SYSTEM ARCHITECTURE

Project Number:	**TR 1101**
Project Title:	**CONVERGE**
Deliverable Type:	**PU**

Deliverable Number:	**DSA2.3**
Contractual Date of Delivery:	**10/02/98**
Actual Date of Delivery:	**10/02/98**
Title of Deliverable:	**Guidelines for the Development and Assessment of Intelligent Transport System Architectures**
Work-Package:	**SA2&3**
Nature of the Deliverable:	**Report**
Author(s):	**P H Jesty, J-F Gaillet, J Giezen, G Franco, I Leighton, H-J Schultz**

Abstract:

A system architecture provides the basis for a working and workable system. These combined guidelines provide advice for Transport Telematics projects on the development and assessment of their system architectures. They distinguish between three levels of architecture which together form a system architecture, and show how the development should proceed once the user needs are known. The issues covered include system characteristics and system requirements; reference models, including those created by SATIN for urban, inter-urban and in-vehicle systems; and enterprise functional, information, physical and communications architectures. Checklists of the issues to be considered are provided, and a number of reviews and analyses for the assessment of an architecture are described.

Keyword List:

Assessment, Functional Architecture, Logical Model, Physical Architecture, Physical Model, Reference Models, System Architecture, System Requirements

CONVERGE
TRANSPORT TELEMATICS
SUPPORT & CONSENSUS

USERS PROVIDERS TELEMATICS APPLICATIONS

SYSTEM ARCHITECTURE

Guidelines for the Development and Assessment of Intelligent Transport System Architectures

February - 98

Issue : 1.0

CONVERGE System Architecture Partners:

Atkins Wootton Jeffreys (co-ordinator)

ERTICO

Heusch/Boesefeldt GmbH

ISIS

MIZAR Automazione S.p.A.

The University of Leeds

TNO-TPD

Table of Contents

1. EXECUTIVE SUMMARY 7

2. LIST OF ABREVIATIONS 8

3. INTRODUCTION 9

3.1 Purpose of the document 9

3.2 Structure of the document 9

3.3 Who we are 10
3.3.1 Our aims 10

4. DEFINITION OF SYSTEM ARCHITECTURE: 11

4.1 Introduction 11

4.2 Working and Workable Systems 11
4.2.1 Feedback from Experience 12

4.3 Levels of Architecture 14
4.3.1 Infrastructure 16

4.4 Further Implications 16

4.5 What is Architecture Assessment 17

5. SYSTEM ARCHITECTURE ISSUES 19

5.1 Complexity Management 19
5.1.1 Connections 19
5.1.2 Abstraction 19

5.2 Consistency 20

5.3 Flexibility 20

5.4 Maintainability 21

5.5 Scale enlargement 21

5.6 Emergent Properties 22
5.6.1 Controllability 23

5.7 Multi-disciplinarity 23
5.7.1 Hardware and Software 23

5.8 Verification &Validation 24
5.8.1 Testability 24

6. SYSTEM ARCHITECTURE DEVELOPMENT 26

6.1 User Needs **29**

6.2 System Concept **29**
6.2.1 Vision Statement 30
6.2.2 Identification of the Users 30
6.2.3 Mission Statement 31

6.3 System Characteristics **31**

6.4 System Requirements **31**
6.4.1 User Needs vs. System Characteristics vs. System Requirements vs. System Specifications 33

6.5 System Boundary **34**
6.5.1 Preliminary Safety Analysis 35

6.6 Level 2 and 3 System Properties - Reference Models **35**
6.6.1 Responsibility 37
6.6.2 Zones of Autonomy 38
6.6.3 Production of a Layered Reference Model 38

6.7 Level 1 System Structure - Functional Issues **39**
6.7.1 Enterprise Architecture 40
6.7.2 Logical Model 40
6.7.3 Object-Orientation 41
6.7.4 Physical Model 42

6.8 Level 1 System Structure - Behavioural Issues **43**
6.8.1 Usability 43
6.8.2 Risk/hazard Analysis 44

6.9 Level 0 - Design **45**
6.9.1 Standardisation 45

6.10 Legacy Systems and Migration **45**
6.10.1 Compatibility 46
6.10.2 New system architecture with interface bridge 46

7. THE ARCHITECTURE ASSESSMENT PROCESS 48

7.1 Objectives **48**

7.2 Overview **49**

7.3 Project System Architecture Assessment Report **49**

7.4 User Needs System Concept **50**
7.4.1 Vision Statement 50
7.4.2 Identification of the Users 50
7.4.3 Mission Statement 50
7.4.4 The System Boundary 50

7.5 System Concept System Characteristics **50**

7.6 System Characteristics System Requirements **51**
7.6.1 System Context Diagram 51

7.7 System Requirements Reference Models **52**
7.7.1 Analysis of Goal-Oriented Functions 52
7.7.2 Review of Functionality and Behaviour 52

7.8 Reference Models Level 1 Architectures **53**
7.8.1 Functional Architecture 53
7.8.2 Control Architecture 54
7.8.3 Information Architecture 54
7.8.4 Enterprise Architecture 55
7.8.5 Physical Architecture 55
7.8.6 Communication Architecture 55

7.9 Checks for Consistency **56**

7.10 Benefit Analysis **56**

7.11 Cost-Benefit Analysis **56**
7.11.1 Market Analysis 59

7.12 Functional Analysis **59**

8. REFERENCES & BIBLIOGRAPHY **61**

APPENDIX A PROPOSED SYSTEM ARCHITECTURE DELIVERABLE CONTENTS LIST

APPENDIX B PROPOSED SYSTEM ARCHITECTURE ASSESSMENT DELIVERABLE CONTENTS LIST

APPENDIX C LEVELS OF ARCHITECTURE

APPENDIX D SYSTEM CHARACTERISTICS

APPENDIX E CONTEXT REQUIREMENTS

APPENDIX F FUNCTIONAL REQUIREMENTS

APPENDIX G NON-FUNCTIONAL REQUIREMENTS

APPENDIX H PRELIMINARY SAFETY ANALYSIS

APPENDIX I SYSTEM ARCHITECTURE: THE DEVELOPMENT OF THE REFERENCE MODEL

APPENDIX J SATIN REFERENCE MODELS

APPENDIX K ENTERPRISE ARCHITECTURE

APPENDIX L FUNCTIONAL ARCHITECTURE

APPENDIX M ODP AND THESE GUIDELINES

APPENDIX N INFORMATION ARCHITECTURE

APPENDIX O ASSESSMENT BY A REVIEWER

APPENDIX P THE PASSPORT CROSS

APPENDIX Q BEHAVIOUR ANALYSIS TOPICS

APPENDIX R THE CONVERGE ANALYSIS TOOL

APPENDIX S SATIN DOCUMENTS

APPENDIX T WHISPER CASE STUDY

APPENDIX U BACKGROUND INFORMATION

1. EXECUTIVE SUMMARY

- These guidelines provide advice for Transport Telematics projects on the development and assessment of their system architectures, and suggestions are made as to the possible contents of their System Architecture Deliverables.
- All systems have an architecture, even though most developers do not yet explicitly write it down. This architecture provides the structure around which the system is developed. Once the structure has been defined, either implicitly or explicitly, it is then usually very difficult and expensive to change it at a later date.
- A system architecture provides a stable basis for a working and workable system for the benefit of the entire system life-cycle.
- The principal issues that need to be address during the creation of a system architecture are Command and Control, Information Interchange, Collaboration, Conflict Resolution, Complexity Management, Emergent Properties and Flexibility.
- We can distinguish between three levels of architecture which together form a system architecture.
- A system architecture should be created before any detailed design is done. The creation process needs inputs from the User Needs, System Concept, System Characteristics and the System Requirements. The System Requirements should include both primary and derived requirements, the latter should include any safety requirements.
- Top level system properties are described in Level 3 and Level 2 Architectures, normally using a layered Reference Model.
- The system structure is described in the Level 1 Architectures, these normally consist of Functional, Information, Physical and Communication Architectures, with possibly Control and Enterprise Architectures as well. The expected behaviour of the system structure must also be written down.
- In order to confirm that a system architecture will provide a suitable basis upon which to design and develop an ITS, an assessment should be made on all its aspects.
- A full assessment process consists of verifying that the outputs of each phase is on conformance with the output of the previous phase, and validating that the output of each phase reflects the User Needs. Benefit, and Cost-Benefit analyses can also be done.
- It is possible to estimate the potential performance of the final system using the Architecture Assessment Tool that was also developed by CONVERGE-SA.

2. LIST OF ABREVIATIONS

CAE	Computer Aided Engineering
CASE	Computer Aided Software/Systems Engineering
CIM	Computer Integrated Manufacturing
DRIVE	Dedicated Road Infrastructure for Vehicle safety in Europe
E-R	Entity-Relationship
ERD	Entity-Relationship Diagram
HMI	Human Machine Interface
IRTE	Integrated Road Transport Environment
IT	Information Technology
ITS	Intelligent Transport System
IUTM	Inter-Urban Traffic Management
ODP	Open Distributed Processing
OO	Object-Oriented
OSI	Open Systems Interconnect
SAd	System Administration
SATIN	IRTE System Architecture and Traffic control INtegration
SD	System Dictionary

3. INTRODUCTION

3.1 Purpose of the document

This document is aimed at system designers and has the objective of bringing about the use of "good practices" in the area of system architecture. It does this by stressing the importance and the need for a system architecture to achieve working and workable systems capable of integrating well with other systems. A rationale, together with a set of techniques, is proposed which, although not mandatory, will assist system designers in the development of their architectures. A case study is given in Appendix T which illustrates how these guidelines would be used in practice on a project. A proposed Architecture Deliverable structure is provided which will help projects present their (system requirements and) architecture definitions based on the guidelines.

This document also provides systematic approach for the assessment of an Intelligent Transport System (ITS) system architecture which can be applied by any Transport Telematic project. For maximum effect this assessment process should be performed either during the creation of a system architecture, or immediately afterwards so that any deficiencies can be rectified quickly and cheaply. The assessment methodology consists of techniques and checklists that can be applied to the results various phases in the system architecture stage of the life-cycle.

3.2 Structure of the document

Section 4 defines the concept of system architecture and describes the various elements that would normally make up a full system architecture. It also defines what is meant by architecture assessment.

Section 5 discusses the various issues that need to be considered during the creation of a system architecture.

Section 6 describes that part of the system life-cycle in which the system architecture should be developed. Starting from User Needs, the processes involved in producing the System Concept, System Characteristics and System Requirements are described. These are then used to create the Level 3 and Level 2 Reference Models, after which the various Level 1 Architectures can be produced. Consideration is also given to the development of system architectures for those systems which have to be integrated with existing systems, as opposed to being created for 'green field' sites.

Section 7 describes the various processes that should be undertaken for a full assessment of a system architecture. It will be useful to read this Section before any development work begins, because it does look at the products of the development process from a different viewpoint,

and time could be saved in the case that a product would otherwise be written in an unsatisfactory manner.

Section 8 contains the references used in these Guidelines, and a bibliography of texts around the subject of system architecture.

There are a number of appendices which contain descriptions of techniques and checklists for use during the development and/or assessment processes. Appendix T contains a case study which demonstrates the practical application of system architecture development. Appendix U contains some background information which expands on the issues raised within the main document, its objective is to introduce new concepts which could stimulate the reader. It is recommended that the reader should refer to this appendix to obtain a more theoretical background.

3.3 Who we are

CONVERGE - System Architecture (CONVERGE-SA) was a horizontal project with the following participants:

a) Atkins Wootton Jeffreys - Co-ordinator

b) ERTICO - Liaison with projects and point of contact

c) Heusch/Boesefeldt

d) ISIS

e) MIZAR

f) The University of Leeds

g) TNO-TPD

3.3.1 Our aims

The aims of CONVERGE - System Architecture can be summarised as follows:

- To promote System Architecture in the Transport Telematics Programme by:
  - providing common guidelines and tools;
  - working with the Framework IV Transport Telematics Projects;
- To identify "good practices" in the area of System Architecture;
- To identify the key attributes of Integrated Road Transport Environment (IRTE) architectures and then of Integrated Transport System (ITS) architectures.

It was the objective of CONVERGE-SA to work with projects and not to dictate to them. This version of the Guidelines are made up from the earlier versions of the separate Development and Assessment Guidelines, combined with feedback after they had been used.

4. DEFINITION OF SYSTEM ARCHITECTURE:

4.1 Introduction

All systems have an architecture even though most developers do not yet explicitly write it down. Thus, for example, when a decision is taken to use only two digits to indicate the year, part of the architecture becomes "this program will only work with dates from a single century". The fact that few people have written down this particular architectural feature is likely to lead to the "millennium time bomb".

The issue of architecture becomes of increasing importance when systems are created from the integration of two or more sub-systems. There is a naï ve assumption that systems integration is ***only*** about data communication. This myth remains despite the great difficulty that many developers have in creating good quality integrated systems. We need to consider whether or not there might be a fundamental reason for this difficulty, namely that of incompatible architectures.

All systems are designed to work in an environment (possibly more than one) and assumptions are made about that environment which are then built into the design. It is the top-level statements and/or assumptions about a system which make up the architecture of that system, providing it with form and style as well as the more well known attributes of functionality, size, performance etc. The architecture provides the structure around which the system is developed. Once this structure has been defined, either implicitly or explicitly, it is usually very difficult and expensive to change it later. The Docklands Light Railway discovered this fact when it wanted to operate trains longer than two carriages with stations which had only been built to this original length. A knowledge of the system architecture is vital.

4.2 Working and Workable Systems

Once the existence of the system architecture is recognised, it can be used for the benefit of the entire system life-cycle. The objective of a system architecture is to provide ***a stable basis for a working and workable system***. The requirement that a system should be "working" is obvious. A working system is one that not only has a set of fully functioning sub-systems, but for which these sub-systems co-operate fully to provide the full functionality required by the goals of the system. However, the need for a system also to be "workable" is often overlooked in the rush to produce a system that can be seen to be doing something. A workable system is not only pleasant to use, but is also easy to manage and maintain for its planned lifetime.

