

# Proactive and Reactive Resource Reallocation in DoD DRE Systems

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

Patrick J. Lardieri

Lockheed Martin Advanced Technology Labs  
1 Federal St., A&E 3W  
Camden, NJ 08102  
patrick.j.lardieri@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 styles of resource reallocation are commonly used: reactive resource reallocation, which begins when the need for reallocation arises, and proactive resource reallocation, which is planned before the need arises.

Reactive resource reallocation is widely used in both commercial and United States Department of Defense (DoD) distributed real-time and embedded (DRE) systems. In addition, reactive resource reallocation is the focus of a vigorous research and development community. Proactive resource reallocation, 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.

## **1 Problem and Context**

Reactive Resource Reallocation and Proactive Resource Reallocation are solutions to a common problem in a common context. This context and problem are described first.

### **1.1 Context**

A system with quality of service constraints that is subject to changing service requirements and/or changing availability of resources. The system must be 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.

### **1.2 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 reallocation will be required during system execution. The system state must be monitored, reallocation of resources must be triggered, a more nearly optimal resource allocation must be determined, and the reallocation must be carried out.

The mechanisms for triggering, allocation determination, and reallocation must resolve the following forces:

- The monitoring function must be impose a minimal overhead on the normal operation of the system; in particular, the benefit

of reallocating resources must exceed the cost of for monitoring.

- The triggering and allocation determination functions must be either predictive or fast, and the reallocation function must be fast, in order to minimize the period of time during which system quality of service constraints are violated.
- The allocation determination function should be flexible, to allow for resource failures and for the installation of new resources.
- The allocation determination function can be simplified and accelerated by restricting certain resource to support only specified services (“stovepiping”), but resource utilization will likely be thereby decreased.
- The combination of the triggering function with the allocation determination function must be stable, to ensure that after reallocation another reallocation will not be immediately initiated, which could lead to system collapse.
- The allocation determination function must respect the criticalities of current allocations, such as TCP connections that must not be broken, or that messages must not be lost.
- The reallocation function must respect criticalities of on-going services (e.g., avoiding dropped messages)

## **2 Reactive Resource Reallocation**

The Reactive Resource Reallocation 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 reallocate the available resources to prevent or correct any requirement violation; and then performing that resource reallocation.

### **2.1 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.

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 reallocator*. The resource reallocator examines the currently available resources and

reallocates them appropriately. For example, the resource reallocator 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 [MetaMech] that implements the Interceptor pattern [POSA2].

An excellent example of reactive resource reallocation technology is provided by the DARPA Quorum project [Quorum].

## 2.2 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.

Implement the triggering function by an object that implements the Observer pattern [GoF] to receive notification when any of the multiple monitor objects notices a significant change in a monitored quantity.

Perform the allocation determination function only after notification of a present or impending violation of a metric limit.

## 2.3 Consequences

Reactive Resource Reallocation 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 Reallocation pattern encounters the following liabilities:

- A long 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

## 3 Proactive Resource Reallocation

The Proactive Resource Reallocation 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 reallocation when the system state changes.

### 3.1 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 anti-ship cruise 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.

### 3.2 Solution

Divide the (possibly very large) state space of the system into subsets called *modes*. (Abstractly, a mode may be regarded as a Boolean-valued function on the system state space; ‘battle mode’ is a mode, and ‘at least one engine is operational’ is a mode.) Express quality of service and functional requirements as functions of mode; e.g., “When at least one engine is operational, engine RPM values shall be distributed with a latency of at most 50 milliseconds.”

Implement the triggering function by monitoring relevant aspects of the system state. Create multiple triggering objects, each of which responds to one relevant mode change. Use the Observer pattern [GoF] to cause each triggering object to receive notification of system state changes that are relevant to the triggering object’s mode change.

Design 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 modes other than the current mode; this permits resource allocations to be planned in advance of need. Use the Active Object pattern [POSA2] to perform such anticipatory planning.

Implement the allocation determination function with the Reflection pattern [POSA1], 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 [GoF] to support the creation of metaobjects for resources that were not anticipated at system design time.

### 3.3 Consequences

Proactive Resource Reallocation 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 Reallocation pattern encounters the following liabilities:

- A metaobject class must be created for each infrastructure component class.
- Not every mode change can be anticipated (e.g., multiple resource failures); hence support for some reactive resource reallocation is generally desirable

## **4 Acknowledgment**

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

## **5 References**

- [GoF] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, Addison-Wesley, 1995
- [MetaMech] N. Wang, K. Parameswaran, and D. C. Schmidt, "The Design and Performance of Meta-Programming Mechanisms for Object Request Broker Middleware," in *Proceedings of the 6<sup>th</sup> Conference on Object Oriented Technologies and Systems* (San Antonio, TX), USENIX, Jan/Feb 2000
- [POSA1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture—A System of Patterns*, John Wiley and Sons, 1996
- [POSA2] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture—Patterns for Concurrent and Distributed Objects*, John Wiley and Sons, 2000
- [Quorum]  
<http://www.darpa.mil/ito/research/quorum/index.html>