The building block method. Component-based architectural design for large software-intensive product families
Müller, J.K.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
The core BBM can be specialised for specific kinds of systems. Additional architectural patterns and guidelines are integrated into the core method to make it suitable for these kind of systems.

An important question is how the BBM is to be applied to distributed systems such as distributed client-server systems or centrally-controlled distributed systems. In this chapter, we want to show the specialisation of the core BBM for centrally-controlled distributed embedded systems. This architectural style is quite common for large electronic products. Also the tss product family uses that style.

Other specialisations for distributed client-server systems are also possible. Technologies such as COM+ (.NET) and Enterprise JavaBeans provide a good basis for the BBM. Their support for different types of aspect functionality such as persistence and security and for SW components fit with the concepts of the BBM.

The main points of this chapter is to show how the application of the BBM to the central controller of such a system leads to specific objects, layers and aspects. First, the notion of a managed object will be introduced as an extension of the object notion presented in chapter 4. Second, at least one additional standard layer of functionality called equipment management will be introduced. Third, a number of standard aspects such as configuration management, fault management and performance observation are introduced. These specific objects, layers and aspects become part of the specialised BBM because they are relevant for all systems for all centrally-controlled distributed embedded systems.
10.1 System Architecture

In this section we describe the system architecture of many centrally-controlled distributed systems. Processing nodes of such systems are managed via control relations between the nodes. We take a look at the functional and the control architecture to provide a background for the additional guidelines given by the specialised BBM.

10.1.1 Functional Architecture

The system architecture of embedded systems is often partitioned along the flow of domain-specific signals and streams. Signals and streams are forwarded from processing node to processing node. The complete function of the system is realised by the coordinated activities of the processing nodes. Processing nodes are implemented by for instance hardware boards or operating system processes.

Two example architectures illustrate that situation. The first example concerns telecommunication infrastructure systems, the second medical imaging systems.

The architecture of the tss system is partitioned into I/O boards, that is, subscriber and trunk line cards and a central switching matrix (see figure 76 on page 198). Telephone calls are switched by the switching matrix from their incoming lines to outgoing lines. The central controller handles the hardware configuration, determines switching parameters and handles call facilities.

The architecture of a medical imaging system [Pro99] is partitioned along the flow of image data. An acquisition process handles the generation of imaging data by the front-end equipment, e.g. X-ray or MR. A reconstruction process transforms the raw image data into basic images. Image enhancement procedures are applied to basic images. A viewing application interactively allows enhancement and annotation of images. Images are then forwarded to diagnostic workstations, printed and/or stored on different media.

The two examples illustrate functional architectures. Object-oriented modelling, as advocated by the BBM (see chapter 4), takes place within the system’s overall functional structure. Hardware entities as well as logical entities such as images and calls are naturally modelled as objects.

Aspects, as defined by the BBM (see chapter 5), are an example of functional modelling. They constitute a refinement of the overall functional structure and are used as standard substructures of BBs (see section 5.6).
A similar approach to the functional refinement is presented in [BM99] where iterations of architectural transformation are following the definition of an initial functionality-based architecture. Architecture transformations are used to adjust the initial architecture to meet quality requirements like maintainability, performance and reliability. Requirements for each of the qualities are met by transforming the architecture in such a way that the new architecture is functionally equivalent and meets the quality requirements.

### 10.1.2 Control Architecture

In centrally-controlled systems, a specific processor, the central controller (CC), runs the central control software. In the hierarchical control relation, the central controller has the role of controlling equipment, while the so-called peripheral equipment has the role of controlled equipment. If the control relation has more than two levels, so-called intermediate controllers (sometimes also called peripheral group controllers (PGC)) have both roles, controlled equipment and controlling equipment. Such an architecture is called central control architecture.

Distributed control architectures also exist but are, in general, difficult to evolve due to the built-in shared information. An interesting alternative to distributed control systems are SPLICE-based systems [Boa93b]. A data backbone forms the essential infrastructure of these systems. Problems, such as management of inter-component relations, can be avoided because of this backbone [Boa93a].

Figure 58 shows a control architecture with tree-type relations. General hierarchical relations are also possible. The exact structure is important for modelling equipment control software (see section 10.2.2).
The advantages of a central control architecture are its low implementation complexity and easy extensibility. Knowledge about a specific functional unit is only needed in the directly cooperating units and the controlling equipment. Disadvantages are the single point of failure and the processing bottleneck. These disadvantages can be addressed in the following ways:

the single point of failure can be avoided by additional reliability measures such as redundancy of controllers in a cold or hot stand-by configuration (see section A.2.2), and

the processing bottleneck can be addressed by locating only functions at the central controller which have a coordinating character over the periphery. Protocols between the central controller and the peripheral processing units need to be designed so that consistency-preserving actions have priority (see section 3.2.3) and the central controller can exercise flow control.

