Proactive and Reactive Resource Allocation

Joseph K. Cross
Lockheed Martin Tactical Systems
P.O. Box 64525, M.S. U2X26
St. Paul, MN 55164-0525
(651) 456-7316
joseph.k.cross@lmco.com

Abstract

Resources, such as processors and communications bandwidth, are allocated to providing services. In long-running systems, it is common to re-allocate these resources, due to changing service requirements or to changing resource availability. Two patterns of resource allocation are described in this paper: reactive resource allocation, which begins when the need for reallocation arises, and proactive resource allocation, which is planned before the need arises. A supporting pattern, called Mode Observer, is also described.

Reactive resource allocation is widely used in both commercial and United States Department of Defense (DoD) distributed real-time and embedded (DRE) systems. In addition, reactive resource allocation is the focus of a vigorous research and development community. Proactive resource allocation, on the other hand, seems to be employed primarily, if not exclusively, in DRE systems, where in some cases it provides the only means to meet real-time response requirements.

Introduction

This paper describes three patterns: the Proactive Resource Allocation pattern, the Reactive Resource Allocation pattern, and the Mode Observer pattern. The two resource allocation patterns have overlapping contexts and problems, and this common context and problems areas of the two resource allocation patterns are described immediately below. The third pattern, Mode Observer, supports Proactive Resource Allocation and is more widely applicable.

Resource Allocation Context

The Proactive Resource Allocation and the Reactive Resource Allocation patterns apply within the following context:

- A system with quality of service (QoS) constraints that is subject to changing service requirements and/ or changing availability of resources
- A system that is resource-constrained, as may be required by physics or economics, to the extent that some resources must be used for different purposes at different times.
Systems in this context undergo resource reallocations from time to time. We use the term *resource allocation* to refer to the process of causing specific system functions to use specific system resources. We call the process of deciding how to allocate the system’s resources, both for the system’s initial, power-up, state and for subsequent reallocations, *allocation determination*. For example, deciding which of several processors will execute a specified thread is an allocation determination; actually inserting that thread into the processor’s ready queue is a resource allocation. Of course, allocation determination must occur before the corresponding resource allocation.

**Resource Allocation Problem**

The general resource allocation problem is described in this section. The Proactive Resource Allocation and Reactive Resource Allocation patterns solve sub-problems of this problem.

In long-lived, complex, computational systems, no single allocation of resources to services is likely to be permanently optimal, or even acceptably efficient. Hence resource allocation and reallocation will be required during system execution. In order to allocate its resources, a system that lies within the context described above must:

1. Gather the information on the system state to determine when a resource allocation is needed.
2. Make the necessary resource allocation determination
3. Make the corresponding resource allocation

The mechanisms for gathering state information, allocation determination, and resource allocation must resolve the following forces:

- **Benefit must exceed cost**: The benefit of reallocating resources must exceed the cost of performing the preceding three activities. This includes, of course, that the reallocation process must do no harm, such as leading to instabilities. Other reallocation costs to be considered include the consequences of losing in-transit messages, and the results of breaking TCP connections.

- **Need for speed**: All three of the preceding activities must be completed quickly enough that the system continues to satisfy its performance requirements, or, if that is not possible, quickly enough that the violations of performance requirements are tolerable.

- **Need for flexibility**: The allocation determination function should be flexible, to allow for resource failures and the installation of new resources.

Note that resource allocations are not required to be optimal, and indeed on-line resource allocations (see below) for medium to large systems are very unlikely to be optimal according to any useful measure.

The possibility of overload must be anticipated. That is, the system may not have enough resources to satisfy all demands. The patterns presented here offer no new responses to such situations, but only a means to detect the condition.
Categories of Solutions

Solutions to the resource allocation problem can be divided along several dimensions: on-line vs. off-line, static vs. dynamic, eager vs. lazy, and reactive vs. proactive. These distinctions are discussed in this section.

Reactive vs. Proactive

Proactive resource allocation is characterized by making resource allocation determinations for system states other than its present state, so that a resource allocation determination will be available when it is needed. Reactive Resource Allocation makes resource allocation determinations only when the system state requires a new allocation. Both Proactive Resource Allocation and Reactive Resource Allocation actually perform their allocations when the allocations are needed.

On-Line vs. Off-Line

On-line resource allocation determinations are made while the subject system is operating. Off-line resource allocation determinations are made otherwise. Resource allocation determinations that are made during system design are necessarily off-line. Some resource allocation determinations are made while the subject system is operating but they are made by a system that is functionally disconnected from the subject system (e.g., the allocation of satellite communications resources by ground stations); we consider these to be on-line resource allocation determinations.

All off-line resource allocation determinations are proactive. On-line resource allocation determinations may be reactive or proactive; examples are given in the pattern definitions below.