A system architecture therefore encompasses both the goal oriented functions that provide the "working" objective, and also the supporting functions that provide the "workable" objective. Figure 4.1 shows that there are, in fact, a number of different sets of people for whom these objectives must be satisfied, and they are likely to be interested in different attributes (see

Section 6.2.2). One particularly important stakeholder who may, or may not, be included in the categories of 'Problem Owner', 'Service Providers' or 'Users' is the organisation which takes the financial risk and provides the money. In many cases its will be (local) government, and the politicians behind it, that will have a major influence over what will be constructed.

A system architecture must therefore provide flexibility so that solutions can be developed which satisfy the separate objectives of the various clients in the environment chosen for the application. It must also lead to systems that are both manageable and maintainable throughout their lifetime.

4.2.1 Feedback from Experience

Figure 4.1 contains the arrow "Feedback from Experience". In most other branches of engineering more is learnt from those systems which have failed, than from those which have succeeded [Petroski 1982]. Unfortunately, despite the many failures of informatic and telematic systems, few formal studies have been made as to what actually went wrong and where. Many failures of large systems are not due to a single fault, but can be result of interacting faults from different parts of the system. The avoidance of these failures is particularly challenging especially when the different parts are normally considered to be the subject of different (academic) disciplines, and the person who should be considering these issues at an early stage in the project is the **System Architect**. Ideally, therefore, the system architect should have considerable experience in the development of systems in general, and of multi-disciplinary systems in particular.

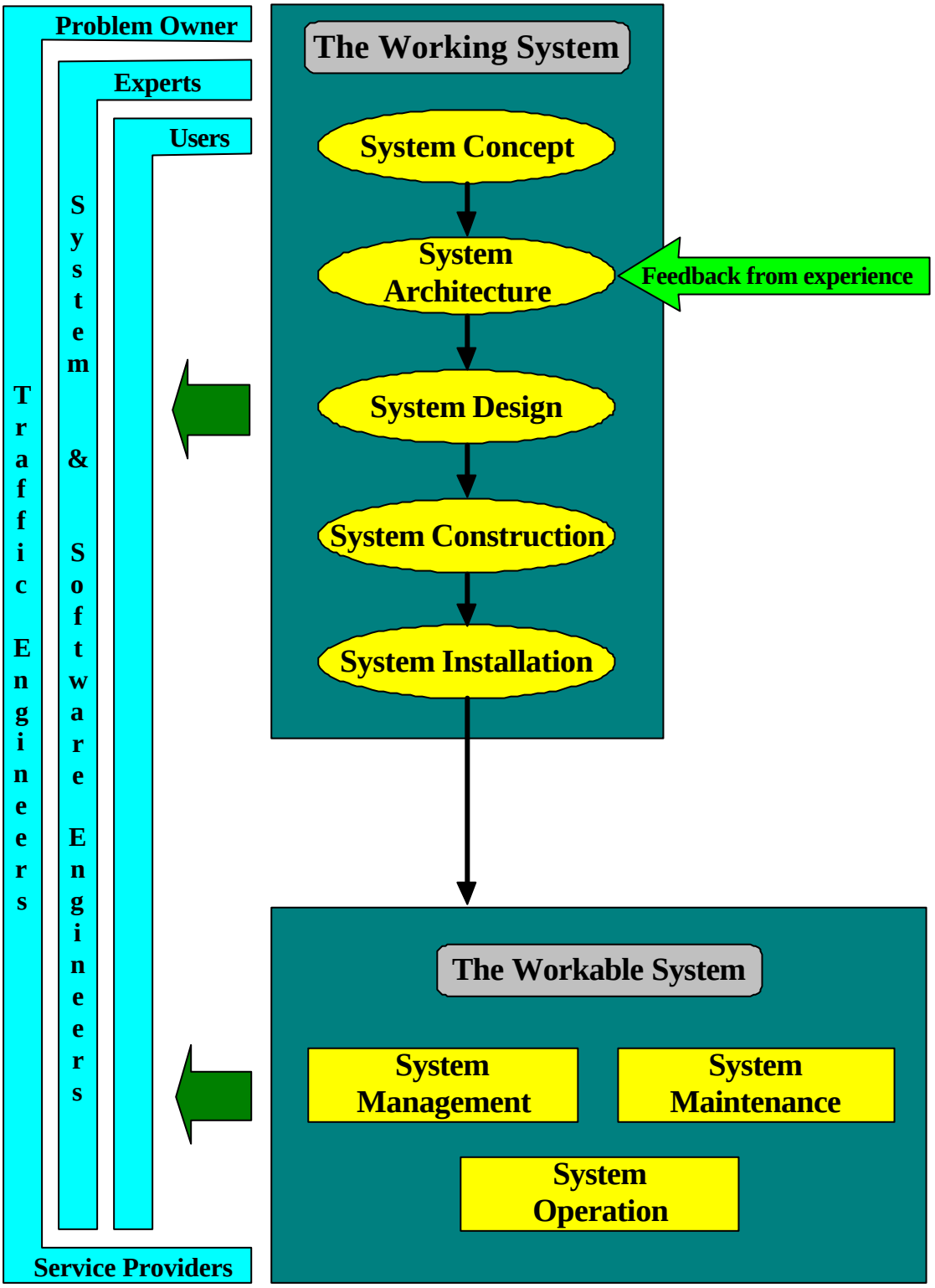


Figure 4.1 - Working and Workable Systems

4.3 Levels of Architecture

The popularity of the term system architecture is causing confusion, in particular it has the effect of blurring the distinction between an architecture and a design, to the extent that many people do not even realise that there is a difference. When you visit two stately homes, each in the same architectural style, you do not expect them to be identical; as would be the case if they had the same design. An architecture is thus something at a higher level than a design, such that whilst it remains constant, many different designs can conform to it. We can distinguish between (at least) four levels of architecture for ITSs, as shown in Figure 4.2.

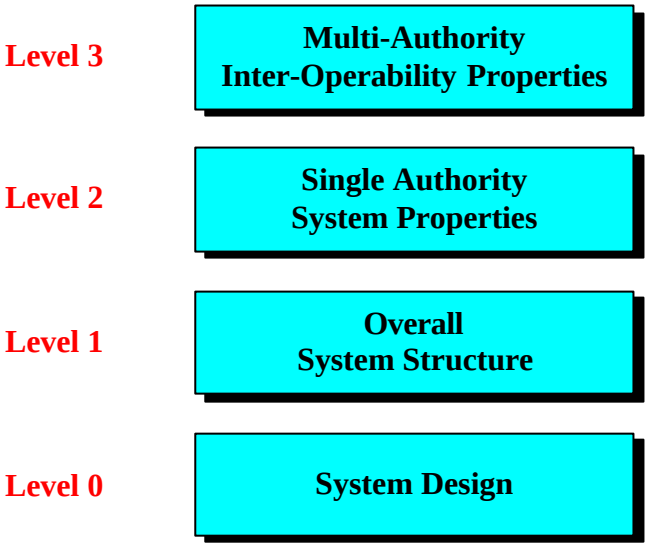


Figure 4.2 - Model for ITS System Architecture

We can envisage the process of creating each level as the identification of anything and everything which is needed to provide a stable basis for the working and workable system under consideration. Each level of architecture fixes those assumptions that need to be stable at that level for the environment(s) in which the system is to operate (see also Appendix C). The principal issues that need to be addressed during the creation of a system architecture are:

- Command and control
- Information interchange
- Collaboration
- Conflict resolution
- Complexity management
- Emergent properties
- Flexibility

A ***Level 3*** Architecture is necessary when the need for inter-operability between autonomous enterprises or authorities arises. Whilst the nature of the Level 3 Architecture is similar to that at Level 2, the issues to be considered at Level 3 are likely to be different. The means of achieving consensus will also be different since the system architect will have to negotiate,

rather than dictate, a solution. It is for this reason that it is advisable to deal with all the autonomous enterprises inter-operability issues separately, rather than trying to incorporate them into Level 2. (See also Section 6.6.1).

The ***Level 2*** Architecture is necessary for the integration of functions and sub-functions in a working an workable manner. It is normally written as one or more Reference Models in which the main information and control flows are identified. If the ITS comes under a single authority then a Level 2 Architecture is probably the highest necessary.

The ***Level 1*** Set of Architectures defines the overall structure of the system, and how the sub-systems relate to each other. It will normally consist of at least four separate individual architectures (see Figure 4.3):

- Functional Architecture - this describes the functions and sub-functions [CORD D004-PT3] of the ITS, the flow of data between them, and the main databases (see Section 6.7.2).
- Information Architecture - this describes the data needed by the ITS, and their interrelationship (see Section 6.7.2).
- Physical Architecture - this describes the grouping of the functions into physical units, or even "market packages", and the communication lines between them (see Section 6.7.4).
- Communication Architecture - this describes the flow of data between the physical units both in terms of message sets, and the characteristics needed of the media (see Section 6.7.4).

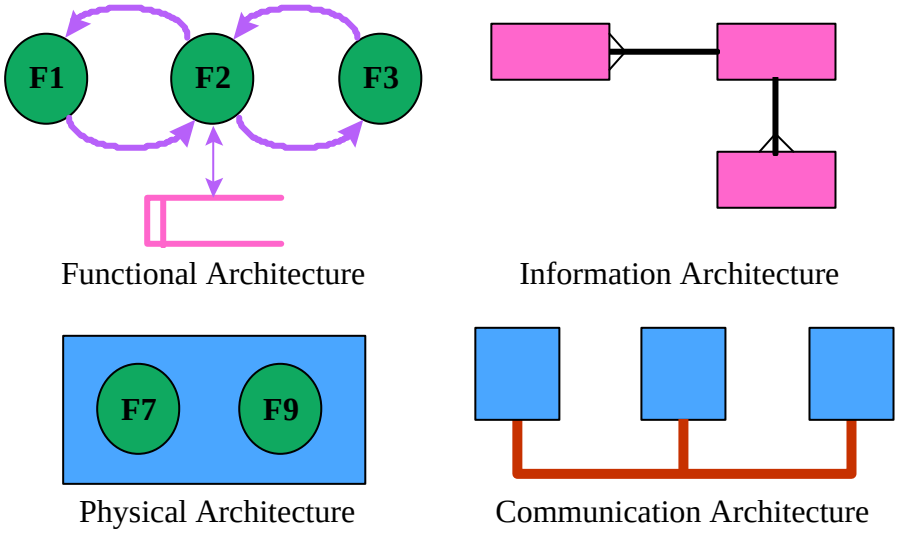


Figure 4.3 - Examples of Level 1 Architectures

It should be noted that a Level 1 Architecture should be, as far as possible, technology and/or manufacturer independent. Thus, for example, a Level 1 Communications Architecture might

indicate the necessity for a broadcast communications architecture, whilst at Level 0 the decision may be taken to use Ethernet.

For some ITS is may also be useful, or even necessary, to include two additional architectures:

- Control Architecture - this describes the method of controlling of the system (see Section 6.7.2);
- Enterpise Architecture - this describes the commercial and/or business relationships of the various enterprises or authorities within the ITS (see Section 6.7.1).

The ***Level 0*** Architecture is not really an "architecture" at all; engineers have been quite happy to call this a detailed design for many years. It is a manifestation of the Level 1 Sets of Architectures with each sub-system and component being fully described, and all the necessary standards having been chosen.

4.3.1 Infrastructure

In normal speech an infrastructure is a framework of equipment to which other equipment may be added, an existing infrastructure is therefore defined at Level 0 and should not be confused with an architecture. An existing infrastructure will also have Level 1 and 2 Architectures, though they may not necessarily be written down explicitly. The architecture of existing infrastructures are of particular importance when there is a need to integrate with legacy systems (see Section 6.10).

4.4 Further Implications

A **Level 2** Architecture can lead to a family of Level 1 Architectures, and a Level 1 Architecture can lead to a family of Level 0 Architectures. It therefore follows that in order to have compatible instances at Level 0 it is necessary to fix the architecture at Level 1, and in order to have compatible instances at Level 1 it is necessary to fix the architecture at Level 2. Indeed it may also be true to say that in order to successfully integrate two systems at Level N, they must have a common architecture at Level N+1. Whilst this is a novel hypothesis it does go a long way towards explaining the current high failure rate of integrated systems.

The activity of creating a system architecture is uncommon and hence there is little expertise normally available. It is particularly important to understand the distinction between a design, an architecture and a system architecture.

Design

Normal design techniques, and the life-cycle models that incorporate them, are based on the assumption that it is possible to obtain a full and complete set of requirements, and produce a product that will satisfy them. This is the deterministic approach. A design is deterministic and describes, normally in detail, exactly how a particular model of the system will be created. There are no options remaining in a design [Crowe 1996].

Architecture

This is a design of a design (at Level 1) and describes the inter-relationship of entities. An individual architecture is usually only concerned with one aspect of the entire system; thus the functional architecture describes the inter-relationship of the various functions etc. An architecture is a description of a class of solutions and will permit the creation of a number of designs, with different components, to achieve the same objective though perhaps in different environments [Crowe 1996].

System Architecture

A system architecture is the set of all the individual architectures, and any other statements, that describe the essence and constitution of the system. Whilst a system architecture will not in itself describe everything, all aspects of the system must be contained within it. Sometimes this can be achieved using system requirements, which state what the system has to do, rather that system specifications which state how (in some detail) the system has to do it; on other occasions it is possible to permit options within a fixed structure [Rechtin 1991].

A ***system architecture is therefore not a design, nor is it a system***, it is a description which forms the basis for a class of systems and hence for a set of designs. A system architecture describes all the attributes for a class of systems, and specifies those structures which are fixed, and those which may have multiple instances. For a class of ITSs it is the amalgamation of knowledge from diverse fields of expertise; indeed a system architect must be a "Jack of all Trades" covering all the parts (an holistic approach), and all the aspects (an ontological approach), for the entire life-cycle [Rechtin 1991] (see also Appendix I). It should contain a description of both the functional properties and the behavioural properties. In order to distinguish between them then consider either a public utility before and after privatisation; or two word processing packages, one based on Windows and the other on DOS. In each example the same basic set of functions are being carried out, but the way in which this happens, and is experienced by the user, may be very different indeed. The functional properties form the basis of the "working" objective, whilst the behavioural properties contribute to the "workable" objective.