The quality that can be achieved with a central control architecture is often sufficient.

Management for Centrally-Controlled Systems

An extension of the central control architecture introduces connections to one (or more) management system(s). The resulting system comprises three stages, with the central controller in the middle stage (figure 59). The functionality of the three stages can be characterised as follows:

1. Domain-specific signal and stream processing may be effected in HW and/or SW. The use of specific processing HW with its usually better performance has to be weighted against the usually cheaper and more flexible general-purpose hardware. Often a mixed solution is chosen. This is the field of hardware / software co-design. Processing usually has to meet strict timing requirements. Processing units in this domain are connected, in the most general case, to form a network of streaming relations. Control relations are restricted to a hierarchy. The BBM does not address this area specifically.

2. Building a system consisting of several processing units requires that they be coordinated and kept in a consistent state. Large systems have hundreds or even thousands of those processing units. The coordination and control func-
tion is called system control or embedded system control. It has to bring, and subsequently keep, a system automatically in a consistent state, possibly support graceful degradation and transition to a fail-safe state. This is the area where the specialised BBM is aimed at.

3. The third stage comprises the functions that are responsible for the system’s management. They have to support (different kinds of) operators in their daily work. Management may be local or remote, affect single systems or networks of systems, and provide different kinds of user interfaces. Timing requirements are oriented at human perception. The specialised BBM can be used here as well.

Table 6 characterises the three stages by giving typical examples.

<table>
<thead>
<tr>
<th></th>
<th>signal and stream processing</th>
<th>system control</th>
<th>system management</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality</td>
<td>depending on system size, different types (between 5 and 50) and instances (between 20 and 1000)</td>
<td>logically 1, may be duplicated for reliability reasons</td>
<td>usually small: &lt; 10, specialisation leads to different types</td>
</tr>
<tr>
<td>main functions</td>
<td>signal and stream processing</td>
<td>automatic recovery (and re-configura- tion) &amp; centralised logical processing</td>
<td>flexible operator support</td>
</tr>
<tr>
<td>configuration management</td>
<td>establish processing element parameters</td>
<td>configure equipment &amp; functions (reference DB for system state)</td>
<td>manage configuration data</td>
</tr>
<tr>
<td>fault management</td>
<td>self supervise processing</td>
<td>monitor HW and SW configuration, automatic recovery from faults and failures</td>
<td>alarms, fault notifications, error logs</td>
</tr>
<tr>
<td>performance observation</td>
<td>generate appropriate data</td>
<td>monitor system performance</td>
<td>performance logs, statistics</td>
</tr>
</tbody>
</table>

Table 6: Typical Functional Distribution Over The Stages
Extreme cases are not mentioned in table 6, since they would only extend it without adding any value to it. An example of an extreme case is the requirement to re-configure in a situation of failure in such a way that the system's signal and stream processing function is not noticeably interrupted. Such a case brings hard-real time requirements to the central controller.

Another observation is that with most of these systems the greatest complexity lies in the embedded system controller because it pertains to the whole of the processing units, has to supervise them, execute centralised logical processing, receive configuration change commands from the management system and report configuration changes and errors to the management system. This is illustrated in figure 60, which represents the connection structure of the system controller. The number of connections on the left are much more than the ones on the right. Possible direct connections between the processing units have been omitted.

![Connection Structure of a Central Controller](image)

Figure 60: Connection Structure of a Central Controller

Yet another point is that the core system, consisting of the processing units and the system controller, must be able to run without the management systems. The core system is a highly reliable subset. It must run autonomously and maintain system consistency. The reasons for this are that operators cannot be forced to be "on-line", and that the management systems might be located at another site with less reliable connections.

### 10.2 Additional Guidelines

In the following we present additional guidelines for four of the BBM design tasks, namely object design, aspect design, composability design and deployability design. The guidelines are related to the system architecture as described in the previous section.
10.2.1 Object Design

Modelling the functionality of a system controller (section 10.1) is different from modelling functionality which is assumed to run on a network of distributed peer nodes. The location of an object is not transparent. Part of the functionality of the system controller deals with bringing and keeping the entire system in a consistent state. As explained in section 10.1, a system controller is a processing bottleneck of the system. System functionality has to be designed so that the consistency-maintaining functions of the controller are not impeded by this bottleneck (see section 3.2.3).

