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

Citation for published version (APA):
Concurrency design is about the mapping of functionality to processing resources like threads and processes. The concept of an address space is important for concurrency design and deployability design (see section 7.9).

This chapter consists of two sections. In the first section we describe the usage of address spaces and the consequences for objects, aspects and threads. The second section describes the concurrency design focusing on an internal concurrency model.

6.1 Using Address Spaces

Both, units of deployment and units of execution make use of address spaces. Address spaces do not actually represent execution structures. Rather, they represent boundaries to execution structures. Address spaces can be a consequence of hardware boundaries or may be instantiated by software.

Symmetric multi-processing uses a single address space with several processors. Threads are executed on the processors according to allocation strategies. Identification of threads is done in the same way as with single processor address spaces.

Address spaces are used for two purposes.

Heuristic 28: Use address spaces as failure containment units. Recovery from failure is realised within an address space.

Heuristic 29: Use address spaces to design for deployability. The freedom to relocate functionality to different processors depends on the absence of common data between address spaces.

For the discussion of the BBM, we shall assume that both the object and the thread dimensions are partitioned by address spaces. In the following we shall describe the relations between the three dimensions and address spaces.
Object dimension: Objects may be internally distributed over address spaces. The concept of a managed object (section 10.2.1) does this via an asymmetrical distribution, that is, the proxy object forms part of the controlling equipment, while the real object is controlled by it. A symmetrical distribution over different address spaces which splits an object into multiple peer subobjects is also possible. We will not discuss this concept here and assume that if the concept of distributed objects is needed, it will be modelled explicitly as a collection of objects.

Thread dimension: Threads are seen as logical threads of execution which cross address space and even hardware boundaries. This view may be helpful in the early stages of architecting. Later, these logical threads are split into a number of physical threads, limited to using a single address space. The physical threads are connected via inter-address space communication. In the BBM, concurrency design maps functionality clustered into logical threads to physical threads of the execution environment.

Aspect dimension: Aspects constitute a classification of system functionality which cuts across objects and they are not affected by address space boundaries.

6.2 Concurrency Design

Concurrency design is about the mapping of functionality designed during object design and aspect design to execution structures of the computing platform. \textit{Logical threads} are used to describe parallel execution of functionality described in domain-induced objects and aspects. Physical threads are a refinement of logical threads according to address spaces. We use the term \textit{physical thread}, or thread for short, to consist of all methods and objects, which execute under its control. This is sometimes called the reach of a thread.

The use of term \textit{process} in the context of execution structures is confusing. The term \textit{physical thread} means an independent unit of processor cycles allocation having its own stack. Some real-time kernels use the term \textit{process} in this way, others use the term \textit{task}, which again is used for minimal independent execution units of application functionality. In the context of workstation operating systems, Solaris and Windows NT use the term \textit{process} either as a pure handle for resources and for an address space or as additionally having also a thread. Throughout this thesis we will omit the term \textit{process} to denote execution structures and rely on the terms \textit{thread} or \textit{address space} depending on which facet of the term \textit{process} we refer to.
Rules on the architectural level must be given to guide local design. Because of the complexity of the relation of threads to objects, aspects and BBs, a consistent global picture of all threads is essential to good system design. Such a global picture supports the understanding of local interface descriptions. The usage of threads is not considered local implementation detail.

**Heuristic 30:** Consider the use of a thread on the architectural level.

**Heuristic 31:** An overview of all threads should be given in a global concurrency design.

Besides the identification of threads we will also take a look at the identification of address spaces.

### 6.2.1 Determining Concurrency

The design starts with initial steps which are later refined.

**Starting with behaviour of domain objects**

The basic approach for structuring functionality into threads is to look for intrinsic concurrency in the application domain. Behavioural modelling of the application domain is described via domain objects, their interaction and their internal states (see section 2.6.2).

**Heuristic 32:** Mirror independent behaviour of application domain objects by separate logical threads.

**Determining independent external sources**

Independent external interaction sources are users or external devices modelled as domain objects. External concurrency is the first source for determining internal concurrency. A system’s internal concurrency structure should resemble the concurrency structure of its external environment (see above). This mirroring makes the concurrency structure easier to understand. A concurrency structure of a system with more concurrency than that of the application domain needs to introduce extra synchronisation within the system. A concurrency structure with less concurrency needs to explicitly switch between external sources. Two types of external concurrency are important.

Application concurrency is a consequence of actors in the application domain. These actors may be users or other equipment interfacing with the system.
Heuristic 33: Use a separate thread to handle an external connection or external messages.

Heuristic 34: Cluster all functionality which is activated via object interaction by the external connection or messages into the thread.

Heuristic 35: Use a separate thread for the interaction of a user with the system.

This is the user role task type of [Gom93]

Hardware equipment directly connected to a system is important for the concurrency structure of the software handling these connections. Take for instance a system controller having connections to controlled hardware equipment, or instances of the management systems. All equipment usually runs in parallel.

Heuristic 36: Represent the receiving direction of an external channel or bus by its own thread.

Heuristic 37: Message sending over an external channel is done on the budget of the sending thread

These heuristics are mentioned by Gomaa as I/O task structuring criteria [Gom93].

Heuristic 38: Refine the design of a separate thread per bus to have thread instances per connected equipment instance to the bus.

Prioritising aspect functionality

Several heuristics are used to determine priorities of aspect functionality.

