Part III
Chapter 17

Technology

17.1 Introduction

The essence of all approaches to system design is abstraction: to deliberately ignore some of the details of a system design to make designs and system descriptions manageable. The detail that is deliberately omitted is not unimportant, however. It is often necessary to collect the detail together so it may be separately addressed.

One of the kinds of detail that is frequently omitted is information about how a system of connected objects is to be implemented or supported. Often this detail is similar or even the same for all objects in a particular system. A designer will then need to instruct the implementer only once as to how to implement the system. Such detail can be extracted from each of the objects in a system and represented by a single object. This object is then referred to as the **infrastructure** that supports the objects.

The infrastructure can now be separately described and decomposed and even implemented. This is where the concept of a platform stems from. In decomposing the infrastructure, several components are distinguishable. Processing components such as **information processors**, communication components such as **ports** and **messages**, and **memory** components can be part of the infrastructure and are described in this chapter†.

The concept of resource management is also important (scheduling, virtual memory and virtual processors). These concepts have been described in Chapter 12 on Resource management.

17.2 Reference section

In the following manual pages the concepts of infrastructure, information processor, message, and memory have been described.

The description of the concept of port is reserved for future editions.

† The current descriptions are largely intended as place holders.

NAME

Infrastructure

PURPOSE

An infrastructure is an object that represents some of the detail how all other objects in a system are to be implemented.

SYNOPSIS

Object diagrams are used to describe systems. The diagrams do n always offer the implementer sufficient information about the wa in which a system should be implemented. A designer may colle some of this detail in one object, called an infrastructure. Tl infrastructure is said to support the objects in the system design.

CANONICAL FORM

The canonical form is shown in Figure 17.1. An infrastructu object is connected to all other objects in a system design. Tl implementation details of the connections between the objects ha been collected in the infrastructure. The infrastructure shou normally be drawn as an object as in Figure 17.1, but a horizont line separating objects outside the infrastructure from objec within it is also permitted as in Figure 17.2.

Figure 17.1 The infrastructure: canonical form

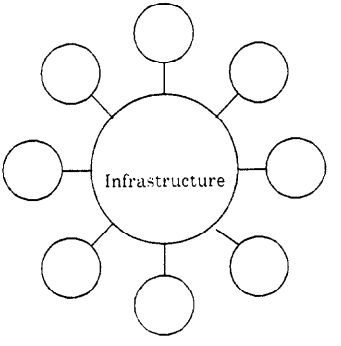
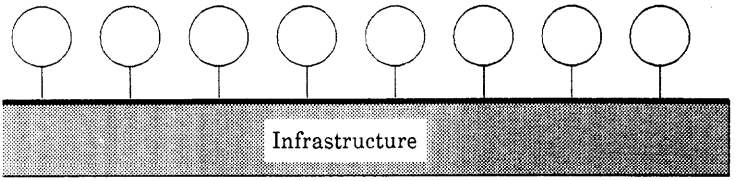


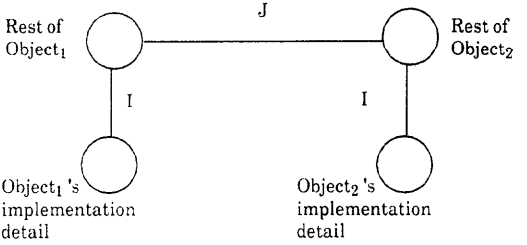
Figure 17.2 The infrastructure: alternative canonical form



SPECIFICATION

Two objects, $Object_1$ and $Object_2$ interact using interaction alphabet J. $Object_1$ may be decomposed into an object that represents some of $Object_1$'s implementation detail and an object that represents the rest of the original $Object_1$. These objects interact using interaction alphabet I. Similarly $Object_2$ may be so decomposed, as illustrated in Figure 17.3.