As with the core BBM, object design starts with the application domain model. Domain objects are used as an initial implementation object model (see chapter 4).

However, modelling hardware and software of the rest of the system relies on the concept of a managed object. A managed object (MO) models a real resource (RR) for the purpose of control or management [X700]. It encapsulates the underlying resource and allows its manipulation through well-defined operations (figure 61).

![Figure 61: Managed Object](image)

There are two types of real resources, namely physical and logical ones. A typical example of a physical real resource is a processing board and an example of a logical real resource is a software processing node. If a logical real resource is located on the system controller itself, it is collapsed and joined with its managed object. We shall base our discussion on the abstract view.

This means that we shall not discuss communication protocol issues which are assumed to be local to managed objects.

The concept of a managed object is comparable to that of a proxy object [GHJ*94] [BMR*96]. The proxy design pattern makes the clients of an object communicate with a representative, i.e. the proxy object, rather than with the object directly.
The mapping of domain objects to the system is more complex than the one for the core BBM. Application functionality will be distributed over peripheral units (PU) and the central controller (see section 10.2.4). Furthermore, the PUs will have control objects in the equipment maintenance layer of the CC (see section 10.2.2). In figure 62, the arrows show the mapping of domain objects into system objects and the mapping of hardware entities into the OS and EM layer of the CC.

**Heuristic 84:** A managed object may consist of an object in the CC and an object in the peripheral hardware.

**Heuristic 85:** Hardware objects and hardware abstractions of the CC will often be part of the OS.

Heuristic 86: Maintenance replaceable units (MRU) are good candidates for hardware managing objects.

Heuristic 87: Represent MRUs, which only together realise a specific function in the system, by one hardware managing object.

*Figure 62: Mapping of External Objects to Internal Objects*
In the latter case, the hardware handling object needs to be updated whenever an MRU is replaced.

The object design of the tss product family is described in section A.3.1.

**10.2.2 Composability Design**

As described for the core BBM, layering is very common in modelling functionality for electronic systems. The main reason for this is a desire to differentiate between hardware and the functionality realised by this hardware. The functionality realised by the hardware is part of the application domain and evolves with the application domain. The selection of functionality, its partitioning and its implementation technology change over time. The abstract nature of software makes the coupling of application functionality and solution technology a loose one.

Heuristic 88: *For the specialised BBM, we will always have the two layers in the central controller, namely application and equipment management.*

The equipment management layer (see figure 63) contains MOs for the hardware entities and the application layer contains MOs implementing application functionality. The equipment management layer is sometimes also called hardware reflection layer since it mirrors the hardware.

On the basis of these two layers, which contain objects representing the application functionality and support functionality, additional layers may be appropriate. We shall give examples in which functionality is so extensive that additional layers are reasonable.

Heuristic 89: *When interface abstractions between the two layers have themselves state and behaviour create a new layer for these abstractions. They are then to be modelled as managed objects in their own right and be represented as an intermediate layer.*
Examples of such interface objects are abstract devices and logical resources. They are abstractions of the real hardware with the property that they evolve at a slower speed than the actual hardware. Figure 64 shows the logical resource layer between

```
Application
Logical Resources
Equipment Management
```

*Figure 64: Three Layers*

the two basic layers (see also [MHM98]).

**Heuristic 90:** A further division may be appropriate if additional abstractions are introduced to abstract from the distribution of the controller over several sites. The application functionality then runs on top of the multi-site abstractions.

Functionality such as channel abstractions with data transmission bandwidth and redundancy handling are assigned to objects of such a layer. The logical resources layer is split in two: a lower layer for logical devices and single-site abstractions, and a higher one for multi-site abstractions (figure 65).

```
Application
Multi-site Resources
Single-site Resources
Equipment Management
```

*Figure 65: Four Layers with Multi-site Resources*

**Heuristic 91:** A different division of layers may be appropriate if application functionality extends significantly. An application-specific platform encapsulates application infrastructure abstractions. Various advanced applications may run on this platform.
See figure 66 for these layers.

```
  Advanced Application
  Basic Application Platform
  Logical Resources
  Equipment Management
```

*Figure 66: Four Layers with Basic and Advanced Applications*

*Heuristic 92: Infrastructure functionality such as basic services which should be used by all the objects implemented on the system controller are modelled in the lowest layer.*

See figure 67 for these layers.

```
  Application
  Logical Resources
  Equipment Management
  Operating Infrastructure
```

*Figure 67: Four Layers with Operating Infrastructure*

As noted earlier, layering is not inherent in the functionality but is a means of introducing structure. The purpose of layering is to achieve separation of concerns and management of complexity.