Heuristic 39: Let internal consistency have priority over external reaction.

Internal consistency of the system is vital for its correct functioning.

Heuristic 40: Give operational tasks priority over background tasks.

Certain types of logging may be skipped under heavy system load while other types may be more important than operational tasks. For instance performance logging may have a lower priority than operational tasks. Error logging may have a higher priority than operational tasks. Design decisions have to be compatible with the expectations of the stakeholders.
**Heuristic 41**: Use separate thread per different priority.

An example is the separation of fault handling actions from other actions in controller SW. If a fault message is received from some equipment, the states of the managed objects of the equipment and of all the dependable equipment have to be adjusted. To limit the effects of the failure, this update action has priority over other actions.

**If necessary, encapsulating specific objects in a thread**

Internal concurrency may additionally be necessary to deal with different priorities of actions to be performed. This may concern functionality of specific objects. For instance, an emergency call may have priority over other calls.

**Heuristic 42**: Use a separate thread per cluster of objects with given priority.

This is often a mixture of Kruchten’s outside-in approach [Kru95] and Gomaa’s task priority criteria [Gom93].

**Refining the logical threads into physical threads**

Physical threads are confined to a single address space.

**Heuristic 43**: Split logical threads up into physical threads per address space.

Inter-address-space communication connects physical threads.

**Determining interfacing between threads**

Buffers or queues of messages or shared data may be used. The use of messages in the same address space may result in unnecessary copying of data (see also section 6.2.2).

Thread identification is also treated in Kruchten [Kru95] and Gomaa [Gom93]. The main difference is that in the BBM the design of the thread structure is an independent dimension besides the object and the aspect dimension.

Gomaa uses the term "object or function" when referring to the content of a thread. The BBM puts these concepts in different design dimensions. In one situation objects are used as concurrent units while in another situation aspects may be concurrent units.
6.2.2 Thread Interaction

Interaction between threads is always located within a BB. An object method calls a method of another BB in the same thread and data is transferred to the other BB. Particular BBs such as message handlers managing different kind of buffers or a socket manager may be involved. Call-backs may be used to actively deliver data to a BB (see section 7.2.4).

6.2.3 Concurrency and Aspects

Sometimes the suggestion is made that concurrency is just another software aspect. Why is that not the case?

Our definition states that an aspect is a type of system functionality. Examples are initialisation and fault management. The domain functionality is captured by the operational aspect. The list of aspects is a partitioning of a system’s functionality. All the system functionality needs to be driven by threads. The concurrency design is different from aspects, it is the mapping of aspect and object functionality to the available processor time in form of threads. The unit of functionality allocated to a thread may be one or more objects or one or more aspects or part of them (see section 6.2.1).

In the tss system the aspect functionality of two of the four layers (equipment management and logical resource management) is handled by a set of shared threads (see section 6.2.4).

6.2.4 Example: Concurrency Design of tss

The concurrency design of the entire application functionality (Equipment Maintenance, Logical Resource Management, Service Management; see section 4.3) is based on one address space and comprises six thread types.

tss service management performs call processing. A singleton thread type receives the call-initiating messages from the peripheral cards. A call thread is started to handle a new call. The call thread type has instances for the maximum number of parallel calls allowed in a system, e.g. several thousand (heuristic 32).

tss equipment management and logical resource management perform control processing in two thread types. They go along aspects and cross many objects. A fault handler covers the operational, recovery and fault management aspects (heuristic 39), while a configuration handler covers configura-
tion control and data replication. A separate thread instance is used for each equipment instance at the peripheral bus (heuristic 32).

A further thread type covers the system management interfacing aspect of all three layers of the application functionality. It is instantiated per operator (heuristic 32, heuristic 35).

The entire incoming and outgoing communication is handled via one central BB which handles the bus connection. The incoming direction is covered by a singleton thread type (heuristic 36), while for the outgoing direction functions are provided which run under the budget of the sending thread (heuristic 37). The incoming messages are distributed from this single thread to the threads representing equipment instances (heuristic 38).

A more extensive description of the tss concurrency design can be found in section A.5.4.

Heuristics Overview

Heuristic 28: Use address spaces as failure containment units. Recovery from failure is realised within an address space.

Heuristic 29: Use address spaces to design for deployability. The freedom to relocate functionality to different processors depends on the absence of common data between address spaces.

Heuristic 30: Consider the use of a thread on the architectural level.

Heuristic 31: An overview of all threads should be given in a global concurrency design.

Heuristic 32: Mirror independent behaviour of application domain objects by separate logical threads.

Heuristic 33: Use a separate thread to handle an external connection or external messages.

Heuristic 34: Cluster all functionality which is activated via object interaction by the external connection or messages into the thread.

Heuristic 35: Use a separate thread for the interaction of a user with the system.
Heuristic 36: Represent the receiving direction of an external channel or bus by its own thread.

Heuristic 37: Message sending over an external channel is done on the budget of the sending thread.

Heuristic 38: Refine the design of a separate thread per bus to have thread instances per connected equipment instance to the bus.

Heuristic 39: Let internal consistency have priority over external reaction.

Heuristic 40: Give operational tasks priority over background tasks.

Heuristic 41: Use separate thread per different priority.

Heuristic 42: Use a separate thread per cluster of objects with given priority.

Heuristic 43: Split logical threads up into physical threads per address space.