# Designing Software Components

Author: Richard Veryard
Version: February 10h 1999

richard@veryard.com
http://www.veryard.com

For more information about SCIPIO, please contact the SCIPIO Consortium.

info@scipio.org
http://www.scipio.org

# Preface

## Purpose of document

- ➢ To state the requirements for an effective and successful design approach.
- ➢ To describe the SCIPIO approach to designing federated organizations and networks.
- ➢ To describe the SCIPIO approach to designing distributed software systems and software components.

## Audience

This document is intended for the practitioners of organizational and systems design, and for people who want to participate in the specification of requirements, especially where these systems are to be federated between multiple organizations or distributed across decentralised organizations.

## Questions for reader

The text is interspersed with open questions, which the reader is invited to consider.

**Q** How will software engineering practices need to change, if they are to accommodate Component-Based Development?

**Q** How soon do you expect these changes to be widespread within the software industry?

## Acknowledgements

This material draws on research carried out under the Enterprise Computing Project. This work benefited from the participation of John Dobson, David Iggulden, Rob van der Linden, Ian Macdonald and Sally Jack.

# Introduction

## How do CBD and ODP affect Design?

In an open distributed processing world, design is different in two ways:

➢ Different resulting systems

➢ Different design process - design process is itself a distributed system

What are the methodological issues arising from these differences?

| | |
|---|---|
| System specification | Designers need to specify **flexible** yet **coordinated** solutions. Prior to the use of open distributed processing technology, flexibility and integration were often opposed goals: the more you had of one, the less you had of the other. Open distributed processing helps the designer reconcile these goals. |
| Object service specification | The generic architecture of an open distributed processing system comprises objects providing services. The **identity** of any such object or service depends not only on what the object actually does, but also how this is presented or described, either by the object itself or by traders/brokers on its behalf. This aspect of the specification of these objects and services is a key design issue. |
| | The **viability** of an object depends on its ability to provide required services under a range of likely scenarios. For short-term viability, the object must generate enough value to support the continued availability of its services. For longer-term viability, the object must generate enough value to support adaptation of the object to changing requirements. |
| Design process | 'Open' and 'distributed' are not only characteristics of the systems that are being developed and implemented in an open distributed processing world, they are also characteristics of the design process itself, which typically lacks both a fixed statement of requirements and a central design authority. Coordination between multiple designers is focused on the negotiation of interfaces. We may need to address in new ways the quality of the design process, as well as of the designed product. |

## Requirements for a Design Method

### Quality of design

A design judgement needs to be evaluated at three levels:

➢ at the level of the system being designed, where the design should represent a good pattern, and embody good design values and practices

- at the supersystem level, where the designed system should contribute in a positive way to some broader system
- at the subsystem level, where the design should create an integrating structure/environment in which the lower-level detail can be worked out

A good design is a collection of good design judgements. We can talk about the overall value of a design - this is precisely what is required for a **business case**. A good design judgement increases the overall value of the design, but this overall value cannot be distributed arithmetically between the judgements that make up the design.

The requirements for a design method can be brought under the umbrella of quality. Quality relates to everything else.

## Quality of design as end-product

A socio-technical system needs to be evaluated in three ways:

1. From a combined socio-technical perspective
2. From a social perspective
3. From a technical perspective

In other words, some of the quality characteristics apply to either the social or technical aspects of the system, while some of the quality characteristics apply to both at once.

From a combined perspective, a systems design should be:

**Well-modularized** - i.e. defining modules that have maximum cohesion and minimum connection

**Measurable and testable** - i.e. the non-functional requirements should be objective and (if possible) quantified.

**Well-connected** - i.e. consistent with architectures and policies, and with an appropriate level of integration with other related systems

From a social perspective, a systems design should be:

**Usable** - i.e. fitting into the intended business environment, and providing useful support to the user in carrying out his/her job

**User-friendly** - i.e. with computer functionality matching the structure of the business operations, so that the system works the way the user thinks

From a technical perspective, a systems design should be:

**Implementable** - i.e. technically feasible on the chosen target platform

**Efficient** - i.e. with an adequate balance between speed, throughput and cost-effective use of computer resources.

## Quality of design process

We can also identify the process quality criteria for the design methodology:

**Correctness -** i.e. no design errors found during testing or operations

**Maximum reuse** of design components

**Minimum 'thrashing' -** i.e. going round in circles before agreement can be reached

**Low maintenance costs of system** (other than owing to changes in model)

**Maximum learning** for participants and entire organization

**Efficient & effective -** i.e. achieving a good result with a reasonable expenditure of time and energy

# Design Process

## Rationale

The rationale for revisiting the design topic is to consider how distribution affects the classical design process. A basic characteristic of the classical design process whether of the waterfall, spiral or incremental development variety is that systems were generally produced under a single design authority. With distribution this stringent aspect is relaxed as components of a distributed system may be designed by different hands at different times and with different assumptions and constraints.