Static vs. Dynamic

Both resource allocation determinations and resource allocations can be either static or dynamic. Here, static means unchangeable, and dynamic means changeable.

A static resource allocation determination may apply to a system for its entire lifetime, or only under certain circumstances; e.g., “When the system is in recovery mode, resources shall be allocated as follows….” A dynamic resource allocation determination may change during the lifetime of a system.

A static resource allocation determination is generally proactive. (It is technically feasible for a resource allocation determination to be made when the need for it first arises – e.g., when the system first enters recovery mode – and for that resource allocation determination to not be permitted to change. This would be a static, reactive resource allocation determination.)

A static resource allocation persists for the lifetime of the system. Systems with static resource allocation perform neither reactive nor proactive resource allocation.

Note that the static/dynamic distinction applies to an implementation, while the proactive/reactive distinction is closer to a functional requirement.
Eager vs. Lazy

An eager resource allocation is made before the resource is to be used; a lazy resource allocation is made only immediately before the resource is to be used.

Any proactive resource allocation is eager.

A reactive resource allocation may be eager or lazy. Suppose that a reactively-allocated system enters a state in which a TCP connection will be needed and none is now allocated. If the TCP connection is made at this point, then that is an eager, reactive, resource allocation. But if the TCP connection is not made until data is ready to be transmitted across the connection, then that is a lazy, reactive, resource allocation.

Reactive Resource Allocation

The Reactive Resource Allocation design pattern maintains the performance of a computing system within required bounds by observing when the system’s performance is close to crossing, or has crossed, a threshold; determining how best to allocate the available resources to prevent or correct any requirement violation; and then performing that resource allocation.

Example

The combat system on a warship contains an object called a tracker. The tracker serves as a real-time database of tracks, where a track is an object that represents a physical, moving object that is of tactical interest to the ship, such as a friendly or hostile aircraft, missile, ship, or submarine. The tracker accepts inputs from sensors such as radars, and uses these inputs to update its tracks. The tracker’s task is computationally intensive, and its performance is subject to severe real-time constraints. The number of CPU cycles per second that the tracker requires depends on the number of tracks in its database and the number of sensor reports per time interval.

A performance monitor object continuously observes the behavior of the tracker. The monitored quantities may be direct measures of its performance, such as the time required for the tracker to respond to queries, or indirect measures, such as the number of tracks currently in the tracker’s database.

When the performance monitor notes that a measure of performance is approaching a pre-defined limit, the performance monitor notifies a resource allocator. The resource allocator examines the currently available resources and reallocates them appropriately. For example, the resource allocator may split the tracker object into several sub-trackers, where each sub-tracker executes on a different processor, and each sub-tracker maintains only a subset of the entire set of active tracks. Interactions with the (now virtual) tracker object that are initiated by other objects are routed to the appropriate sub-tracker by a servant locator [2] that implements the Interceptor pattern [4].

An excellent example of reactive resource allocation technology is provided by the Quality Objects (QuO) project [5].
Context
A system as described in Resource Allocation Context section above, together with the conditions that:

- The system must be able to tolerate violations of its performance specifications for a (possibly unbounded) period of time, as the system adapts to the requirement and/or resource change
- Neither the exact time nor circumstances to which the system will have to adapt can be fully specified in advance
- Making resource allocation determinations requires information about the configuration of the system at the time adaptation must occur.

Problem
The problem is that described above in Resource Allocation Problem section above, together with the condition that

- There is sufficient time between the detection of a need for resource reallocation and the time that the allocation must be completed to make the required resource allocation determinations. This may be because the system’s performance requirements are loose, or because the resource allocation determinations can be made quickly.

In the Tracker example, the need for a resource reallocation is detected by observing that performance is degrading – not that performance has violated requirements. Hence there is not a need for fast reallocation. Moreover, the resource allocation determination is of a simple, pre-defined form: to spawn sub-trackers. Hence the resource reallocation determination process is unlikely to take very long. For these reasons, the Tracker example problem meets the “sufficient time” condition above.

Solution
Implement the monitoring function by multiple objects each of which monitors one or a few aspects of the system behavior, and comparing observed metrics with predefined threshold values. The monitored quantities may be direct measures of its performance, i.e., quality of service values, or indirect measures, i.e., quantities that have a known relationship with qualities of service.
<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>/ Trigger</td>
</tr>
</tbody>
</table>

**Responsibility**

/ Monitor one or a few aspects of system behavior
/ Compare observed values with thresholds
/ Notify the Trigger when significant change occurs

