, the Partial Acquisition pattern determines *how many* stages a resource should be acquired in along with *how much* of the resource should be acquired in each stage.

Structure The following participants form the structure of the Partial Acquisition pattern:

A *resource user* acquires and uses resources.

A *resource* is an entity such as instances of objects or memory. A resource is acquired in multiple stages.

A *resource environment* manages several resources.

The following CRC cards describe the responsibilities and collaborations of the participants.

Class Resource User	***Collaborator***
Responsibility • Acquires and uses resources	• Resource

Class Resource	***Collaborator***
Responsibility • An entity, such as memory or a thread. • Is acquired from the resource environment and used by the resource user	•

Class Resource Enviornment	***Collaborator***
Responsibility • Manages several re-sources	• Resource

Implementation There are six steps involved in implementing the Partial Acquisition pattern.

1 *Determine how many stages the resource should be acquired in.* The number of stages in which a resource should be acquired depends upon system constraints such as available memory and CPU as well as other factors such as timing constraints and availability of dependent resources. For a resource of unknown or unpredictable size, it may not be possible to determine the number of stages that it would take to acquire the entire resource. In this case, the number of stages would not have an upper bound and a new stage would be executed until the resource has been completely acquired.

In the case of the NMS in our example, the number of stages could correspond to the number of levels of hierarchy in the topology tree. In each stage, an entire level of the hierarchy can be constructed by obtaining the details of the components of that level. If a level in the hierarchy is complex and contains a large number of components, then obtaining the details of all the components of that level can be further divided into two or more stages.

2 *Determine when each stage of resource acquisition should be executed.* Patterns such as Lazy Acquisition and Eager Acquisition can be used to control when one or more stages of resource acquisition should be executed. For example, Eager Acquisition can be used to acquire an initial chunk of the resource. The remaining chunks can then be acquired using Lazy Acquisition or can be acquired some time in between — that is after the system has started but before a user requests for them.

In the case of NMS, Eager Acquisition can be used to fetch the details of the NEs but not of its components. The details of the components and sub-components of an NE can be fetched using Lazy Acquisition when the user selects the NE in the GUI and tries to view the details of its components.

3 *Configure strategies to determine how much to acquire partially in each stage.* Different strategies can be configured to determine how much of a resource should be acquired in each stage. If the size of a resource is deterministic then a simple strategy can evenly distribute the amount of the resource to acquire in each stage among all the stages of resource acquisition. A more complex strategy would take into account available system resources. Thus, for example if sufficient memory is available, such a strategy would acquire a large chunk of the resource in a single stage. At a later stage if the system memory is low, the strategy would acquire a smaller chunk of the resource.

Such an adaptive strategy can also be used if the size of the resource to be acquired is unknown or unpredictable. Additional strategies can be configured that make use of other parameters such as required response time. If there are no system constraints, another strategy could be to greedily acquire as much of the resource as is available. Such a strategy would ensure that the entire resource is acquired in the fastest possible time. A good understanding of the application semantics is necessary to determine the appropriate strategies that should be used.

4 *Determine whether the partially acquired resource should be buffered*. Buffering a resource can be useful if the size of the resource is unknown or if the entire resource needs to be consolidated in one place before being used. If a resource needs to be buffered, the amount of buffer space to allocate should be determined to ensure the entire resource can fit. For a resource of unknown or unpredictable size, a buffer size should be allocated that is within the system constraints (e.g. available memory) but sufficiently large enough to handle the upper bound on the size of the resources in the system.

5 Set up a mechanism that is responsible for executing each stage of resource acquisition. Such a mechanism would then be responsible for acquiring different parts of a resource in multiple stages. Patterns such as Reactor [POSA2] can be used to implement such a mechanism. For example, a reactor can be used to acquire parts of a resource as they become available. An alternative mechanism can be set up that acquires parts of the resource proactively.

6 *Handle error conditions and partial failures.* Error conditions and partial failures are characteristic of distributed systems. When using Partial Acquisition, it is possible that an error occurs after one or more stages have completed. As a result, part of resource may have been acquired but the attempt to acquire the subsequent chunk would fail. Depending upon the application semantics, such a partial failure may or may not be acceptable. For example, if the resource being acquired in multiple stages is the content of a file, then a partial failure would make the data acquired successfully inconsistent. On the other hand, in the case of the NMS in our example, the failure to get the details of one of the subcomponents will not have such an impact on the details of the remaining components acquired successfully. A partial failure could still make the details of successfully acquired components available to the user.

One possible way to handle partial failures is to use the Coordinator [Jain01] design pattern. The pattern can help ensure that either all stages of resource acquisition are completed or none are.

Example Resolved Consider the example of an NMS that is responsible for managing a network of several NEs. Using the Partial Acquisition pattern the acquisition of the details of the NEs along with their components can be split into multiple stages. In the initial stage only the details of the NEs will be fetched from the physical network. The details of the components of the NEs will not be fetched.