4.5 What is Architecture Assessment

The word 'assessment' has a number of possible meanings and it is therefore useful not only to state what architecture assessment is, but also what it is not. There are a number of closely related words which are sometimes confused:

Assessment - the undertaking of an investigation in order to arrive at a judgement, based on evidence, of the suitability of a product for its intended purpose.

Auditing - the process of examining a product and its related documentation for accuracy, quality, completeness, consistency and traceability, and that its production has been performed in conformance with a specified process.

Certification - the process of obtaining regulatory agency approval for a function, equipment or system, by establishing that it complies with all applicable statutory regulations.

Evaluation - the process of measuring the effects of a system in operation to establish the degree to which it satisfies its intended purpose.

Review - the process of checking the conformance between the requirements of a product and the draft product, with the aim of finding anomalies.

Clearly there are overlaps between these processes, but the important points to notice are that:

a) except for evaluation, and occasionally assessment, they are all performed before the operational phase of the life-cycle.

b) assessment is likely to include both Audit and Review, especially if the production process is novel, and itself needs possible improvements.

c) assessment is not certification, and may therefore be done in-house.

The need for Design Reviews to take place before implementation is now widely recognised, and with the introduction of the system architecture phase to the life-cycle [see Section 6], there is now a corresponding need for an assessment of the architecture:

- to identify any weakness before the normal design phase commences;
- to confirm, as far as possible, that a system built in conformance with this architecture will provide the benefits expected of it.

The "10:100:1000" rule describes the savings in cost which will be achieved the earlier in the life-cycle that faults are discovered. This rule highlights the fact that the cost of fixing a fault early in the design might cost, say, 10 ECU; during development the cost to fix the same fault will rise to 100 ECU because of the work that must be re-done. If the fault is not discovered until the system is in operation the cost to fix it rises to 1000 ECU. In practice, of course, the costs are higher but their ratio remains the same.

5. SYSTEM ARCHITECTURE ISSUES

During the creation of a system architecture there are a number of issues that recur throughout the process, they have therefore been extracted and discussed in this section.

5.1 Complexity Management

One of the obvious consequences of building a large system is that it can have a tendency to become incomprehensible to those who have to design or use them. This is due to the phenomenon of the combinatorial explosion which increases exponentially as each sub-system is added. Failure to implement fully functional systems can, in part, be traced to an inability to manage the actual, as opposed to the perceived, complexity. The basic principle of complexity management is often expressed by the mnemonic KISS (Keep It Simple, Stupid), e.g. by maintaining transparency of structure. However one must be aware of the danger of not making too many simplifications such that the solution becomes simplistic and hence unworkable.

5.1.1 Connections

Complexity can, in part, be managed by considering the means by which the parts of the system are joined together.

Cohesion

A component is said to exhibit a high degree of cohesion if the elements in that unit exhibit a high degree of functional relatedness [Sommerville 1992]. A cohesive object is one where a single entity is represented and all of the operations on that entity are included in that object. Cohesion is a desirable characteristic for both the design and the maintenance phases of the system life-cycle.

Coupling

Coupling is an indication of the strength of interconnections between units. Highly coupled systems have strong interconnections, with units dependent upon each other. Coupling of this form may be needed to create an integrated system with emergent properties (see Section 5.6). Loosely coupled systems are made up of units which are independent or almost independent. Coupling of this form is needed when two (dissimilar) systems need to harmonise with each other.

5.1.2 Abstraction

Another way of keeping control over the system and its design is the use of abstraction, i.e. to produce a design (of a design) of a design. Although this can be difficult to do for the first time, because the thought processes are quite different from those normally taught and used, the results can be very simple and are often considered ‘obvious’ by those who only read them

and have not participated in their creation. This is the approach recommended within these Guidelines.

5.2 Consistency

A system architecture is constructed from a number of disparate architectures and other statements. In order that they coalesce correctly it is necessary that they are both self-consistent, and consistent one with another. There are only a few techniques to assist in this task, e.g. the PASSPORT Cross (see Appendix P), and even this should be supplemented by a formal Review.

5.3 Flexibility

Normal design techniques are based on the possession of a full and complete set of requirements, and aim to produce a product that will satisfy these requirements. This is the deterministic approach. A system architecture however, is a description of a class of solutions and the more requirements that are fixed, the smaller the number of possible solutions that can satisfy them. We need a methodology for the development of system architectures that will ensure the possibility of a non-deterministic approach when developing a number of compatible ITSs, each satisfying their own local conditions. It is therefore essential to identify the minimum set of fixed assumptions that are necessary to describe the ITS. This is particularly important at Levels 2 and 3 where each decision taken may limit the choices available at Levels 1 and 0.

Each system needs to be flexible to meet the varying demands of the system clients, and to be able to respond to the stress of a changing system environment. However, underlying this desire for flexibility is also a desire that once a function has been provided, the system will be stable and predictable in its configuration. Thus ***flexible systems must be built upon stable architectures***. An architecture provides a foundation for a class of systems with a set of possible functions, although this set does not need to be fully defined; however, if a new function is required that lies outside this set, then it is the architecture itself which must be changed.

Since the above argument is sometimes seen to be counter-intuitive we give the following analogous example. A human being is based on a fixed architecture which includes a skeleton, muscles, vital organs, etc. This architecture provides a class of systems, namely the human race, each member of which is unique (flexible creation). Each system itself is also extremely flexible, being able to perform a multitude of functions (e.g. reading a book, driving a car, etc.). There are, however, some functions that cannot be performed efficiently using this architecture, for example catching mice. If this additional function is required it will be necessary to include parts of the "cat" architecture in that of "human". This modification should

not be undertaken lightly, and a better solution might be to obtain a separate system that only conformed to the "cat" architecture.

A system architecture therefore provides the means of achieving controlled flexibility, i.e. flexibility within a set of known constraints. Thus whilst the choice of technology can usually be omitted from the system architecture and left until Level 0, the partitioning of the system into sub-systems, and how they are connected will have a profound effect on all future possible development paths. All the influences that one sub-system may have upon other sub-systems must be identified and captured in the system architecture. It is usually assumed that the use of software provides almost unlimited flexibility in the development of a system. In fact this is only true before any software is written. As soon as any code emerges it will carry a set of assumptions, and from this point onwards the software will only be flexible within these assumptions. In a similar manner the integration of sub-systems will carry with it a set of assumptions. By defining these assumptions in a system architecture it is possible to ensure flexibility in a desired direction. A system architecture therefore provides a stable basis for flexibility for the entire life cycle of an ITS.

5.4 Maintainability

Closely related to flexibility is the concept of maintainability. It is a common phenomenon that large, complex systems are never finished, need constant attention and change continuously: this phenomenon can be expressed as “the only constancy is change”. It is important to understand the distinction between flexibility and maintainability.

Flexibility is a measure of the number of implementation choices that are possible to cater for different environments, objectives, users, optimisation criteria etc. Maintainability is a measure of the amount of effort, money, time, education and training of staff, etc. necessary to bring the system from the current state to the desired state, without mistakes being made. It is therefore perfectly possible to create a system that is flexible but difficult to maintain, for example, a system made of heterogeneous components and/or strong internal coupling. Maintainability can be improved by adding functionality to the system, which will assist specifically with the task of maintenance.

5.5 Scale enlargement

Experience from all branches of engineering shows that a particular class of design (architecture) is only suitable for limited range of system size. The reasons are varied, the most obvious being an increase in complexity that eventually becomes unmanageable. On occasions some lateral thinking is needed in order to predict the problem, though it will often be "obvious" once it has been identified. There is a "rule of thumb" that states:

"Any time the size of a system is enlarged by a factor of two, a completely new kind of problem should be expected".

Scale enlargement is of particular importance to transport information and control systems because they will start small and will grow in many dimensions simultaneously. These will include:

- geographic growth: as the system expands in physical area;
- functional growth: as the new functions are added;
- growth in the number of data elements: as the functionality increases it is likely that additional types of data will be needed;
- growth of data volume: a natural consequence of the geographic and functional growth;
- growth in the number of users: a natural consequence of the geographic growth;
- growth in the number of user types: a natural consequence of the functional growth.

An important issue here is that the individual growths are related and feed on each other, leading to a sort of exponential expansion which, in turn, may lead to undesirable consequences. A good example of this is the Internet, which not only has longer waiting times as the number of users increase, but it can be very difficult to find useful information from the amorphous mass, due to the limited number of keywords available in any one language.

The consequences of any scale enlargement should be identified and planned for at an early stage in the system development if a good system is not to expand into a bad one.

5.6 Emergent Properties

An emergent property is one that comes into existence because of the way the parts of the system have been integrated; it cannot be allocated to a particular part, e.g. the property of "transport" in a car cannot be allocated to any of the car's components, just to the whole system. The natural place to consider emergent properties is at the architectural level.

Emergent properties can be both desired and unwanted, and the unwanted one are likely to be unexpected. Whilst this makes them even more difficult to find, especially those connected with the behaviour of the system, every effort should be taken to do so. Examples of unwanted emergent properties include:

- Vibrations: a combined structure will have different properties from its component parts
- Electromagnetic interference
- Corrosion
- Noise: related to vibrations
- Response times: see also Scale Enlargement (Section 5.5)

5.6.1 Controllability

The essence of the development of a control system is that it should be able to perform its control functions, and that the control system can be controlled by the system operator in a proper manner: i.e. "control of the controlled system necessitates control of the control system". The violation of this rule can lead to systems that run out of control. Thus controllability is an emergent property of the control system and has to be addressed as such.

Loss of control can occur as a result of loss of sensor information or actuator action, due either to their own failure or through loss of communications, power failure etc. Consideration must be given to the training of operators in the case of failure. Whilst the control system should, of course, be kept as simple as possible (see Section 5), the creation of a separate Control Architecture can help to provide clarity.

5.7 Multi-disciplinarity

One of the consequences of the modern education system is that people tend to be experts in one discipline, and have little knowledge of any other. This is sometimes called the "inch wide, mile deep" syndrome. When such a person is placed in charge of a multi-disciplinary development, he or she has a natural tendency to give high priority to his or her domain to the detriment of the others. Most ITS will have multi-disciplinary aspects, and the system architect must overcome his or her prejudices, understand the various viewpoints, and come to balanced decisions. Related to this phenomenon, and associated with the issue of emergent properties (see Section 5.6) is the need for someone to oversee the system development as a whole. This should also be the task of the system architect.

5.7.1 Hardware and Software

The distinction between hardware and software, and hence their implications for system architecture and design, is not always fully understood.

Multiplication and modification

Whilst software and hardware both have to be designed, software is easy to replicate, e.g. the MSDOS DISKCOPY command. Hardware, however, has to be 'manufactured' each time a new copy is needed. Hence any changes that may be proposed for hardware are normally thought out very carefully because their consequences are usually obvious and great. Meanwhile software is seen to be easy to change because it is much easier to distribute; indeed this is often the reason why a software solution is chosen. However, unless due regard is given to the additional complexity available with software, the modification process can prove to be very error prone.

Failure Modes

The failure mode of many hardware components are often well known and statistically predictable, and they normally manifest themselves by a component that has been working properly turning into one that does not work properly. Software does not fail in this manner; faults are designed into the program which then fails every time that part of the code is executed. In addition it is often possible to state, with a fair degree of confidence, that a hardware product is correct. It is normally never possible to say this about a piece of software, due to its complexity [Jesty 1997].

Software wear and tear

Whilst there is a common understanding of how many hardware components are affected by wear and tear, and will ultimately wear out, it is rarely recognised that an analogous process can happen with some software system. It is particularly noticeable with databases. Over a length of time the data may become obsolete or loose accuracy, records that should have been deleted are still there, records that should have been updated are still in the old version, and records containing similar information have not been updated in a similar manner. Such 'database pollution' can lead to an overall reduction is the effectiveness of the database, i.e. software wear [Neumann 1995].

5.8 Verification &Validation

At all stages in the creation of a system, and a system architecture, consideration must be given as to how it will be tested. A good system architecture should identify well defined sub-systems with clear interfaces to facilitate a well ordered testing process to be undertaken during the system development, thus facilitating the gaining of the necessary level of confidence that the system will both perform correctly in its current form, and after any enlargement (see Section 5.5). The system architecture must also allow for the ready replacement of any sub-system or component that was found to be faulty.

5.8.1 Testability

One consequence of the existence of emergent properties (see Section 5.6) is that they can only be validated by testing the entire system. This is sometimes called "The Aeroplane Principle":

> "You can test the parts as much as you like, but the only way to see whether the aeroplane is able to fly is to let it fly".

In fact the situation is worse because, due to the complexity and scale enlargement issues, at no single point in the development life-cycle can a large system be fully tested (ignoring the fact that software cannot be tested exhaustively anyway). It is therefore essential that testability should be built into the system from the beginning, and hence be a dominant feature of the

system architecture. Plans will also have to be made to validate those features which cannot be tested directly.

6. SYSTEM ARCHITECTURE DEVELOPMENT

Normal design techniques, and the life-cycle models that incorporate them, are based on the assumption that it is possible to obtain a full and complete set of requirements, and produce a product that will satisfy them. This is the deterministic approach. A system architecture however, is a description of a class of solutions, and we need a non-deterministic approach to the development of system architectures if we are to be able to create a multitude of compatible ITSs, each satisfying their own local conditions. This section describes the production of system architectures during the life-cycle, and Appendix A contains a possible contents list for a System Architecture Deliverable.

Whilst it is normal to show the processes needed for the development of a system in terms of a life-cycle model, such models can only highlight certain features. One common life-cycle model for systems, used to emphasis the verification and validation processes, is the "V model" shown in Section 7. Another common life-cycle for systems is known as the "waterfall model" concentrates on the development process, and it can be expanded to show the phases when the system architecture is developed.

Figure 6.1 is based on [Rechtin 1991] and shows the main external influences on the various phases of the life-cycle. The shaded section has been expanded into Figure 6.2 upon which the remainder of this Section will be based. It is important to note whilst reading this section that, although it has been written in a sequential manner, some of the processes may run in parallel and they may all need many iterations before a satisfactory solution has been reached. Examples of the output of each stage of the system architecture life-cycle can be found in the case study (see Appendix T).