Implement the triggering function by an object that implements the Observer pattern [1] to receive notification when any of the multiple monitor objects notices a significant change in a monitored quantity. Implementing the triggering function using the Strategy pattern [1] enables the triggering decision logic to be changed easily at design time or at run time. If there are multiple independent performance requirements, consider implementing multiple triggering functions. If there is strong interdependence between multiple performance requirements, then consider implementing a higher level triggering coordinator that integrates the inputs from the lower level triggering functions.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>/ Monitor</td>
</tr>
<tr>
<td></td>
<td>/ Resource Allocation</td>
</tr>
<tr>
<td></td>
<td>Determiner</td>
</tr>
<tr>
<td></td>
<td>/ Resource Allocator</td>
</tr>
</tbody>
</table>

**Responsibility**

/ Observe Monitors
/ Initiate resource allocation determination
/ Initiate resource allocation

Perform the allocation determination function only after notification of a present or impending violation of a metric limit. The allocation determination function can delay or ignore the reallocation recommendation; for example, there may be times when the system is forbidden from reallocating. The allocation determination function typically requires a model of the current system configuration to reason about reallocations. The allocation determination function uses analytic or heuristic methods to predict whether a candidate allocation will actually improve system performance. Care must be taken in the design of the allocation determination function to maintain stability of the system; consider using a Lyapunov function for this purpose. Finally, the allocation determination function may also be implemented using the Strategy pattern [1] for the reasons stated above.
### Consequences

Reactive Resource Allocation provides the following benefits:

- The ability to maintain end-to-end qualities of service within specified bounds as the load on the system and the capabilities of the system’s infrastructure evolve smoothly over time

- The ability to make effective use of reactive capabilities that are built in to COTS components, such as routers

However, the Reactive Resource Allocation pattern encounters the following liability:

- A long and possibly unbounded time may be required to react to a sudden change in service requirements or infrastructure capabilities if the change invalidates a large proportion of the current resource allocations, if the infrastructure is complex, or if the infrastructure is geographically dispersed
See Also

Proactive Resource Allocation (below) solves a similar problem in a similar context. The difference between the two lies in when the resource allocation determination is made.

In Reactive Resource Allocation, the resource allocation determination is made only when the need for a resource reallocation is observed; while this permits the decision to be made with the most current information about the system and its environment, it interposes the decision-making time between the need for a reallocation and its accomplishment.

In Proactive Resource Allocation, the resource allocation determination is made in advance of the need to allocate, so that allocation can be fast. In general, not all circumstances that might require a resource reallocation – such as failures of multiple system components – can be planned for.

Note that there is not a rigid distinction between Reactive and Proactive Resource Allocation. For example, consider a battleship that detects an incoming anti-ship cruise missile on its port side, and then undertakes planning a resource reallocation in anticipation that the missile impact may cause the loss of power to all equipment on the port side of the ship. This resource allocation determination is made in anticipation of an imminent need, and could be considered to be either reactive or proactive.

Several systems, such as DeSiDeRaTa [6] implement the monitoring function using multiple objects.

Nobel and Weir [7] address resource reallocation, with an emphasis on various forms of memory resources.
Proactive Resource Allocation

The Proactive Resource Allocation design pattern maintains the performance of a computing system within required bounds by anticipating critical changes in the system state, planning resource allocations appropriate to those changes, monitoring the system state for those changes, and implementing the appropriate pre-planned resource allocation when the system state changes.

Example

In order to improve the quality of life for personnel on board future warships, plans call for the distribution of crew entertainment video over the ship’s backbone communication infrastructure.

When such a ship transitions from its normal mode of operation to battle mode, as might be triggered by the detection of an incoming missile, the distribution of such entertainment video must be immediately terminated and the communication bandwidth it occupied must be re-allocated to more critical functions. Since there will not be time for protracted negotiations among critical functions for access to the communication resources, such access must have been pre-planned.

Context

A system as described in Resource Allocation Context section above, together with the conditions that:

- The system may not be able to tolerate violations of its performance specifications as the system adapts to the requirement and/or resource change
- At least some of the times or circumstances under which the system will have to adapt can be specified in advance
- System state is stable enough that resource allocations decisions can be usefully made in advance

Problem

The problem is that described above in Resource Allocation Problem section above, together with the condition that

- There is insufficient time between the detection of a need for resource reallocation and the time that the allocation must be completed to make the required resource allocation determinations. This may be because the system’s performance requirements are stringent, or because the resource allocation determinations cannot be make quickly.

In the crew entertainment video example, the need for a resource reallocation is detected by observing an incoming missile – very little time may be available before the new resource allocation must be implemented. Moreover, the resource
allocation determination may be very complex, involving negotiations among many sources and sinks of data distributed over a wide area. Hence the resource reallocation decision-making process is likely to take a significant length of time. For these reasons, the crew entertainment video example problem meets the “insufficient time” condition above.

**Solution**

Implement the allocation determination function to plan resource allocations immediately when a requirement for resources is specified and the available resources are known. Note that resource requirements may (and should) be specified for system conditions other than the current condition; this permits resource allocations to be planned in advance of need. Use the Active Object pattern [4] to perform such anticipatory planning.