Another characteristic of distributed systems is the potential for reuse of components and the manufacture of components from templates as required. The design process therefore needs to bear all these aspects in mind especially the tendency to construct systems from pieces of low-grained functionality with the extra information requirements that are entailed. These information requirements are pointed up by the discussions on design repositories

## Who is the designer?

We should think of the designers as anyone doing design, and not only professional designers (technologists). These people may well be managers and management consultants, rather than Designers with a capital D.

## Nature of design

The potential scope of design ranges from the complete business to the design of individual modules or activities.

The question of the design (or rather re-design) of the business or enterprise arises through economic, commercial and regulatory pressures on the one hand and on the other by the opportunities provided by technological innovations. SCIPIO models provide the possibilities of animation so that the stakeholders may make choices and state preferences amongst alternatives. By separating the functional, economic and change issues these choices are made clearer and are supported by a number of analytical disciplines.

Although the design process is generally limited in scope to the design of software components it is necessary to consider to periodically question the fitness of purpose. This may be done by some kind of incremental development or prototyping where validation of the design may be frequently tested. In a teamwork environment then the design may be worked at the same as the production and support processes and certainly with the co-operation of the various users and stakeholders of the product.

## Design contract

For our purposes, the notion of **contract** is a codification of agreement between known parties for the development of a design. We see this as a particularly crucial factor: the biggest driver of the design process is typically the location of contractual boundaries.

In any particular context the design process will be driven by the boundaries of the location of the contract in place. This is effect saying that there has to some prior *agreement* between the parties involved which determines the extent of the boundaries and allows them to be renegotiated and redrawn.

It should be noted that the telecoms world use the word ‘contract’ to refer to a specification, or to a proforma agreement with arbitrary future parties. We want to call this something else (e.g. a **service specification**).

### Technical environment for design process

This section discusses the information and communications requirements of the designers themselves, and the extent to which appropriate socio-technical systems may satisfy these requirements.

Having derived as complete a description of a software component as possible, it is necessary to ensure that this description becomes available to all those who need to have access to it. In general this means everyone who is in some way legally interested in the component and its interactions. Maintenance personnel will need to know about the components they are maintaining. Designers of new features will require access to descriptions of the components already in the system. They will also be extending descriptions of components that make up a new feature.

At times they will want to query the set of descriptions to see if a component with desired characteristics has already been developed. This raises the question of what kinds of query should be supported. Using formal languages is probably not a good idea: writing the query would take as long as writing the specification, syntactically different descriptions may be semantically the same, and there is no such thing as a close match between two formal specifications.

Designers will also want to get access to completed software components, or to organizations that are proposing to build such components, once they have identified the required characteristics. In open distributed computing systems this can be provided in one of two ways: either a reference to a construction service (a factory), or a reference to an actual instance is passed back to the party that invoked the query. A construction service is able to construct a software component with particular characteristics.

To support these and other design, development and maintenance activities, on a large scale, the information service must itself be built as large-scale service in a wide area communications network. (The structuring principles discussed later in this paper should of course be applied to such a system.) Such a service needs to span organizational boundaries and be able to store a wide variety of information. In the future this information will also be used to drive application transformation and configuration tools.

## Design Negotiation

This section, on the generic design process, is meant to highlight the issues in design in the world of distributed systems and to be of use in understanding more general development issues especially those of construction and migration. The main issues relate to conflicts arising because of parts of the system deriving from different sources and under different regimes and the consequent need for maintaining and making available specifications and reasons for particular design choices.

One of the key notions of open distributed processing is the capability to negotiate an **interface** at execution time. Technical design considerations (beyond the scope of this document) are involved in designing systems with such capability. This leads inevitably to recognizing the importance of negotiating the interfaces within the design process itself.

## Difficulties of designing in an open distributed processing world

Several obstacles have to be overcome specifically when software components and information about them are passed across organizational boundaries. The problems are partly technical, partly human, and partly a consequence of the way in software is procured.

| | |
|---|---|
| Technical problems | The difficulty of developing reusable components.<br><br>The difficulty of a priori deciding whether a component is actually reusable.<br><br>The absence of any effective catalogues, which can be searched effectively. |
| Human problems | Software reuse involves a kind of de-skilling of what are seen as highly skilled developers [Wood & Sommerville 1988]. |
| Economic problems | Deciding who owns a complex object which has been built by X for Y, using components developed by Z. |

There are many other differences on either side of the organizational boundary:

- No common purpose; different organizational entities often lack a common purpose, specially if boundaries lie between dissimilar industries.
- No trust; there is an inherent lack of trust between different organizational entities. This is most vividly reflected in the uncertainties about the quality of a software component that is obtained from another domain.
- No common economic framework; where components are provided by one organization and used in another there may be a need for payment. There is no clearly defined market for software components or information about them, neither is there a clear idea of how to charge for either use or ownership.
- No common legal framework; software components may be of inferior quality, or the information about a component may be inaccurate. It is unclear what if any legal framework applies in these cases.
- No common meaning and representation; there is little agreement about the languages which should be used for the description of software components. Misunderstandings can arise despite the presence of comprehensive descriptions (some aspects can only be described in natural language for instance; formal languages need a common context for agreed interpretation).
- No common quality standards; the agreed measures for software component quality have been derived from those suitable for hardware components (MTBF, MTTF etc.). Software quality of service is often better expressed in application dependent terms, such as mission time (the time during which a component is to stay operational with a precisely defined probability of failure for instance).

- No support for distribution; this is a problem which requires the application of distributed processing technologies.

The above organizational issues should be solved within the business process of producing new features for telecommunications systems. They are part of a business process re-engineeringactivity in which several telecommunications companies may wish to become engaged.

## Teamwork and design coordination

Various teamwork disciplines are described in the literature such as Total Quality Management, Concurrent Engineering and the Virtual Enterprise. Such disciplines need particular tools which themselves need to be designed. The need for change management in such an environment with various players puts a premium on distributed databases, groupware, multimedia and graphical interfaces and display.

Because of the likely distribution of the tools there is the need of support for replication, replication and concurrency. This is in addition to the question of recording and display of design information. Team design activity needs to use such support services as lookup, computation, communication, negotiation, decision and storage (or archiving).

The design process as embodied in tools and procedures and in the statements and guidelines of best practice must tie choices being made in the socio-technical design with formulation of the business case, and the procedures and models of change, commissioning and decommissioning. Models and tools for configuration management may provide a hint of how to carry this out. In other words there has to be some notion or mechanism for establishing the consistency of the models being developed. This will probably be done by repositories of designs and design choices in all three areas.

# SCIPIO Design Guidelines

## Design for Flexibility

Designers need to specify **flexible** yet **integrated** solutions. Prior to the use of open distributed processing technology, flexibility and integration were often opposed goals: the more you had of one, the less you had of the other. Open distributed processing helps the designer reconcile these goals.

Flexibility is promoted by the following design precepts, which are supported by SCIPIO modelling techniques:

| | |
|---|---|
| Design systems by composition from smaller units. | ➢ The SCIPIO model shows (perhaps many) ways in which an enterprise may be decomposed.<br>➢ The SCIPIO model also shows (perhaps many) ways in which the parts of an enterprise may gain by being connected. |
| Decentralize control subsystems | ➢ Bottlenecks of command and control, in which opportunities for massively parallel operations are frustrated by central (decision-making) authority and lack of local empowerment. |
| Improve the overall robustness of a system by improving the robustness of individual components. | ➢ The SCIPIO model may indicate particular areas of uncertainty where early commitment may be unwise. |
| Use responsibility clustering to divide a system into subsystems. | ➢ The SCIPIO model should tell us enough about the intentions of the various agents to indicate what are the important states (conditions) for which some agent needs to be responsible.<br>➢ The enterprise model should also tell us which agent (if any) is currently responsible for which states.<br>➢ Where an agent is responsible for two contradicting conditions p and q, convert this into a responsibility for maintaining a proper balance between p and q.<br>➢ Where one agent is responsible for p and another agent is responsible for q, and there is a positive or negative correlation between p and q, consider making one agent responsible for both conditions (and for the balance between them).<br>➢ Where one agent is responsible for too many different things, look for a way of subdividing these responsibilities into separate agents, with low coupling between them. |

How much flexibility is required? Since flexibility can be thought of as the adaptability of a solution to changing requirements, flexibility itself is a second-order requirement. This is an area for further research.

## Design for Coordination

Coordination is promoted by the following design precepts, which are supported by the technique of enterprise modelling:

| | |
|---|---|
| Use the mechanisms, concepts and techniques of open distributed processing to handle data in different formats, with different (but overlapping) scope, and with different levels of granularity and detail.<br><br>Allow the user to view data from heterogeneous sources in the same format, for ease of comparison.<br><br>Allow the user also to view the data in the original format, since reformatting may sometimes cause information loss or distortion. | ➢ Information provision can be modelled as a service. The service chain can be modelled, to identify the ultimate source of information. This helps verify that two apparently independent items of information really are independent.<br><br>➢ Analyse what difference an item of information might make. Is there (for example) a process whose outcome will be significantly affected? What is the risk of errors and omissions? |
| Federate, as a cheap way of reducing interaction distance between heterogeneous systems. | ➢ Determine actual interaction distance by analysing relevant exchanges. Determine desired interaction distance by analysing relevant intentions. |
| Ensure conversation parties have agency over the related policies; for example, a highly distributed system will lead to lack of centralized control over information accessibility and may therefore weaken a chain-of-command structure. | ➢ Examine the responsibilities of the parties to conversations in the conversation hierarchy<br><br>➢ Responsibility for **making** policy (and other intentions) versus responsibility for **executing** policy. |

## Design for Identity

The generic architecture of an ODP system comprises **objects** providing **services**. Specification of these objects / services is a key design task.