Once the system has started up, the details of the components and sub-components of the NEs can then be fetched. If an interceptor [POSA2] is used [See Variants], this can be done transparently to the user. The interceptor can fetch the details of the components and sub-components in the background and make them available by the time the user requests for them.

Using Partial Acquisition, only the top-level of the topological tree is therefore constructed at system start-up or at the time of recovery. The remaining tree is constructed in one or more stages later on. The result is a significant increase in performance.

Variants *Transparent Partial Acquisition*: An interceptor can be introduced that would intercept user requests for resource acquisition and using configured strategies, acquire an initial part of a resource. It would then acquire the remaining parts of the resource in additional stages transparent to the user. For instance, in the motivating example, an interceptor can be used to intercept user requests and only fetch the NEs. The interceptor would not fetch the subcomponents right away; they can be fetched at a later stage transparent to the user.

Known Uses **Incremental Image Loading —** Most modern web browsers such as Netscape [NETSCAPE], Internet Explorer [IE], HotJava [HOTJAVA], and Mosaic [MOSAIC] implement Partial Acquisition by supporting incremental loading of images. The browsers first download the text content of a web page and at the same time create markers where the images of that web page will be displayed. The browsers then download and display the images incrementally while the user can read the text content of the web page.

File Input — Reading data from a large file in an efficient manner makes use of Partial Acquisition pattern. Instead of reading the contents of a large file in one step, multiple read operations are typically performed. Each read operation partially acquires data from the file and stores it in a buffer. The size of the buffer is pre-allocated depending upon the size of the file. Once all the read operations complete, the buffer contains the contents of the file consolidated.

Socket Input — Reading from a socket also typically makes use of Partial Acquisition pattern. Since data is typically not completely available at the socket, multiple read operations are performed. Each read operation partially acquires data from the socket and stores it in a buffer. Once all the read operations complete, the buffer contains the final result.

Data-driven Protocol-compliant Applications — Data-driven applications that adhere to specific protocols also make use of Partial Acquisition pattern. Typically such applications follow a particular protocol to obtain data in two or more stages. For example, an application handling an HTTP request typically reads the HTTP header in the first step to determine the size of the request body. It then reads the contents of the body in one or more steps. Note that such applications therefore use partially acquired resources. In the case of the application handling an HTTP request, the application makes uses of the HTTP header obtained in the first stage.

Consequences There are several **benefits** of using the Partial Acquisition pattern:

- *Reactive Behavior:* The Partial Acquisition pattern allows acquisition of resources that become available slowly or partially. If this partial acquisition would not be done, the resource user would have to wait an undefined amount of time until the resource becomes available completely.
- *Scalability:* The Partial Acquisition pattern allows the size of the resource being acquired to be scalable. The number of stages in which a resource is acquired can be configured depending upon the size of the resource being acquired.
- *Configurability:* The Partial Acquisition pattern can be configured with one or more strategies to determine how many stages to acquire a resource in as well as how much of the resource to acquire in each stage.

There are some **liabilities** of using the Partial Acquisition pattern:

- *Complexity:* User algorithms that handle the resources need to be prepared to handle only partially acquired resources. This can add a certain level of complexity to applications.
- *Overhead:* The Partial Acquisition pattern requires a resource to be acquired in more than one stage. This can result in the overhead of additional calls being made to acquire different parts of the same resource.

See Also The Reactor [POSA2] pattern is designed to handle events efficently. In the case of I/O operations it allows to reactively serve multiple resources, which are acquired partially. Buffering can be used to keep partially acquired resources in each stage. This can be especially useful if the size of the resource being acquired is unknown.

Acknowledgements Thanks to the *patterns team* at Siemens AG and to our shepherd, Terry Terunobu, for their feedback and valuable comments.

References

[GHJV] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[HOTJAVA] HotJava Browser, http://java.sun.com/products/hotjava/

[IE] Microsoft Internet Explorer, http://www.microsoft.com

[Jain01] P. Jain, *Coordinator Pattern*, Proceedings of Pattern Language of Programs conference, Allerton Park, Illinois, U.S.A., Sept. 11-15, 2001

[Kirc01]	M. Kircher, *Lazy Acquisition Pattern*, European Pattern Language of Programs conference, Kloster Irsee, Germany, July 5-8, 2001, http://www.cs.wustl.edu/~mk1/LazyAcquisition.pdf
[Kirc02]	M. Kircher, *Eager Acquisition Pattern*, submitted to European Pattern Language of Programs conference, Kloster Irsee, Germany, July 4-7, 2002, http://www.cs.wustl.edu/~mk1/EagerAcquisition.pdf
[MOSAIC]	NCSA Mosaic, http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
[NETSCAPE]	Netscape Browser, http://www.netscape.com
[POSA1]	F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns,* John Wiley and Sons, 1996
[POSA2]	D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture—Patterns for Concurrent and Distributed Objects,* John Wiley and Sons, 2000