Implement the allocation determination function with the Reflection pattern [3], whereby *metaobjects* are associated with each infrastructure component, where each metaobject can answer interrogations concerning the ability of the associated infrastructure component to provide a specified quality of service for a specified additional load in a specified mode. A metaobject must retain in its state the set of previously committed-to qualities of service and loads, both as a function of mode. For example, the metaobject for a LAN could determine whether the LAN could carry a specified additional traffic load with specified latency and reliability, whenever the system is in a specified mode. Consider the use of the Strategy pattern [1] to support the creation of metaobjects for resources that were not anticipated at system design time.

Implement the triggering function however is appropriate to the system. Consider using the Mode Observer pattern, defined below.

**Consequences**

Proactive Resource Allocation provides the following benefits:

- The ability to provide specified end-to-end qualities of service quickly following an anticipated change in system state
- The ability to determine that specified qualities of service cannot be provided in advance of their need (and thereby to prevent a problem rather than responding to the problem)

However, the Proactive Resource Allocation pattern encounters the following liabilities:

- Not every mode change can be anticipated (e.g., multiple resource failures); hence support for some reactive resource allocation is generally desirable.
See Also

Reactive Resource Allocation (above) solves a similar problem in a similar context. See the comparison between reactive and proactive resource allocations, and the related efforts in that section.

Mode Observer

The Mode Observer design pattern allows an object to act as on Observer [1] of the mode of a system, even when the space of states of the system is very large.

Example

Consider a system that is composed of a hierarchy of parts called configuration items. A configuration item may be small (such as a motherboard in a computer) or large (such as a ship). A configuration item may exist statically (as does a router) or may be created and destroyed dynamically (as is a thread within a process). Configuration items may contain other configuration items.

Every configuration item is always in one of a fixed, finite set of states. For example, a workstation may be in a training state or an operational state, and a radar may be in a search state, tracking state, self-test state, or off-line state. The state of a configuration item may (but need not) be a function of the states of its contained configuration items.

A mode of this system is a Boolean function on the states of its constituent configuration items. For example, “the ship is in battle state” is a mode, and “both engines are in their operational states” is a mode.

Since many Boolean functions have the value true on any given set of states of configuration items, it follows that the system is simultaneously in many modes. The value of a mode can change abruptly. For example, the failure of a component affects the modes of a system.

Suppose the system is required to take some action when the system state undergoes certain transitions. For example, if the system leaves the “both engines are operational” state, then an alert must be displayed at the helm.

Context

An application that must react to changes in its state, where the state is complex.

Problem

The states of a system may be too numerous to enumerate.

In the display-alert example, the capability could be implemented by representing the system state as an object, and implementing the Observer pattern, wherein the
system state object is the subject, observed by the alert-displaying function at the helm. But if the system contains thousands of objects, each of which can be in several states, then the state space of this system is far too large to manage as a single value.

**Solution**

For each mode transition that must be reacted to, create a Responder object.

<table>
<thead>
<tr>
<th><strong>Class</strong></th>
<th><strong>Collaborator</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reponder</td>
<td>Component Metaobject</td>
</tr>
</tbody>
</table>

**Responsibility**

- Register with Component Metaobjects
- Update state according to changes in components
- Be observed by the application

For each component of the system whose state is involved in the mode transition, create a Component Metaobject. Have the Responder observe the relevant state changes in the relevant Component Metaobjects, and update its own state accordingly.

<table>
<thead>
<tr>
<th><strong>Class</strong></th>
<th><strong>Collaborator</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Metaobject</td>
<td>Responder</td>
</tr>
</tbody>
</table>

**Responsibility**

- Maintain internal state according to the state of a system component
- When specified changes in state occur, notify registered Responders

Let the application observe the Responder.

*In the display-alert example, the Responder object is created for the state “both engines are operational”. There are two relevant Component Metaobjects: one for each engine. The Responder registers with both of these, asking to be notified when that engine transitions into or out of its operational state. The Responder can then notify the application when the state “both engines are operational” is exited, thereby causing the alert to be displayed.*

**Consequences**

The Mode Observer provides the benefit that
• The application can respond to changes in system state, even when the system states are too numerous to enumerate

However, the Mode Observer pattern encounters the following liabilities:

• Component Metaobjects must be created for all system components involved in observed state transitions, and these objects must monitor the state of their corresponding component

• Potentially many messages must be exchanged between Component Metaobjects and Responders

**Acknowledgments**

The author is deeply grateful to Bob Hanmer for his skill, patience, and grace in shepherding a previous version of this paper through the PLoP2002 workshop.

This work was sponsored by DARPA under contract number F33615-01-C-1847.

**References**

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns, Addison-Wesley, 1995