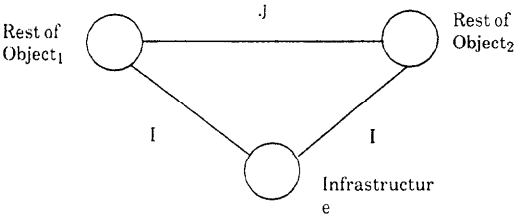
Figure 17.3 Decomposing two interacting objects



The decomposition of $Object_1$ and $Object_2$ must be done such that the specification of the implementation detail of $Object_1$ and that of $Object_2$ are the same. In that case $Object_1$'s and $Object_2$'s implementation detail can be merged into a single object: the Infrastructure. The infrastructure places *exactly the same* constraints on $Object_1$ and $Object_2$, as illustrated in Figure 17.4. $Object_1$ and $Object_2$ in Figure 17.4 are constrained by the infrastructure such that interactions between the objects and the

infrastructure are defined by the alphabet I. If other objects wer added to the system, they too would be subject to the same set o constraints: I.

Figure 17.4 Merging implementation detail into the Infrastructure



Each of the objects may place their own constraints on th interactions with the Infrastructure. Suppose that Rest of Object imposes a constraint I_1 and Rest of Object$_2$ a constraint I_2. Furthe suppose that the infrastructure imposes a constraint I_i.

Rest of Object$_1$ and Rest of Object$_2$ may now not be able t communicate via the infrastructure if

$I_1 \cap I_i \cap I_2 = \{\}$

or if the infrastructure provides some independence that can only b determined from further knowledge of the relationship it maintain between the symbols that appear at its interfaces.

REALIZABILITY ISSUES

SEE ALSO

FUTURE DIRECTIONS

NAME

Information processor

PURPOSE

To provide an interpreter for the instructions in the rest of the object.

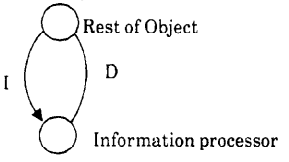
SYNOPSIS

When the information processor is extracted from an object, so that it may appear in an infrastructure, then the active part of the object has been removed. The part that is left behind is a description of the activity that the object as a whole represented. The information processor can thus be seen as an interpreter, that can interpret the description of the activity. In doing so the interpreter, together with the description behave as the object described by the description.

CANONICAL FORM

The canonical form is illustrated in Figure 17.5. The information processor is connected to the rest of the object with two connections. One of the interfaces of the information processor is a directed interface. The alphabets in use between the Rest of the object and the information processor are an alphabet of instructions (I) and an alphabet of data (D).

Figure 17.5 An information processor as an infrastructure for a single object



SPECIFICATION

REALIZABILITY ISSUES

The information processor must be able to interpret the description of the activity held in the Rest of Object.

SEE ALSO

INFRASTRUCTURE
INTERPRETER
OBJECT SPECIFICATION

FUTURE DIRECTIONS

It is important to ensure that the information processor can deal with different forms of instruction and data. It is anticipated that the present canonical form is just one of several possible forms. A dependency between the possible canonical forms and the structure of instructions and data is expected. It is not clear whether the structures of instructions and data should be elaborated in this general model.

NAME

Message

PURPOSE

To provide a means for interaction between objects.

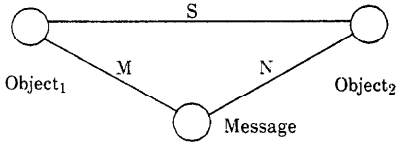
SYNOPSIS

A message is an object. It may be accessed by several objects that take place in an interaction. The objects may exchange information so as to synchronize access to the message, in case it does not itself support any concurrency. The information that is exchanged and the pattern of access to the message may be varied to provide different styles of interaction.

CANONICAL FORM

The canonical form is illustrated in Figure 17.6. Two objects interact with one another and a third object, the message is used as the medium for some of these interactions.

Figure 17.6 A message as a support for communications



SPECIFICATION

REALIZABILITY ISSUES

A message passing paradigm can be realized by allowing only a single binding between the message and an object. By changing the bindings message passing semantics are achieved.

If both objects can have access to the Message, then message sharing semantics are imposed.

The two interfaces to the message may differ in many respects. If the alphabets are different, the representation of the message could

have been changed, or different access right could be assigned to the different interfaces.

SEE ALSO

MEMORY

FUTURE DIRECTIONS

Provision of a specification. A proper evaluation of the different message passing semantics, in terms of the model presented here is possible and may be included in future versions.

NAME

Memory

PURPOSE

To provide an object that allows its interactions to persist over time.