**Equipment Management Layer**

The task of the EM layer is to bring and keep the peripheral units, that is the domain-specific signal and stream processing units, in a consistent state, to administer them and to provide an application platform for the distributed application objects.

The spheres of control of the EM layers of the controllers, e.g. CC and PGC, concern the complete subtrees (see figure 68). EM offers generic recovery serv-
ices for the controlled periphery to the application layers. Applications manage to initialise themselves using these service starting from the CC on outwards.

Figure 68: Control spheres of EM

Heuristic 93: An important set of managed objects and their respective BBs concerns the handling of the PUs. The BBs in the EM layer will reflect the connection structure of the PU.

The communication structure between the EM layers of the CC and the PUs is shown in figure 69.

Figure 69: Communication Relations of EM

Section A.3.8 describes the EM structure of the tss product family

To summarise the discussion about layering we can say that centrally-controlled distributed systems have at least three layers:

the equipment management layer which is intrinsic to the task of a system controller,
a layer below the equipment management layer which utilises the HW function-

ality of the system controller itself, that is, the operating infrastructure, and

an application layer above the equipment management layer which models
the application functionality.

Object modelling and layering together are shown in figure 62.

10.2.3 Aspect Design

Aspect design for centrally-controlled distributed embedded systems can be
much more specific than for the core BBM. The reason is we can assume quite a
bit of required system functionality from the system architecture. In particular
the facts that the system is embedded, that it has a distributed architecture and
that it has a central controller induces certain types of aspect functionality.

The identification of aspects is not different to the one of the core BBM. However, several aspects can be taken for granted, namely:

a system management interfacing aspect,
a recovery aspect,
a data replication aspect,
a configuration management (control) aspect,
a fault management (error handling) aspect,
a performance management aspect.

Note that this list is not a starter set to analyse for functionality but these aspects are
inherent from the system architecture. They are types of functionality that are present
independently of the application domain.

*Heuristic 94: The system management interfacing aspect consists of the
functionality to communicate with a system management
system and with the operators.*

*Heuristic 95: The recovery aspect consists of functionality for system ini-
tialisation and automatic recovery.*

An example for the design of the recovery aspect is given in section A.3.3.4.
Heuristic 96: The data replication aspect is a consequence of the distributed architecture. It consists of functionality to replicate data within a managed object, that is, the control and management data of the control object is sent to the real resource object, and changes in the real resource object are propagated to the control object.

The configuration management aspect, the fault management aspect and the performance management aspect are closely related (see below).

Heuristic 97: The configuration management aspect establishes configuration parameters according to a system database or operator actions.

An example of the design of the configuration management aspect is given in section A.3.3.3.

Heuristic 98: The fault management aspect supervises the system configuration and takes decisions on required actions in case of failure or other abnormalities.

As an example we describe a widely used approach to fault management. It bases fault management functions on a standardised state model for all components. Failures, faults and errors are arranged into a model of fault classes. System-wide fault management leaves handling of specific faults to the context in which the fault occurred and is based only on the fault classes. Objects are responsible for handling their faults. They choose the appropriate fault class and use the reporting support which is designed for the entire system.

A basic approach for a system is to handle only hardware failures and failures on external interfaces. Fault management functions are then restricted to those objects which deal directly with hardware or external interfaces. Objects which deal only indirectly with hardware and external interfaces handle a boolean availability state. This way specific knowledge about failures remains local with the respective objects.

An extension to this basic approach also handles some addressing faults. The system functionality is separated into independently recoverable parts. Multiple address spaces are used to prevent unallowed access across these parts. Fault management ranges from reinitialising individual address spaces to reinitialising the entire system.

A standardised fault management model permits a generic solution for fault management functions. This approach is also chosen for the tss product family.

Heuristic 99: The performance management aspect has the task to monitor and register the quality of the system configuration. If
certain quality thresholds are exceeded fault management is informed.

The relations between the three aspects are shown in figure 70.

**Configuration Management**

Establish configuration according to data base or operator action.

**Fault Management**

Supervise configuration. Take decisions for recovery on detected or reported abnormalities.