Many systems will not be created on a "green field" site, and will require existing systems to be integrated with the new systems. System Architecture has an important role in this situation, which is discussed further in Section 6.10.

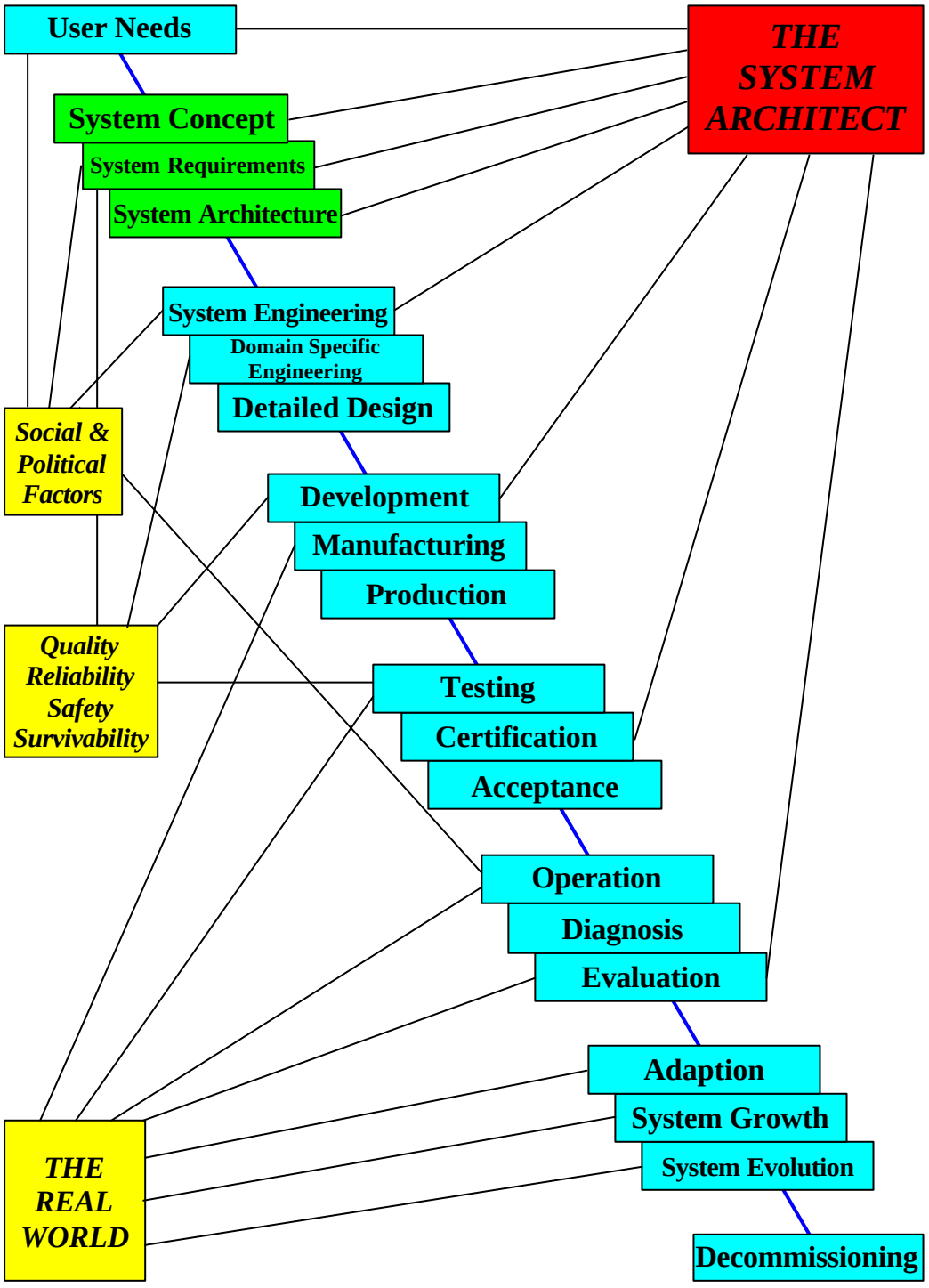


Figure 6.1 - Expanded Waterfall Life-cycle.

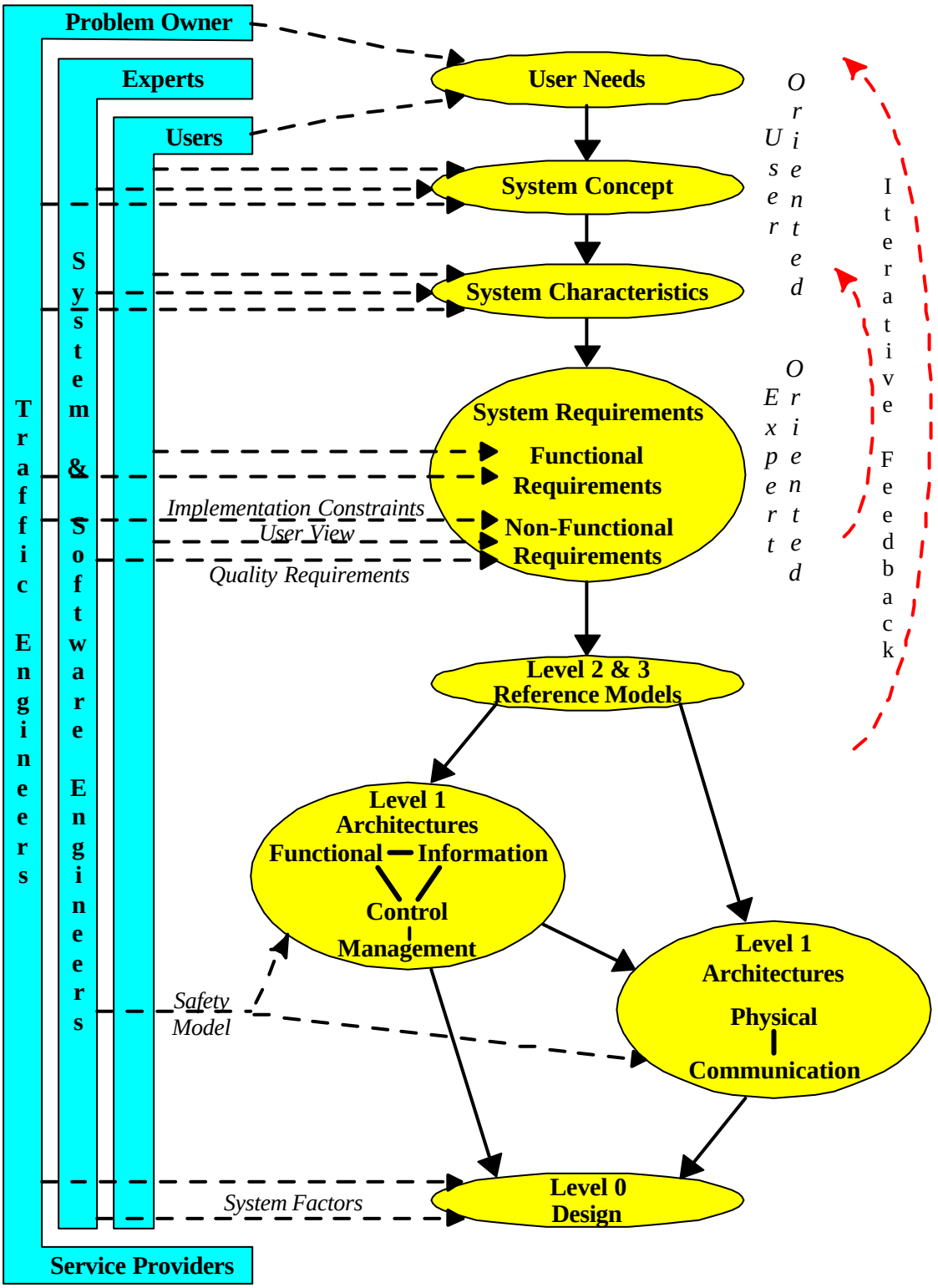


Figure 6.2- System Architecture Development within the System Life-Cycle

6.1 User Needs

The first three stages of the development of the system architecture are oriented around the needs and expectations of the user, thus:

- User needs - the unsatisfied desires
- System concept - the proposed solutions
- System characteristics - the main features of the proposed solutions

The needs of the user can be expressed in a number of different ways from a simple "wish list" to a formal statement of "User Requirements". User Needs when expressed as a list of desired functions can often end up being quite detailed, and there is always a danger that they may be inconsistent, especially when they have been gathered from more than one source.

An alternative approach is to express the needs of the user in the form of the Services that the user will wish to receive. These are a high-level form of user needs which will have to be decomposed into functions later in the development life-cycle. See also Section 6.7.

In addition to stating *what* services or functions are needed, the User Needs are also likely to state *how* the user wishes to receive those services or functions. They may also vary in their level of abstraction, with some affecting the high-level System Concept, and others affecting the low-level detailed design.

The SATIN Task Force produced a set of high level user requirements for European ITS [SATIN D007-PT1].

6.2 System Concept

Is it necessary for everyone involved in a project to have a clear understanding of what is intended to be done and why. An effective way to achieve this is by the key personnel encapsulating their ideas in a few sentences so that the entire team may refer to them at any time in order to ensure that it is maintaining the top level objectives.

One of the specific features that must be clear as a result of this exercise is exactly where the ***system boundary*** lies. The exercise requires the identification of that part of "anything and everything" that lies inside the system, and thus subject to the system architecture, and the remainder that lies outside the system, and hence may be effected by, or have an influence on, the system.

It should be noted that, although the formal output of this exercise may only be one or two pages long, the performance of the exercise may take some time. This must not be treated as an undesirable overhead, since the need for a long discussion probably indicates a potential conflict of ideas within the team.

The key issues to be considered are:

- WHAT should the system achieve?
- WHO are the various persons who will be affected by or have an impact on the system?
- HOW does the team propose to produce the system?

6.2.1 Vision Statement

This should be a statement of WHAT it is that the system should achieve. It should be possible to capture the main essence of the system in about two sentences.

6.2.2 Identification of the Users

It is important to identify and categorise all the various persons who will be affected by, or have an effect on, the final system. There are four different categories of the user who may either affect, or be affected, by the system. For Information Technology (IT) systems they have now been given the following titles:

- **Want IT** - The service providers want the system to solve (or diminish) traffic problems, or to provide information services to the public.
- **Make IT** - Component suppliers will deliver hardware and software components for the system. System integrators will combine the components into a complete system.
- **Use IT** - The primary users will benefit from the output of the system. The secondary users will control the system and provide input.
- **Rule IT** - The local authorities have the responsibility for issuing the regulations on how to implement and use the system. The international authorities may also issue regulations, as well as standards and recommendations for international inter-operability.

A check must be made to ensure that all the users associated with the system have indeed been identified. For ITSs (see Figure 4.1) the Problem Owner is likely to be an arm of local or national government who is trying to provide a solution with the advice of its Traffic Engineers. It will be normal for this authority both to pay for the system and to manage its operation, as consequence it is likely to have a strong influence over the User Needs/Requirements. An alternative scenario is for the private sector to identify a gap in the market of the services that could be provided, and produce a system for sale either to the traffic authorities or directly to the general public. The system and software engineering Experts will work closely with the Problem Owner and the Traffic Engineers during development of the system. It should be noted that, whilst they are experts in System or Software Engineering, they are unlikely to have a full knowledge of the application. It is therefore vital that there should be a sympathetic interaction between the Problem Owner and the Experts: in particular the Experts will bring knowledge of experience of other systems, possibly not in the transport field, with lessons that

can be applied to the proposed system. The Problem Owner must be receptive to suggestions. Once the system has been commissioned it is likely that the Experts will have completed their contract. Effective plans must therefore be made for the maintenance of the system throughout its lifetime: in particular it must be recognised that even if the same organisation of Experts is used, it is most unlikely that the same individuals will be used. The consequence of this is that specific attention must be given to documentation both during the development and the maintenance parts, and this must be specified early in the life-cycle, and hence be part of the system architecture.

The relationship between the Users and the Service Providers will vary depending on the application. An enforcement system imposed upon the general public will be considered in a very different manner to a commercial system providing facilities for which the user has to pay. Indeed it is possible to envisage the Service Provider having quite different criteria for the "working" and "workable" objectives to those of the Users. It is a requirement of the system architecture to reconcile those differences in perspective.

6.2.3 Mission Statement

This should be a statement of HOW the team proposes to produce the system. It should be possible to capture the main essence of the process in about two sentences.

6.3 System Characteristics

Based upon the user needs and the system concept, the various experts should write down the desirable characteristics of the system. Although many of the statements are likely to be qualitative at this stage, it is important that all the aspects of the system are considered by all those who have an affect on, or will be affected by, the system (see Appendix D for a checklist of those issues that should have been considered). The system characteristics will provide a description of the properties that the resulting system(s) should exhibit.

This stage investigates the main features and properties which follow from the system concept. The aim is to promote a full, multi-disciplinary understanding of all the characteristics of the system by the system and domain specific engineers. CONVERGE-SA is not aware of any specific methodology for this task, text will be sufficient in many cases, sometimes diagrams may help. A good starting point for ITS may be found in [SATIN D007-PT1].

6.4 System Requirements

With the variety of categories of users, each with a different interest in a proposed ITS, one can expect the capture of the user needs to be fraught with difficulty. The final collection is likely to be incomplete, inconsistent and vary widely in their level of detail, especially since the contributors are likely to have a wide variation in knowledge of what might be possible. The system requirements, however, need to be consistent, and as complete as possible. During the process of transforming the user needs and the system characteristics into the system

requirements inconsistencies and omissions must be identified and reconciled. Some of the user needs will affect the system architecture itself, e.g. reconciliation of goal conflicts, whilst others will be reflected in the functional and non-functional requirements of the later phases of the development.

The system requirements are (semi-)formal statements about the objectives of the system, possibly written in some (semi-)formal language or diagrams. Whilst the user needs an system characteristics will have been expressed in terms of *what* is wanted, the system requirements specify *how* it will be achieved.