The **identity** of any such object depends not only on what the object actually does, but also how this is presented or described, either by the object itself or by traders/brokers on its behalf.

Design for identity therefore implies the creation of **meaningful descriptions**.

We have identified the following design precepts:

| | |
|---|---|
| Build computer systems to facilitate worker learning. Build data systems that allow or encourage creative changes of use. Make models and diagrams of the system's structure and workings accessible to the users. | ➢ The SCIPIO model may be used as a presentation tool, to communicate systems to their users. |
| Provide mechanisms to allow the user to select the level of visibility or transparency of each technical matter. | ➢ Each technical matter may be identified as the responsibility of a given agent (or role). We can analyse how effectively/efficiently these responsibilities are currently fulfilled.<br>➢ Alternatively, technical expertise may be identified as a resource, embedded either in people or in technical devices. We can analyse how effectively/efficiently these resources are currently deployed. |
| To build a Chinese wall, it is not enough merely to withdraw formal communication mechanisms between departments. It will often be necessary to actively discourage or forbid contact. | ➢ |
| Overcome/counteract the tendency to hide the identity of electronically mediated co-workers. Provide mechanisms for personal contact between agents. | ➢ The SCIPIO model may be used to identify those relationships which are too important to be mediated solely by electronic means. |

## Design for Viability

The generic architecture of an ODP system comprises **objects** providing **services**. Specification of these objects / services is a key design task.

The **viability** of an object depends on its ability to provide required services under a range of likely scenarios. For short-term viability, the object must generate enough value to support the continued **availability** of its services. For longer-term viability, the object must generate enough value to support **adaptation** of the object to changing requirements.

Design for viability therefore implies the creation of **added value**.

We have identified the following design precepts:

| | |
|---|---|
| Design for diversity | ➢ The SCIPIO model indicates where multiple agents may provide variant services. |
| Analyse attitudes to current systems to determine any hysteresis. Evaluate the effect of this 'hysteresis' on the proposed socio-technical system.<br><br>Design components of the system so that a 'history of past decisions' is maintained. | ➢ Look for delays between cause and effect. This means we should include some cause-effect modelling in the modelling language.<br><br>➢ Look for negative feedback loops.<br><br>➢ Look for a process for discovering and dealing with negative feedback loops. This means we should model second-order processes (i.e. processes about processes). |
| Guarantees must be formulated and incorporated into the specifications of components and the consequences of failure defined. | ➢ The SCIPIO model must incorporate a failure model. The consequences of failure will allow the importance of certain actions to be assessed and modal distinctions made between the necessary and the possible. |
| Expectations provide a statement of 'ideal' requirements. The design process must allow suitable trade-offs to be made between the ideal and the feasible. | ➢ The changes sought by a requirements statement should satisfy two main criteria:<br><br>➢ They should be **systemically desirable**. They can be incorporated within an overall system design.<br><br>➢ They should be **culturally feasible**. They have to be regarded as meaningful by the people and organizational culture in question. |
| Agents should have the necessary authorization and capability tokens to enable them to access the information resources they require to fulfil their responsibilities and discharge their obligations. | ➢ Agents cannot independently transfer their **responsibilities** to other agents, but they can transfer their **obligations**. The result of this process is the establishment of a new responsibility relationship between the (pairs of) agents involved. A chain of responsibility relationships can be created as obligations pass from one agent to another. Within each responsibility relationship both agents have a responsibility for the same state of affairs, although their obligations differ.<br><br>➢ The first agent acquires a new obligation of a structural nature to do whatever is appropriate with respect to the other agent in order to fulfil his responsibility, such as directing, supervising, monitoring and suchlike of the other agent. The other agent also acquires an obligation of a complementary nature to be directed, to be supervised or whatever. (These are termed **structural obligations**. They may be implicit in the hierarchical structure of the organization rather than as a result of explicit delegation.) |

## References

**RM-ODP** - the reference model for Open Distributed Processing.
The official website for RM-ODP is http://www.iso.ch:8000/RM-ODP/
There are some key papers downloadable from the ANSA website http://www.ansa.co.uk but the website itself is so badly signposted that you would be unlikely to find what you wanted. See instead http://www.dstc.edu.au/AU/research_news/odp/ref_model/

**SCIPIO**. For more information on SCIPIO, including a detailed task structure, please see the SCIPIO website at http://www.scipio.org/

Wood, M., and I. Sommerville, An information retrieval system for software components, Software Engineering Journal, pp.198-207, Sept. 1988.