Inform CM if any re-configurations are required.

**Performance Management**

Monitor and register quality of configuration.

Inform FM on exceeding quality threshold.

*Figure 70: Relations between CM, FM and PM*

The terms configuration management, fault management and performance management stem from the system management functional areas (SMFAs) of the ITU standard on system management [X700]. The standard additionally identifies the areas accounting management and security management. They may be aspects as well but they are not of such general importance because for many systems they are functional objects or not part of system functionality at all. The terms configuration management, fault management and performance management are synonyms for configuration control, error handling and performance observation, respectively.

The aspect design of the tss product family is described in section A.3.3.

**10.2.4 Deployability Design**

The deployability design follows the description of the core BBM (see section 3.2.5). In a centrally-controlled distributed embedded system the central controller has a specific role: it controls and coordinates the operation of the peripheral units and thus of the entire system.
As a central controller is a potential processing bottleneck in the system careful allocation of functionality is necessary. PUs may be either general purpose units or special purpose units with respect to a system's functionality. Allocation of objects has to take possible resource shortage into account. Some objects may be split into managed objects or be moved to the periphery if possible.

**Heuristic 100:** A quite typical design is to separate control functionality from processing functionality. Processing is allocated to the periphery, while control is allocated to the CC.

The general strategy is to let the PU do as much of the computation intensive tasks as possible. The CC functionality is restricted to control and coordination of the overall functionality.

The tss product family does not have subscriber cards with a tone generator for the dial tone. Instead it has specific peripheral cards with tone generators. In consequence each off-hook event from a subscriber leads to building up of a call in the peripheral unit, the PGU and the CC in order to be able to switch a path from the subscriber to the tone generator located on one of the service function peripheral units. Such a design puts considerable burden on the CC during call build-up limiting the overall system performance.

After an initial allocation of system functionality component boundaries, fault containment unit boundaries, and thread and process boundaries have to be aligned with HW instances.

The BBM specialisation for centrally-controlled distributed systems uses the same main design tasks as the core BBM. Based on the characteristics of the system architecture of such systems we described additional guidelines and examples for object design, composability design, aspect design and deployability design.

**Heuristics Overview**

**Heuristic 84:** A managed object may consist of an object in the CC and an object in the peripheral hardware.

**Heuristic 85:** Hardware objects and hardware abstractions of the CC will often be part of the OS.

**Heuristic 86:** Maintenance replaceable units (MRU) are good candidates for hardware managing objects.

**Heuristic 87:** Represent MRUs, which only together realise a specific function in the system, by one hardware managing object.
**Heuristic 88:** For the specialised BBM, we will always have the two layers in the central controller, namely application and equipment management.

**Heuristic 89:** When interface abstractions between the two layers have themselves state and behaviour create a new layer for these abstractions. They are then to be modelled as managed objects in their own right and be represented as an intermediate layer.

**Heuristic 90:** A further division may be appropriate if additional abstractions are introduced to abstract from the distribution of the controller over several sites. The application functionality then runs on top of the multi-site abstractions.

**Heuristic 91:** A different division of layers may be appropriate if application functionality extends significantly. An application-specific platform encapsulates application infrastructure abstractions. Various advanced applications may run on this platform.

**Heuristic 92:** Infrastructure functionality such as basic services which should be used by all the objects implemented on the system controller are modelled in the lowest layer.

**Heuristic 93:** An important set of managed objects and their respective BBs concerns the handling of the PUs. The BBs in the EM layer will reflect the connection structure of the PU.

**Heuristic 94:** The system management interfacing aspect consists of the functionality to communicate with a system management system and with the operators.

**Heuristic 95:** The recovery aspect consists of functionality for system initialisation and automatic recovery.

**Heuristic 96:** The data replication aspect is a consequence of the distributed architecture. It consists of functionality to replicate data within a managed object, that is, the control and management data of the control object is sent to the real resource object, and changes in the real resource object are propagated to the control object.

**Heuristic 97:** The configuration management aspect establishes configuration parameters according to a system database or operator actions.
Heuristic 98: The fault management aspect supervises the system configuration and takes decisions on required actions in case of failure or other abnormalities.

Heuristic 99: The performance management aspect has the task to monitor and register the quality of the system configuration. If certain quality thresholds are exceeded fault management is informed.

Heuristic 100: A quite typical design is to separate control functionality from processing functionality. Processing is allocated to the periphery, while control is allocated to the CC.