Three broad categories of requirements for ITS have been identified [SATIN D007-PT1], and are shown in Figure 6.3:

- context requirements (formally referred to as "Introductory Requirements") - these specify the reaction to the constraints imposed by the environment on the introduction of the system They may be assumptions about that environment, or statements as to what is needed within the environment (see Appendix E).
- functional requirements - these specify the service(s) that will be expected from the system, and/or the functions needed to provide a working system (see Section 6.7 and Appendix F).
- non-functional requirements - these specify the performance and/or quality attributes of a workable system (see Appendix G).

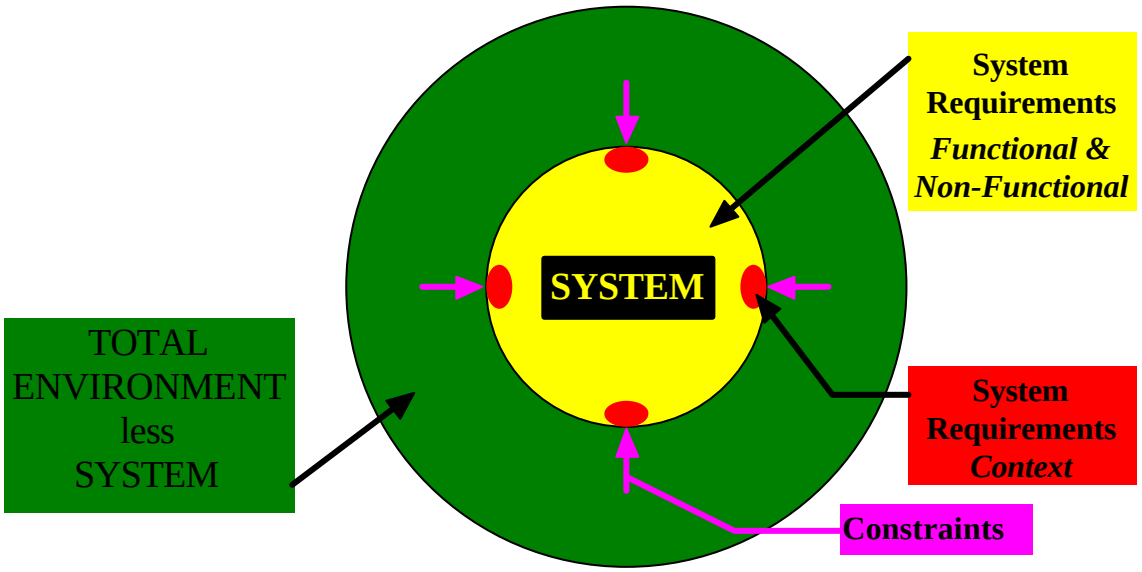


Figure 6.3 - Functional, Non-Functional and Context Requirements

Whilst this categorisation is possibly subject to argument, the important thing is to ensure that all the system requirements are captured, and not the heading under which they are listed. Indeed since most high level non-functional requirements will be later expanded into low level

functional requirements, such categorisation is, to some extent, arbitrary though the headings do indicate the thought processes that should be undertaken to ensure completeness.

Initially the system requirements can be identified as being a direct consequence of the user needs, the system concept and the system characteristics, i.e. the "primary system requirements". However the technical experts will also identify another set of requirements (e.g. safety requirements) that will be necessary to create a working and workable system for the primary system requirements, these are known as the "derived system requirements". For a system as wide ranging and complex as an ITS it is likely that persons from a number of different areas of expertise and knowledge will have to contribute to the process of collating the total set of system requirements.

6.4.1 User Needs vs. System Characteristics vs. System Requirements vs. System Specifications

User needs emanate from the users and are entirely user oriented. They will not necessarily be consistent, and are likely to be expressed in plain text with informal diagrams. Completeness is measured in terms of the numbers and types of users who have contributed.

System characteristics are also user oriented and expressed in plain text with informal diagrams, and permit the users to embellish the user needs, which may be lacking in detail. They start from the user needs and the system concept but as well as receiving contributions from the users, they also bring in the experience of system engineers and system architect(s) who have worked on other systems. It is not necessary for the system characteristics to be entirely consistent, but the most obvious inconsistencies in the user needs should be removed.

System requirements are system and implementation oriented, and will use (semi-)formal text and diagrammatic techniques to capture all the requirements; they will not necessarily be easy to read by the users. They should be consistent and traceable back to the system characteristics. The primary requirements will come from the user needs and the system characteristics, but in addition the system engineers and system architect(s) will add derived requirements to provide the *working* characteristics of the system. Whilst the system requirements may be used to test the resulting system, many of the detailed implementation issues may not have been decided, i.e. they may be technology independent. Figure 6.4 shows the difference between the system characteristics and the system requirements.

	System Characteristics	System Requirements
Main purpose	Basis for requirements	System development
Secondary purpose	Identify assumptions	Articulate assumptions
Orientation	User (domain expert)	System
Origin recognition	Submitting user identified	Not necessary
Organisation	User, system parts, functionality, behaviour	Context, functional, non-functional
Coverage	Anything deemed relevant	Process and product
Internal consistency	Not necessary	Necessary
Completeness	For each user	For the system
Type of mediator	Domain & system experts	System engineers
Character of mediation	Understandable, readable	Exactness
Type of presentation	Text, simple figures, tables	(Semi-)formal descriptions
Character of presentation	Understandable	Implementable
Main point of view	User desirability	System feasibility
Limitation	Time restrictions	Cost of implementation

Figure 6.4 - Difference between System Characteristics and System Requirements

System specifications are system oriented and represent in detail how the future implementation will work. They represent one of the possible manifestations of the system requirements. They are usually only readable by specialists as they should be written in (semi-)formal specification languages and/or diagrams which are rich in semantics and rules for internal consistency, and form the basis for detailed testing. The system specification form the input for the operator manual, user documentation, training manual, maintenance manual etc.

6.5 System Boundary

Once the system requirements have been identified a diagram should be drawn which shows how the proposed system relates to its immediate environment. For a pure telematic system which is only processing data and providing information, this may be done effectively using a Context Diagram which can then become the top-level Functional Architecture (see Appendix L).

For any system which acts directly or indirectly on its environment, e.g. to provide traffic management, the diagram should contain more information that is possible in a Context Diagram. It is recommended that a PASSPORT Diagram is used [PASSPORT 95] which will not only permit qualitative completeness and consistency checks to be performed, but it is also possible to analyse the top level functional behaviour of the system, and to perform a preliminary safety analysis (see below). Experience with the PASSPORT Diagram during the

DRIVE II programme demonstrated that it was extremely useful to create it as soon as possible in the life-cycle.

6.5.1 Preliminary Safety Analysis

Any system that has sufficient power to advise or to control road safety, transport efficiency or environmental quality might exercise that power for either good or ill, and the latter condition could be quite disastrous in the worst case. Developers naturally concentrate on the benefits of their system, especially when communicating with their financial supporters, private or public. However, in order to ensure that their systems do indeed only provide the benefits that they desire, and do not cause any undesired effects, a responsible developer should execute a series of processes to confirm the overall safety of the system. The first of these processes is a Preliminary Safety (or Hazard) Analysis in which "What if …?" questions are asked of the proposed system in order to discover whether any possible failure of a part of the system could produce a safety hazard (see Appendix H). An analysis of the results of a "What causes …?" analysis on each hazard will provide the top-level safety requirements, part of the derived requirements.

6.6 Level 2 and 3 System Properties - Reference Models

The reference models are the area where the context, functional and non-functional requirements are studied in their entirety, and where the optimal strategy for interrelated implementation of the requirements are proposed. During the later stages of development staff will deal with particular parts, and hence a sub-set of the requirements only.

A system can be divided into two main domains, the functions that produce the goal, and the control of those functions. It is sound engineering practice to keep these two domains distinct and separate in order to maintain simplicity of design. Levels 2 and 3 are mainly concerned with the control domain and are usually described in terms of Reference Models. The most common structure is a *layered* Reference Model (see Figure 6.5) in which the various functions of the system are each allocated to a particular layer such that:

- each layer may take input data for transformation from lower layers or from the environment outside the system[1];
- each layer may generate commands for itself or for lower layers;
- output data may be sent to lower layers for display or transmission to the environment outside the system, but not for transformation;

[1] Sometimes it may be advantageous to use the reference model as the nucleus of the system in a PASSPORT Diagram. In this situation data input and output should be shown at the logical level, i.e. where the data is used/created, and not where it enters the system.

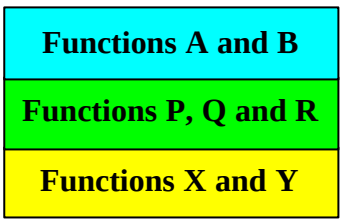


Figure 6.5 - Example of a Three Layered Reference Model

- the number of layers will vary between Reference Models, though experience shows that few systems need more than six or seven layers to represent their full functionality, though additional layers may be needed to improve the behaviour, e.g. for security.

To facilitate the comprehension of this concept, it is useful to refer to some well-known examples of Reference Models such as the Open System Interconnect (OSI) reference model [Zimmerman 1980], or the Computer Integrated Manufacturing (CIM) reference model [ISO CIM]. Both are outstanding examples that have been applied in whole, or in part, to large systems with a fair degree of success. They describe the main structures of the system. The desire for structure is obvious: a system has to have some internal structures and the fundamental ones form the basis of the system architecture. This approach leads to:

- *a family of systems* - a set of systems with many common features. This can lead to a reduction in cost by the use of common components.
- *inter-operability* - systems with a common internal structure are more likely to work together in a successful manner.
- *simplicity* - a reference model contributes to an understanding of the system by both engineers and users. For the engineers this enhances the communication between them and helps to avoid hidden assumptions slipping into the system design. For the users this provides the basis for a consistent mental model which will aid effective and safe operation.
- *stability* - most ITS are expected to continue in operation for many years, whilst at the same time being modified to suit a changing environment and/or user demands. A well formulated reference model will enable all these changes to take place within the system architecture and thus maintain a basic stable view for both engineers and users.
- *flexibility* - reference models do not impose any unnecessary constraints on the design of the system, or on the technologies that may be used in the design. They permit flexibility, indeed support it. Thus reference models describe, rather than prescribe.

A layered reference model delineates the levels of jurisdiction within the system architecture. Although there may be a number of reference models at Levels 2 and 3, each dealing with a single issue, normally they are combined into one model for each level. The reference models are necessary because a system as large and complex as an ITS is unlikely to have one single central point of control. The reference model therefore indicates how the various types of

control are divided throughout the system. Whilst the reference models at Level 2 and 3 both cover the same issues, those at Level 2 may be easier to reconcile under a single Authority, than those at Level 3 when a consensus must be obtained between Authorities.

Layered reference models have already been produced by the SATIN Task Force for Urban, Inter-Urban and In-vehicle IRTE ([SATIN AC13-PT7] see also Appendix J). Road transport projects are strongly recommended to use them as a starting point. Although it will be necessary to confirm that they are suitable, their existence should mean that projects will not have to re-create their own individual reference model(s) from scratch.

Reference models must satisfy a number of requirements, in particular the allocation of responsibility and zones of autonomy. Further discussion on this subject can be found in Appendix I.

6.6.1 Responsibility

Management

The ease by which a system may be managed will depend, to a great degree, upon the structure of the system architecture. It is advisable for the top level control structure of the system to be reflected in the structure of the management, or vice versa. Where a number of organisations are co-operating it may be useful to create an Enterprise Architecture in addition to the Reference Model (See Appendix K).

Many of the components, or sub-systems, of a system may have their own objectives, or goals, which may not always be in harmony with each other. The system architecture must therefore reconcile, for example, the desire of the individual motorist to get from A to B as fast as possible, with the traffic controller's wish to maximise the throughput on a particular stretch of road; or else the necessity to minimise the journey time of emergency vehicles, with a desire for traffic calming in residential areas. The reconciliation process may need some changes to be made to the system requirements.

Safety

The Reference Model should identify the allocation of responsibility for the various types of safety functions throughout the system architecture in a consistent manner.

Security

It is normal practice to impose layers of security around any system that contains personal or sensitive information. The Reference Model should identify these layers within the system architecture.

6.6.2 Zones of Autonomy

It is not sensible to assume that all parts of a large IT system will be operating perfectly all of the time. Indeed some studies have shown that many large systems spend most of their operational life in one failure mode or another [Neumann 1995]. Consideration must therefore be given, within the architecture, to the degraded modes of operation of the system. This is of particular importance for those parts of the system that help maintain a safe situation. In addition corrective and preventative maintenance will require parts of the system to be switched off or disconnected, and the system architecture must permit the safe operation of the remainder of the system during this time, albeit possibly in a degraded mode.

There are two particular issues that must be considered and reconciled within the reference models in order for the zones of autonomy to function correctly. Different functions, or classes of function, will be allocated to different zones within the reference model. It is necessary to ensure that the ***flow of data*** and the ***flow of commands*** necessary for the correct operation of these functions will be maintained in any planned degraded mode of operation.

6.6.3 Production of a Layered Reference Model

1. Choose a number of layers (five, say, at the start of the process), and allocate hierarchical management, safety and security tasks to each of the layers in a consistent manner. Note that although a major task may be allocated to more than one adjacent layer, each layer should be different from its neighbour(s) in some manner.
2. Allocate the functions identified in the functional requirements to each of the layers.
3. By considering the flow of data, and the flow of commands between the functions, check that the following rules have been obeyed:
   - the allocation of functions in Step 2 conforms to the management, safety and security hierarchy chosen in Step 1.
   - each layer may take input data for transformation from lower layers or from the environment outside the system.
   - each layer may generate commands for itself or for lower layers.
   - output data may be sent to lower layers for display or transmission to the environment outside the system, but not for transformation.
4. Repeat Steps 1 and/or 2 and/or 3 until there are no inconsistencies. Note that it may be necessary to change the number of layers (sometimes it may be necessary to split a layer in two in order to isolate a particular function so that it may be developed at a higher level of integrity by a specialist development team, e.g. for safety or security).