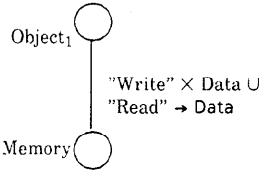
SYNOPSIS

A memory object is an object that can interact with a composite symbol taken from one of two alphabets. The alphabets are related, such that an interaction taken from one alphabet, followed by an interaction taken from another gives the impression that the first interaction has had some lasting effect on the memory object.

CANONICAL FORM

Memory is represented by an object in object diagrams. Figure 17.7 illustrates the canonical form, in which a memory is connected to an object.

Figure 17.7 An object and a memory



SPECIFICATION

The alphabet is composed of two constituent alphabets

"Write" × Data

and

"Read" → Data

The relation that is maintained is summarized by

Write following a write - any relation:
{(("Write", 0), ("Write", 0)), (("Write", 0), ("Write", 1)),
(("Write", 1), ("Write", 0)), (("Write", 1), ("Write", 1))}

Write following a read - any relation:
{(("Read", 0), ("Write", 0)), (("Read", 0), ("Write", 1)),
(("Read", 1), ("Write", 0)), (("Read", 1), ("Write", 1))}

Read following a write - the data is identical:
{(("Write", 0), ("Read", 0)), (("Write", 1), ("Read", 1))}

Read following a read - the data is identical:
{(("Read", 0), ("Read", 0)), (("Read", 1), ("Read", 1))}

REALIZABILITY ISSUES

An underlying mechanism is assumed that can retain a symbol over time. Sometimes energy is required to make a symbol persist in this manner.

SEE ALSO

INFRASTRUCTURE

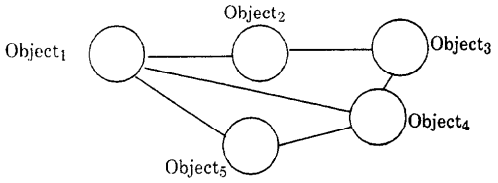
FUTURE DIRECTIONS

The specification should be decoupled from the example of
Data = {0,1}
The entry should be aligned with the entries in the chapter on indefinite interactions.

17.3 Examples

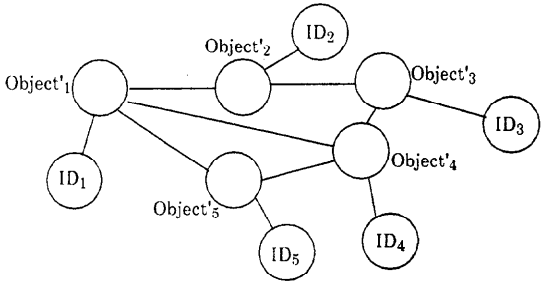
The system of interconnected objects shown in Figure 17.8 contains little detail about how the objects can exist.

Figure 17.8 A model of a system contains little information about how the objects can exist.



A designer will need to instruct the implementer how to implement the system. Some of this detail can be abstracted from the objects in the system design, as illustrated in Figure 17.9. The objects marked ID_n represent some of the implementation detail of the object to which they are connected. The objects that remain after the extraction of implementation detail are slightly different from the ones in Figure 17.8. One of the differences is that they now have two interfaces, instead of one.

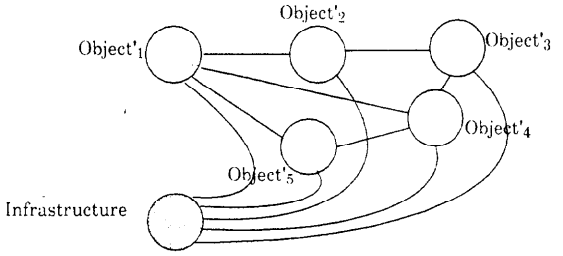
Figure 17.9 Collecting implementation detail in the infrastructure object.



The implementation details of the different objects can be collected in a single object that is connected to all objects of the system design as shown in Figure 17.10. If the implementation detail of the various objects were the same or very nearly the same, then the new object is called an **infrastructure**.