6.7 Level 1 System Structure - Functional Issues

Once the Level 2 (and 3) control domains have been fixed, the Level 1 functional domain can be considered. The Level 1 Architectures define the kernel of the main structure of the system, and how the sub-systems relate to each other. There are many different attributes that need to be considered when creating an integrated system and the main ones will each have their own architecture. The activity of the system is described in terms of WHAT the various sub-units will do so that, together, they create the goal oriented "working" attributes of the system Architecture (the Level 0 Architecture, or design, will provide the details of HOW the sub-units will be implemented). Naturally each Level 1 Architecture must conform to the Level 2 and 3 reference models and be consistent one with another.

Technology Independence

It is normal for a Level 1 Architecture to be "technology independent". By this one means that where there is a choice of technologies that may be used, this choice should be maintained until the Level 0 design is started. On rare occasions the technologies available to perform a service are so different that the choice will affect the Level 1 Architectures. In this situation, of course, it will be necessary to include a "Technology Architecture" at Level 1 that specifies the choice taken, and the reason for that choice.

Services and Functions

One definition for a service is:

"A provision or system responding to some public need"

Most IT services have a common information flow down a chain of functions as shown in Figure 6.6. The five functions are:

- Monitoring of data
- Processing of data
- Decision process
- Dissemination process
- Reception

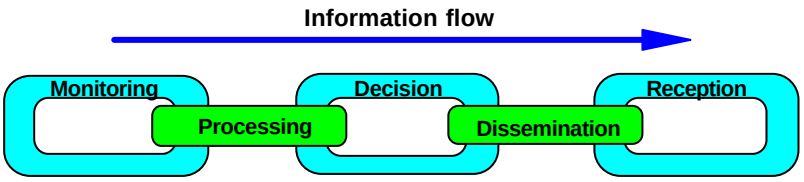


Figure 6.6 - Service information chain

Examples of services in the road transport mode include:

- Lane management
- Parking space management
- Route guidance
- On-trip driver information
- Park and ride
- Travel planning

Each function type is itself a family of functions. For example Monitoring of data could include visibility monitoring, pollution measuring, speed measuring etc. Processing of data could include passenger analysis demand, traffic forecasting, travel planning, etc.

In the SATIN Task Force, a list of functions and sub-functions has been produced for the road mode as an attempt to provide a common basis for functional descriptions [CORD D004-PT3]. Most of the sub-functions described in this list can be seen to belong to one the 5 groups described above, while most functions can be seen to be services. This list has still to be updated as it is only covering the road mode and, for a specialised study of a particular application, it has been recognised that the list does not give enough detail.

6.7.1 Enterprise Architecture

Many ITS either form, or are part of, a business activity. In this situation it is useful to create an Enterprise Architecture to describe this part of the system. This architecture will show the flow of resources between organisations, persons, services and/or functions. Each element in the Enterprise Architecture may be subject to constraints or rules which should be specified. See Appendix K.

6.7.2 Logical Model

Functional Architecture

Starting from a Context Diagram, or otherwise, the Functional Architecture should be produced by the process of functional decomposition. This will show the flow of data between functions and sub-functions, and the data sets that are needed. See Appendix L Road transport projects are recommended to use the CORD List of Functions [CORD D004-PT3] as the basis for their functions and sub-functions in their Level 1 Functional Architecture.

In order to take advantage of the structure provided by the layered reference model, it is possible to use a separate Context Diagram for each layer.

Information Architecture

The Information Architecture defines the structure of the data sets showing the relationships between the various items of data. Where the ITS requires a large database of information, consideration should be given to the production of a full data model. Particular care should be taken for distributed systems that involve more than one controlling organisation to ensure consistency between the various definitions being used. See Appendix N.

Control Architecture

All systems must be controlled, but it depends upon the form and/or complexity of that control as to whether it is necessary to produce a specific Control Architecture, or to combine it with the Functional Architecture. Whilst data driven control may be suitable for some pure information systems, it is not wise to use this method for safety-related systems due to the difficulty of demonstrating that it will be safe under all circumstances, e.g. lack of deadlock. Consideration should be given to using such models as the Finite State Machine, or the Petri Net, or Event Trace Diagrams or What-If Tables for those systems whose control is at all complex.

6.7.3 Object-Orientation

The structured approach for the Level 1 Architectures, which is based on decomposition, is well understood by engineers in the traditional disciplines, and is one of the reasons why it has been given prominence in these Guidelines. It is not, however, the only possible approach for the Level 1 Logical Model. Object engineering, which is based on data abstraction, takes a different approach, identifying data objects, or entities, and grouping functions around these. This gives a single coherent view. It is often seen as a re-packaging of systems which ensures minimum interfaces and thus produces more modular systems.

Object engineering starts, like data modelling, by recognising concepts or objects and their relationships to other objects but instead of recognising functions in a separate model, it identifies the functions performed on or by each object. For example a variable message sign may be seen as an object which receives certain instructions from a controller and accordingly performs the service of displaying certain items of text. Its internal workings are not known by the controller, or by the road users, who see the sign. The notion of services provided by an object can be very useful in a high-level analysis of the objectives or benefits to be provided by the system.

The object-oriented approach has become very popular for information systems where there are claims for:

- greater simplicity in system development and maintenance;
- localisation of change;

- maximum opportunity for the re-use of components;
- good management of variability;
- protection of data from illegal access;
- better management of complexity.

There are a number of different object-oriented methodologies, each with their own set of text books e.g. [Priestley 19??] or [Rumbaugh 1991], and many are supported by CASE tools. Computer science students are now normally taught this approach. An object-oriented approach has been taken for the Enterprise Architecture in the case study (see Appendix T). A discussion on the new Standard for Open Distributed Processing (ODP) [ISO 10746] can be found in Appendix M.

The object-orientation specifically hides complexity in its modules, and for this reason is a valid method for use at the higher levels of architecture and design. However care must be taken if a full object-oriented approach is to be taken down to, and including, programming, e.g. in C++. The concept of information hiding, or the use of run-time libraries for performing complex tasks, should be used with particular care for any system that is safety-related, because in this case a full understanding of the entire process must be demonstrated and the provenance of the libraries may be unknown. There is no evidence that object-oriented technology improves reliability [Hatton 1997]. It must also be remembered that object-orientation, being a software architecture, does not and cannot replace a full system architecture as advocated in these Guidelines.

6.7.4 Physical Model

Physical Architecture

The Level 1 Physical Architecture shows the grouping of the functions specified in the Level 1 Functional Architecture into physical units, often to provide a "market package". It may also show the location of the physical units and the communication paths between them [Hatley 1987]. The Physical Architecture also shows the distribution of data and whether either functions and/or data are replicated to improve performance or provide redundancy. Normally, the Physical Architecture should be technology and/or manufacturer independent.

Communication Architecture

The Level 1 Communication Architecture describes the characteristics of the various channels that have been identified as being needed in the Physical Architecture [Hatley 1987].

The communication architecture, part of the physical view, describes the way subsystems communicate. The communication architecture focuses on the physical transmission of the information between sites. A communication architecture is related to the physical architecture

and each link between spatially separated sub-systems has to be described in the communication architecture using the following attributes (non-exhaustive):

- type of communication medium (wire, radio, infrared, visual, etc.);
- physical characteristics of data flow (regularity, volume, speed, encoding techniques, etc.);
- logical characteristics of data flow (latency, information composition, etc.).

Consideration should be given to using the OSI 7 Layer model for computer communications [Zimmerman 1980], since there are a number of Standards already extant for handling the bottom three layers.

6.8 Level 1 System Structure - Behavioural Issues

We need to distinguish between system behaviour and system functionality. The system functionality is concerned with "what" the system is to do, whilst system behaviour is concerned with the "manner" in which it is to be done. This will have been specified in the non-functional requirements. In general a system function can be tested during development. System behaviour covers those properties which are important but can only really be validated by use. Such properties include performance, safety, security and system interface (e.g. user friendliness). For example, software products from different suppliers may be identical in their functionality, nevertheless these products may differ greatly in their behaviour; e.g. ease of use, degree of configuration possible, failure rate, resistance to user errors, assistance provided, and look and feel. These differences can become extremely important when one desires to integrate two software products.

It is during the study of the desired system behaviour that conflicting goals may be identified. These conflicts must be resolved so that all systems, and sub-systems, that conform to this architecture behave in a consistent manner.

The Level 1 characteristics of the system identify the behavioural features which, together, will create the "workable" attribute of the System Architecture. The features are likely to be written in textual form, rather than with diagrams, with possible references to in-house practices, or Guidelines (e.g. [MISRA 1994])

6.8.1 Usability

In Figure 4.1 and Section 6.2.2 we have highlighted that there may be a number of different sets of people who will interact with an ITS in some manner or another. The system should be user-friendly and easy to use by each and every one. There is growing literature on the design of information screens and control panels in a variety of industry sectors, as well as in the previous Transport Telematic research programmes.

6.8.2 Risk/hazard Analysis

It is not sensible to assume that all parts of a large IT system will be operating perfectly all of the time. Consideration must therefore be given, within the architecture, to the maintenance of the "workable" attributes during the degraded modes of operation of the system.

Safety

The safety of the system will be provided by a combination of the way in which the functions are organised within the Levels 3,2 and 1 Architectures, and the processes used to create the system. Safety, like quality, cannot be added onto a system, it must be built in. There are many relevant Standards, Guidelines and Frameworks covering the many aspects of the safety of ITSs systems (e.g. [MISRA 1994], [EMCATT 1995] and [PASSPORT D9]).

Security

The security of the data within a system will, to some extent, be maintained by the control structures and data flow given within the Levels 3, 2 and 1 Architectures. Most of the security will, however, have to be designed into the relevant functions. There are a number of Guidelines on this matter (e.g. [ITSEM]).

Operability

A system must not only be usable (see Section 6.8.1) but it must remain so under a degraded mode of operation. Hazard analyses should be performed upon the System Architecture to confirm the usability of the system in its many possible configurations.

Maintainability

An ITS, especially one that is successful, is likely to remain in operation for decades, and maintenance, although not necessarily continuous, will take place during the entire operational phase of the system. In order that it may be undertaken in a cost effective manner the system architecture must take maintenance into account. Corrective and preventative maintenance will require parts of the system to be switched off or disconnected, and the system architecture must permit the safe operation of the remainder of the system during this time, albeit possibly in a degraded mode.

The maintenance phase of the system life-cycle is when the flexibility offered by the system architecture is put to the test. Sooner or later the environment within which the system operates will change, and it will be necessary to perform adaptive maintenance so that the system may conform to it. Indeed it is possible that this may occur during the development phase as well due to changes in requirements. Almost as important as the ability to exist as the desired modified system, is for there to be a viable way to go from the previous version to the next one. An example of this situation is the need to upgrade the software in all the road side controllers. The manual change of each one individually is likely to be extremely expensive and

error prone, whereas an automatic update via a communication network is likely to be easier to perform and control.

Some things are easier to change than others. The advent of advanced relational data-base management systems permits the ready modification of most data bases. However this flexible data transport over time is not matched by a corresponding flexibility of transport over space. Data communication message formats are far more difficult to modify because at least two sub-systems are effected. Thus the flexibility of data storage may be rendered ineffective by an inflexible data communication system.

ITSs are unusual in that they are likely to expand considerably, both functionally and geographically, during their lifetime. An ITS system architecture must not only provide the flexibility needed to perform this expansion, but it must also be capable of supporting any possible final size (see Section 5.5). Whilst this aspect is normally only considered with respect to the "working" objective of the system, it is just as important for the "workable" objective, since it is perfectly possible to modify a system that was originally both manageable and maintainable in a manner such that the result loses one or both of these attributes.

6.9 Level 0 - Design

6.9.1 Standardisation

There are a number of standards, draft and pre-standards, as well as guidelines which may be of benefit to any level of architecture, in particular Levels 1 and 0. The road transport projects should contact the project CODE for advice on this matter (see also [Gaillet 1996]).

6.10 Legacy Systems and Migration

Few ITS will be created on a "green field" site, and it will therefore be necessary to combine some of the old systems with the new system, or to combine two or more existing systems, together with some enhancements, into a single system. It should be remembered that all systems have a system architecture, even if it has not been formally written down (see Section 4.1). The problem can be understood by referring to Figure 6.7.

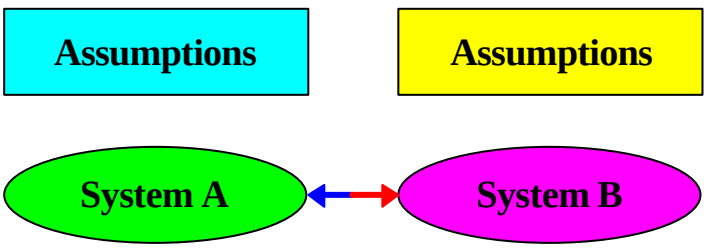


Figure 6.7 - Incompatible Systems

Figure 6.7 shows two systems, A and B, which are intending to work together as an integrated system. Whilst is obvious that it is necessary to have an agreed communications protocol, this is not the only requirement for a working and workable system. The many civil wars

throughout history demonstrate that it takes more than an agreed way of communicating for people, or systems, to work together properly; they must also have an agreed set of assumptions as to what is required of each other. It is therefore necessary to have both a common communications protocol **and** a common set of assumptions about the manner in which each system will work in order to produce the working and workable system shown in Figure 6.8.

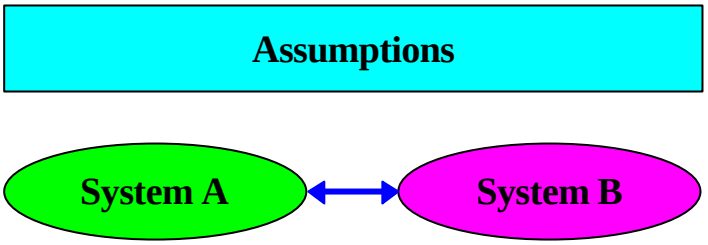


Figure 6.8 - Compatible Systems

The problem when trying to integrate to an existing system is that either the communications interface and/or the built in assumptions may not be known. There are therefore two basic methods of integration.

6.10.1 Compatibility

One obvious way to achieve compatibility is to discover the system architecture of the old system and then to add the new functions in a manner that conforms to this system architecture. This may not be as difficult as it might at first seem. It has always been recognised that communications protocols must be well documented, so there will only be a problem if either the documentation has been mislaid or if the original manufacturer refuses to divulge the information. Identifying the underlying assumptions that are built into the old system may be more difficult because they are rarely documented in an obvious manner. However, it is only necessary to know the relevant assumptions that directly affect the other system(s), and it may well be feasible to identify what they are.

There is, however, a fundamental problem with this approach. By definition, the new system will be created with the architecture of the old system, and it will not be possible to provide features that are not supported by this system architecture. Thus, whilst it may be considered to be the "easy" approach, in the long term the decision to conform to an originally unplanned system architecture may be regretted.

6.10.2 New system architecture with interface bridge

The most likely scenario is for there to be a need to use some existing systems as part of a new, and larger, system. The new system will have a properly planned system architecture, whilst the existing system will have been built up in an ad hoc manner. The assumptions built into existing system are likely to be at best unplanned, and at worst, incompatible with some of

the features needed by the new system. In addition it is possible that the original communications protocols will not fully support the new desired functionality.

In these circumstances the solution is to create a new system architecture that is in accordance with the current user needs. This architecture is then analysed with a view to getting a best fit for the existing systems into that architecture. It may be necessary to modify the new system architecture if the fit is not good enough. In these circumstances the changes should be the minimum necessary because they may curtail the ability to satisfy certain user needs: if at all possible, they should be restricted to the Level 1 Architectures. The new parts of the system will be designed to conform to this architecture, but it may be necessary to create a "bridge" at the interfaces to (some of) the old equipment so that they are seen to operate in accordance with the new system architecture, possibly with reduced functionality. The advantage of this approach is that, in the future, the old components can be replaced by new equipment that conforms to the new architecture and can thus be connected directly to the new system. Eventually a completely new system comes into operation and the bridge can be removed.

7. THE ARCHITECTURE ASSESSMENT PROCESS

Whilst Figure 6.1 concentrates on the development processes, when the system life-cycle is viewed using the "V model" (see Figure 7.1) the checking processes that need to be undertaken are highlighted. Each step on the right hand side of the V is equivalent to one on the left hand side, thus is it against the system design that the system integration and testing are carried out, etc.. In addition it is possible to carry out verification processes to confirm that the output of each successive phase is in conformance to the requirements of the outputs of the previous phase. The architecture assessment process described in this section is made up of the verification processes needed to ensure that the System Architecture conforms to the User Needs. Appendix O contains some advice to those who have to review a system architecture Deliverable.

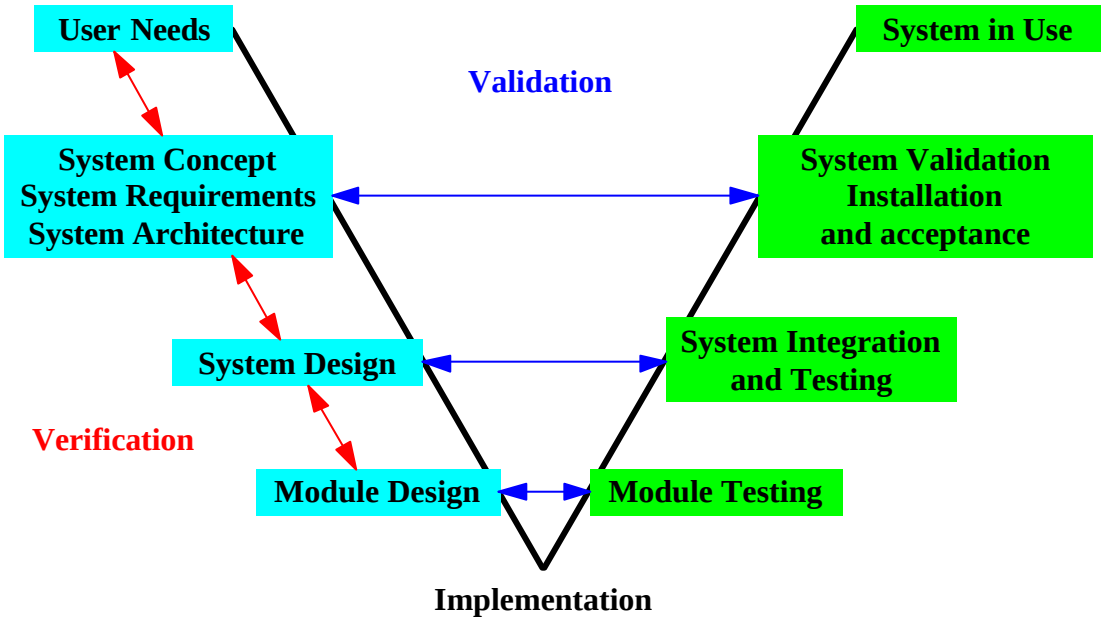


Figure 7.1- Simple "V" Life-cycle Model

7.1 Objectives

The objectives of the assessment process are:

- To confirm the functions necessary to provide the ITS services required;
- To identify to what extent the architecture contains these functions and supports the necessary interconnections to provide these services.
- To identify the extent to which a given ITS possesses redundant links and functions. (Note that this may be intentional for either safety and/or reliability reasons).
- To identify any missing or key links between functions that should be provided in order to produce the selected services.

- To confirm the ability of the architecture to support modifications to its components and/or changes to its environment.
- To confirm the ability of the architecture to lead to implementations that are feasible technically, politically, financially etc.
- To confirm the ability of the architecture to lead to implementations that can be managed, maintained and useable under degraded modes of operation, and to be safe and/or secure.

7.2 Overview

The assessment process consist of four different types of procedure:

- **Verification** - the comparison of the output of each individual phase of the development with the output of the previous phase, the objective being to ensure that the output from the new phase fulfils the requirements specified in the outputs of the previous phase. Verification always requires a comparison to be made (e.g. test results with expected results).
- **Analysis** - the use of a technique to demonstrate the properties of particular products of the life-cycle.
- **Testing** - The process of supplying a set of inputs, process conditions and expected results with the intention of finding faults in a product. (Note that testing can only be used to verify correctness if, and only if, all possible process states can be tested: this situation is extremely unlikely for a telematic system).
- **Validation** - the demonstration that a product satisfies its requirements. Validation requires a decision to be made based on the results of the verification processes and reviews, as well as of the tests performed on the system as it is being integrated.

We have assumed that organisations will have their own review procedures, possibly within a formal quality plan; we have therefore not specified how the following tasks should be organised or managed.

The results of the verification, analysis, test and validation procedures provide evidence to the assessment process. Ultimately a decision must be taken to accept the system architecture before proceeding to the Design phase of the life-cycle based on the contents of the System Architecture Assessment Report.

7.3 Project System Architecture Assessment Report

The System Architecture Assessment Report should provide the evidence and justification (or otherwise!) as to why the architecture will be suitable for the system being proposed by the project. The Report should contain summaries of each of the activities described below, providing information both on how they were done and the main conclusions, in sufficient detail

to justify to the relevant authorities a decision to proceed with the further development of the system. Each project should therefore decide on what type of report will be suitable to their needs. A possible contents list for the System Architecture Assessment Report is given in Appendix B.

7.4 User Needs ® System Concept

The system architecture phase of the life-cycle begins after the User Needs have been identified. It is these Needs that become the driving force of the remainder of the life-cycle, and against which any validation is performed.

7.4.1 Vision Statement

The Vision statement should be checked to confirm that it embraces all the User Needs.

7.4.2 Identification of the Users

A check must be made to ensure that all the users associated with the system have indeed been identified (see Section 6.2.2).

7.4.3 Mission Statement

The Mission Statement should be checked to confirm that it embraces all intellectual, technical, financial, political, etc. attributes necessary to produce a system that will meet the User Needs.

7.4.4 The System Boundary

It is vital that a consistent system boundary is used throughout the creation of the system architecture and the design. It should be possible to identify that part of "anything and everything" that lies inside the system, and thus be subject to the system architecture, and the remainder that lies outside the system, and hence may be affected by, or have an influence on, the system.

The first definitive statement of the system boundary is likely to be as part of the Context Diagram (see Section 7.6.1), but it should be checked at this stage to confirm that it is still consistent with the User Requirements, Vision and Mission Statements before proceeding to the next stage in the assessment. This is to help ensure that there will be self consistency between the system requirements and the system boundary, in a cost effective manner.

7.5 System Concept ® System Characteristics

The system characteristics are derived from the User Needs and the System Concept. A checklist of those issues that should have been considered can be found in Appendix D.

It should be noted that at this stage in the assessment the most important attributes are *completeness* (e.g. that all the identified Users have been considered) and *feasibility*. The System Characteristics may not be consistent and will almost certainly be written at differing levels of detail; this does not matter.

7.6 System Characteristics ® System Requirements

The System Requirements are formal statements about the objectives of the system. The primary system requirements are a direct consequence of the System Concept and the System Characteristics. The derived system requirements are those that are necessary in order to create a working and workable system. The three broad categories of System Requirements are described in Section 6.4.

As distinct from the System Characteristics the System Requirements should be both *consistent* and *complete*. During the process of transforming the System Characteristics into the System Requirements inconsistencies and omissions must be identified and reconciled.

The process of assessing the System Requirements should proceed in two stages:

a) *Verification* - against the System Characteristics, with the inconsistencies having been resolved.

b) *Validation* - against the User Needs.

Specific issues related to the three types of System Requirements can be found in the following Appendices:

- Context Requirements - Appendix E
- Functional Requirements - Appendix F
- Non-Functional Requirements - Appendix G

The System Requirements should be reviewed to ensure that each issue has at least been considered at this stage, the degree to which they have been successfully implemented in the system architecture will be assessed later (see Section 7.8).

7.6.1 System Context Diagram

There should be a Context Diagram showing the system, its terminators and the flow of data between them. A number of analyses can be performed

a) **System Boundary** - the terminators mark the boundary of the system.

b) **Completeness Check** - a crude, but effective completeness check can be performed by confirming that:

- What comes out will satisfy the User Needs.

c) **Consistency Check** - a crude, but effective completeness and consistency check can be performed by confirming that:

- What goes in is used to create what comes out.
- What comes out can be produced from what goes in.

It is important that both checks are performed because they are not the inverse of each other, as might be thought at first sight.

7.6.1.1 Preliminary Safety Analysis

One form of context diagram, but one which cannot be used directly as the top level for functional decomposition, is a PASSPORT Diagram [PASSPORT D9] (see also Appendix H). This has been proved to be particularly useful as the first target for the analysis of an ITS system.

The PASSPORT Diagram was specifically designed to provide a target for the Preliminary Safety Analysis (PSA) of ITS systems (see Appendix H). From this analysis it is possible to derive high level safety functional requirements. The initial estimate of the Safety Integrity Level (SIL) will provide the initial safety integrity requirements, which will become part of the non-functional requirements.

7.7 System Requirements ® Reference Models

There may be a number of Reference Models at Level 2 and Level 3, this will depend upon how many sub-systems are to be integrated, and upon the number of authorities that have control over some aspect of the system. A 'simple' system entirely under the control of a single authority is unlikely to need a Level 3 Architecture. The assessment of the Reference Models should proceed in two stages.

7.7.1 Analysis of Goal-Oriented Functions

The following checks can be performed on layered Reference Models:

- Each layer may take input data for transformation from lower layers or from the environment outside the system.
- Each layer may generate commands for itself or for lower layers.
- Output data may be sent to lower layers for display or transmission to the environment outside the system, but not for transformation.
- Certain sets of lower layers should be able to operate without the availability of the layers above them, albeit with reduced functionality. If this cannot happen then future maintenance may be difficult and/or certain safety requirements may not be being met.

7.7.2 Review of Functionality and Behaviour

The Level 3 and Level 2 Architectures will begin to implement some of the System Requirements. A review should be performed on the Reference Models in two stages:

i) *Verification* - against the System Requirements

ii) *Validation* - against the User Needs

7.8 Reference Models ® Level 1 Architectures

The Level 1 Architectures define the overall structure of the system, and how the sub-systems relate to each other. The activities are described in terms of WHAT the various sub-units will do; the Level 0 Architecture, or design, will provide the details of HOW the sub-units will be implemented. There are four main Level 1 Architectures, with possibility of two further architectures for the more complex systems.

- Functional Architecture
- (Control Architecture)
- Information Architecture
- (Enterprise Architecture)
- Physical Architecture
- Communication Architecture

A review of each architecture should be performed in three stages:

i) *Verification* - that it conforms to the Level 2 and 3 Architectures.

ii) *Verification* - against the System Requirements.

iii) *Validation* - against the User Needs.

7.8.1 Functional Architecture

The Functional Architecture describes the conceptual structure of the logical behaviour of the system. In order to maintain control over the potential complexity of this architecture we recommend the use of functional decomposition.

- Context Diagram - this should be the diagram from which all further decomposition takes place.
- Hierarchical Set of Diagrams - Each (sub-)function may be split into a number of (sub-)sub-functions. This process must maintain the self-consistency of the functional architecture, and the use of a Computer Assisted Software Engineering (CASE) tool is recommended.

The CORD Function List [CORD D004-PT3] provides a set of high-level functions and sub-functions for road transport telematic systems, and its use is recommended for all such projects to facilitate comprehension and the identification of commonalities between systems. The CORD Function List is also the basis of the Functional Architecture Analysis Tool (see Section 7.12).

The consistency between the Functional Architecture and Physical Architecture can be checked using the PASSPORT Cross (see Section 7.9)

Traceability Matrix

A traceability matrix should be created which links the User Needs to the (Sub-)Functions that will provide them (see Figure 7.2). This matrix should itself be subject to Review since it will help to provide confirmation that the architecture does indeed achieve all the User Needs.

	Function 1	**Function 2**	**- - -**	**Function N**
User Need 1				
- - -				
User Need M				

Figure 7.2 - User Needs/Functions Traceability Matrix

7.8.2 Control Architecture

The Control Architecture is often incorporated into the Functional Architecture, however, for a complex control system it may be useful to extract this aspect for clarity. The Control Architecture should be subjected to Peer Review and, if possible, animated with a CASE tool.

It should be noted that the Control Architecture must be incorporated into the Functional Architecture if the PASSPORT Cross is to be used (see Section 7.9).