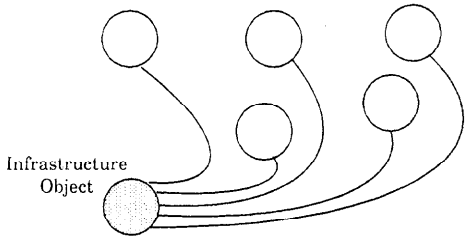
Often the infrastructure includes the implementation detail of the connections between the objects in the system design. In that case all

Figure 17.10 Collecting implementation detail in the infrastructure.



connections between the objects in the system design are routed through th infrastructure, as illustrated in Figure 17.11.

Figure 17.11 The infrastructure may include the details of the connections between objects in the system design

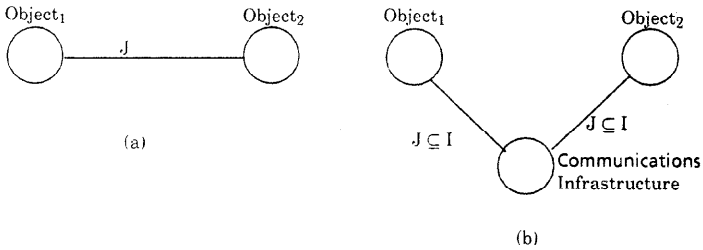


Once the infrastructure has been abstracted away from the rest of the systen design it will no longer be drawn in system models. If the connection between objects were subsumed in the infrastructure they will have to b redrawn and it will then not be obvious from the figure that the now invisibl infrastructure still places constraints on the interactions between the object in the system model.

Communications infrastructure

The implementation detail of the connections between objects may also b incorporated in the infrastructure. Two objects, $Object_1$ and $Object_2$ that ar connected as illustrated in Figure 17.12(a), may have their connectio realized by a communications infrastructure, as shown in Figure 17.12(b).

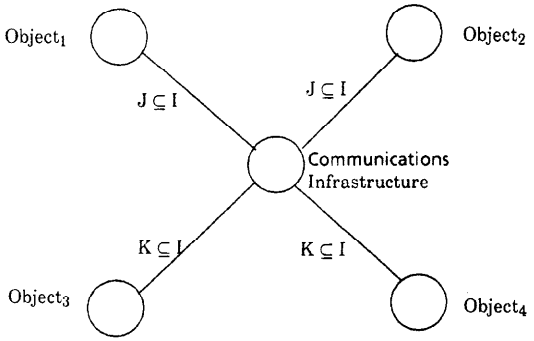
Figure 17.12 The communications infrastructure supports interactions



Because the infrastructure constrains all objects in the same way, the interaction between $Object_1$ and $Object_2$ must be expressed in terms of constraints on interactions between the objects and the Infrastructure. For the purpose of interaction, $Object_1$ and $Object_2$ place a constraint J on their interactions with the Infrastructure, such that $J \subseteq I$.

A communications infrastructure frequently supports interactions between several pairs of objects. In Figure 17.13 the infrastructure also supports interactions between $Object_3$ and $Object_4$. $Object_3$ and $Object_4$ place a constraint K on their interactions with the Infrastructure, such that $K \subseteq I$.

Figure 17.13 The communications infrastructure supports different interactions



In the terminology of the ISO Reference Model for Open Systems Interconnection [IS7498], I is the set of constraints called (N)-Service, and J and K are the constraints called (N+1)-Protocol. The (N)-Service can support multiple protocols, J and K in this example.

Secure infrastructures: separation

If $J \cap K = \{\}$ and if the Infrastructure does not maintain any relation between symbols in J and symbols in K, then $Object_1$ or $Object_2$ cannot interact with either $Object_3$ or $Object_4$. The interactions between $Object_1$ and $Object_2$ and between $Object_3$ and $Object_4$ respectively have been effectively separated and two systems have been created, supported by the same infrastructure. The separation may be achieved in time, in space or by choosing different symbols (events).

It is interesting to observe that the supported objects are the ones that have imposed the constraints on their interactions with the Infrastructure, thus creating the separation. In a secure system this would be insufficient as one of the untrusted supported objects could modify the constraints that it imposes on the interactions, thus breaking down the separation.

17.4 References

[IS7498] International Standards Organization, *Information Processing Systems - Open Systems Interconnection - Basic Reference Model*, ISO, 1983.