7.8.3 Information Architecture

The Information Architecture is related to the Functional Architecture and, to some extent, the form of the one will depend on the form of the other. Some characteristics can also be influenced by the Physical and Communication Architectures. There are issues associated with the Information Architecture at all three levels of architecture that need to be reviewed.

Level 3 Issues

It is possible for databases to come under the management of more than one authority. The responsibility for the availability and accuracy of the entire system database(s) must be defined in order to provide a stable basis for a working system.

Level 2 and Level 1 Issues

There are four main issues that should be considered for the Information Architecture in terms of how it may be influenced by the requirements of the other architectures.

- Availability - the general maintenance of a database (back-ups etc.) can impinge on its overall availability. Do the requirements of the Functional Architecture imply above 'normal' availability of certain data? '100% availability' is both difficult and expensive to achieve.

- Accuracy - the way that the contents of a database reflect the situation in the real world. There is always an error introduced during the transformation of analogue sensor reading into digital data (from where does the Physical Architecture imply that the raw data is to be obtained?). The frequency with which the contents of the database are changed can effect the timeliness of the data (could the transmission paths implied by the Communication Architecture create unacceptable delays?).
- Distribution - the way that the information is distributed around the system should be described. The simultaneous writing of data by many functions needs a different approach to the simultaneous reading of data. Access to a distributed database can take some considerable time when the indexes and data are in different locations. Increasing use is being made of World Wide Web (WWW) pages; sometimes WWW can be 'World Wide Wait'!
- Security and Privacy - consideration must be given to the security of commercially sensitive data and to the privacy of personal data. Do the architectures present some easy targets for a potential intruder?

Level 1 Information Architecture

The Level 1 Information Architecture describes the structure of each individual database using, for example, Entity-Relationship Diagrams (ERD). The use of a CASE tool is recommended to ensure completeness of this architecture, and consistency with the Functional Architecture.

7.8.4 Enterprise Architecture

A large and/or complex ITS may need to have an Enterprise Architecture described for the system. The structure of this architecture should reflect the high level Functional Architecture so that the division of responsibility takes place in a natural manner, with no gaps.

7.8.5 Physical Architecture

The Physical Architecture describes the allocation of the physical units that will perform the functions in the Functional Architecture, and the communication paths between them. The use of a systematic or structured approach is recommended, e.g. [Hatley 1987].

The consistency between the Physical Architecture and the Functional Architecture can be checked using the PASSPORT Cross. It is also possible to identify critical and redundant elements (see Section 7.9).

7.8.6 Communication Architecture

The Communication Architecture describes the characteristics of the various channels that have been identified in the Physical Architecture. Each channel must be checked against the corresponding part of the Information Architecture to ensure that it has the capacity to enable the required data to be transmitted and/or received in the time needed.

7.9 Checks for Consistency

It is possible to check the consistency between the Function Architecture (including the Control Architecture) and the Physical Architecture by using the PASSPORT Cross [PASSPORT D9] (see Appendix P). The check is done by performing a set of increasingly rigorous tests in order to build up confidence. All these tests can be performed manually for a small system, but as the size increases so does the effort required, and it may only be possible to perform the more simple tests for a large system. A CAE tool is currently under development in the ESPRIT project COMPASS.

Once the PASSPORT Cross has been created it can also be used to identify redundant links, functions and physical units. This is also described in Appendix P.

7.10 Benefit Analysis

There are many attributes associated with a system architecture which are not strictly associated with the functions that provide the *working* properties. The Benefit Analysis investigates the *workable* properties of the system architecture.

Benefit Analysis can be approached in one of two possible ways:

- *Qualitative Benefit Analysis* - Each issue is considered in turn and a decision is taken as to whether the system architecture is acceptable in that property or not.
- *Semi-Quantitative Benefits Analysis* - Each issue is considered in turn and scored on a scale of 0-10, say, according to some criteria (these criteria will tend to be specific to the application). It is then possible to produce a weighted average of these scores to produce a figure for the system architecture as a whole (the weights chosen will tend to be specific to the application and the politics of the project). On its own this figure has no meaning but if, by changing one or more features in the system architecture, it is possible to vary one or more of the scores then the change in the average can give an indication as to whether the new system architecture is better or worse than the previous version (see also Section 7.12).

Appendix Q contains lists of issues that should be considered when performing a behavioural analysis on the system architecture.

7.11 Cost-Benefit Analysis

The goal of cost analysis is to produce a high level estimate of the expenditure associated with implementing the physical elements of the architecture. A detailed analysis would also include discussions about funding sources and the expected pay-back period. Performing this analysis at the system architecture level is difficult unless the project selects a practical implementation scenario and takes design decisions. This has been done, for example, by the Americans in

their ITS National Architecture [USA ITS] where they have compared four architectures by choosing common scenarios and design decisions.

However, a project may have a need at an early stage of the system development, for example during the system architecture definition, to **justify** that the envisaged system is likely to lead to a successful implementation; meaning, among other things, that it is cost-effective to implement. It is then necessary to perform an initial cost-benefit analysis trying to answer the following question: **What benefits would I get for what investments?**

It is necessary to identify what elements of the architecture would require a high investment, especially if the system has to be built from scratch, and what are the potential benefits of the implemented system. This implies that some design decisions will have to be taken but does not necessarily mean that a complete design has to be done, because the analysis can be done at a more qualitative level.

To help performing this task, cost and benefit matrices (see Figure 4.1 and Figure 7.4) can be built, with costs and benefits being expressed in terms of both tangible and intangible qualities [EUROBUS].

Examples of non-recurring tangible costs include:

- Purchase of hardware, software
- Staff time of planners, system engineers
- Training of staff

Examples of recurring tangible cost include:

- Maintenance costs
- Communication costs

Examples of non-recurring intangible cost include:

- Costs due to delays

Costs	Tangible	Intangible
Non-recurring		
• sub-system A		
– Purchase	Low	Low
– maintenance	High	Low
–		
• sub-system B		
– Purchase		
– maintenance		
– ...		
• ...		
Recurring		
• sub-system A		
• sub-system B		
• ...		

Figure 7.3 - Cost Matrix

Examples of recurring intangible costs include:

- Costs due to a non-standard system requiring special attention
- Lack of accurate information for planning and decision making

Benefits	Tangible	Intangible
Improve Safety		
Increase Economic Productivity		
Reduced Energy Use		
Enhancement to the Environment		
Improved Mobility/Accessibility		
Increased Efficiency		
Improved Quality of Life		

Figure 7.4 - Benefit Matrix

Examples of tangible benefits include:

- Improved mobility, maintenance
- Reduction of costs

Examples of intangible benefits include:

- Increase staff and customer confidence

- Better quality of service

Once these matrices have been defined, an assessment can be performed based mainly on the project's team experience.

7.11.1 Market Analysis

The question we try to answer is: "what products could I use for implementing the architecture, and is there an opportunity for me to develop, from my architecture, a product that I could sell".

To bring elements of answer to this question three steps, as a minimum, have to be undertaken:

1. What are the current characteristics of the market and mainly:
   - What products are available?
   - At which cost?
   - Can I purchase them easily (good distribution)?
   - Does the manufacturer offer after-sales services?
   - etc.
2. Then there is a need to characterise the market in the future and identify:
   - What are the opportunities in the market?
   - What are the threats?
3. Do I need standards to develop my products? Are these standards available?

So far, R&D projects do not normally perform this type of study. However, if an R&D project intends to launch products on the market or to use products off the shelf, then this type of analysis may be necessary.

7.12 Functional Analysis

The CONVERGE-SA project has developed a Functional Architecture Analysis Tool, which has been design to give:

- System designers
- System integrators
- System owners

the ability to estimate the potential performance of ITS services while the architecture is being developed. In particular:

- The **impact** of a certain choice of architecture, or the certain choice of functions, can be assessed

- The expected **performance differences** between various possible architectures can be investigated
- **Sensitive elements** of an architecture can be identified, i.e. when small changes in one section cause large variations in the expected responses.

An overview of this tool can be found in Appendix R.

8. REFERENCES & BIBLIOGRAPHY

System Architecture is a new subject area and there are only a limited number of books that address the subject. Unfortunately some of them are not readily available in some countries. This list will be updated during the programme. The Books marked with a star(*) are highly recommended.

[CORD D004-PT3]
Per-Olof Ryd (Ed), *Recommended Definitions of Transport Telematics Functions and Sub-functions*, DRIVE II project CORD (V2056), Deliverable Nº D004 - Part 3, 1994. (A new version is being prepared)

[CORD D004-PT6]
Gaillet J-F (Ed.), *Recommended Methodology for Transport Telematics Architectures*, SATIN Task Force, DRIVE II project CORD (V2056), 1994.

[CORDEX AC23]
CORDEX, *DATEX-Net Specifications for Interoperability - Version 1.1 - Annex 2: DATEX EDIFACT Messages*, DRIVE II project CORDEX, 1996.

[Crowe 1996]
M. Crowe, R. Beeby and J. Gammack, *Constructing Systems and Information: A Process View*, McGraw Hill, 1996, ISBN 0-07-707962-0.

[DeMarco 1978]
DeMarco T, *Structured Analysis and System Specification*, Prentice Hall, 1978.

[EMCATT 1995]
EMCATT, *Functional System Safety and EMC*, DRIVE II Project EMCATT (V2064, 1995.

[EUROBUS]
EUROBUS, *Framework for cost benefit analysis*,

[Franco 1997]
Franco G and Jesty P H, *Architecture Analysis Tool Getting Started Manual*, Framework IV Transport Telematic Project CONVERGE (TR1101), Deliverable DSA4.2, 1997.

[Gaillet 1996]
Gaillet J-F, *CEN/TC 278 Framework Part 1: Description of the Different Working Groups*, ERTICO, 1996.

[Gall 1986]
Gall J., *Systemantics: The Underground text of Systems Lore*, The General Systemantics Press, 1986, ISBN 0-9618251-0-3.

[Giezen 1996b]
Giezen J, Jesty P H and de Bruijn M, *System Architecture: The Control of System Behaviour*, Proceedings of the 3rd World Congress on ITS, 1996.

[Hatley 1987]*
Hatley D J and Pirbhai IA, *Strategies for Real-Time System Specification*, Dorset House, 1987, ISBN 0-932633-11-0.

[Hatton 1997]
Hatton L, *Software Failures: Follies and Fallacies*, IEE Review, March 1997.

[Hice 1991]
G. F. Hice, *DSI: Distributed Systems Integration*, CAP Gemini Publishing BV, 1991, ISBN 90-71996-27-1.

[Hice 1992]
G. F. Hice, *MISE: Managing Information Systems Evolution*, CAP Gemini Publishing BV, 1992, ISBN 90-71996-54-9.

[Hitchins 1992]
D. K. Hitchins, *Putting Systems to Work*, Wiley, 1992, ISBN 0-471-93426-7.

[Holloway 1988]
Holloway S, *Data Administration*, Gower Technical Press, 1988.

[Inmon 1992]
W. H. Inmon and J. H. Caplan, *Information Systems Architecture: Development in the 90's*, QED Information Services, 1992 ISBN 0-89435-410-8.

[ISO 9126]
ISO 9126, *Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use*, 1991.

[ISO 10746]
ISO/IEC 10746-1, *Reference Model of Open Distributed Processing (Committee Draft),* JTC1/SC21/N, 1994.

[ISO 11179-1]
ISO 11179-1, *Framework for the Specification and Standardisation of Data Elements.*

[ISO 11179-2]
ISO 11179-2, *Classification for Data Elements.*

[ISO 11179-3]
ISO 11179-3, *Basic Attributes of Data Elements.*

[ISO 11179-4]
ISO 11179-4, *Rules and Guidelines for the Formulation of Data Definitions*.

[ISO 11179-5]
ISO 11179-5, *Naming and Identification Principles for Data Elements.*

[ISO 11179-6]
ISO 11179-6, *Registration of Data Elements.*

[ISO CIM]
ISO TC184 SC5 WG1, *Reference model for Floor Shop Production Standards part 1: A Reference Model for Standardisation and a Methodology for Identification of Standards Requirements.* Technical Report, 1989.

[ITS Eval]
ITS America, *ITS Architecture Evaluation Plan*, June 1995. See also [USA ITS]

[ITSEM 1993]
DG XIII, *Information Technology Security Evaluation Manual (ITSEM)*, September 1993.

[Jesty 1996a]
Jesty P H and Giezen J, *Its Architecture Jim, But Not as We Know It!*, Traffic Technology International, June/July 1996.

[Jesty 1996b]
Jesty P H and Giezen J, *System Architecture: Flexibility, Management and Maintenance*, Proceedings of the 3rd World Congress on ITS, 1996.

[Jesty 1997]
Jesty P H, Giezen J and Fowkes M, *System Safety Guidelines*, Framework IV Transport Telematic Project CODE (TR1103), 1997.

[MISRA 1994]
Motor Industry Software Reliability Association, *Development Guidelines for Vehicle Based Software*, MIRA, 1994, ISBN 0 9524156 0 7. (Available from D Ward, MIRA, CV10 0TU, UK)

[Newton 1993]
Newton J and Wahl D C, *Manual for Data Administration*, US Department of Commerce, National Institute of Standards and Technology Special Publication 500-208, 1993 (Available from Superintendent for Documents, US Government Printing Office, Washington, DC 20402, USA)

[Neumann 1995]
Neumann P G, *Computer Related Risks*, Addison Wesley, 1995, ISBN 0-201-55805-X.

[PASSPORT D8]
PASSPORT, *Towards the Certification of ATT Systems: System Safety Aspects*, , DRIVE II Project PASSPORT (V2057/8), Deliverable Nº 8 , 1995.

[PASSPORT D9]
PASSPORT, *Framework for Prospective System Safety Analysis*, DRIVE II Project PASSPORT (V2057/8), Deliverable Nº 9, 1995.

[Petroski 1982]
Petroski H, *To Engineer is Human*, St Martins Press, 1982, ISBN 0-312-80680-9.

[Priestley 19??]
Priestley M, *Practical Object Oriented Design*, McGraw Hill, ,ISBN 077091760.

[QUARTET D53]
QUARTET, *Final Methodology for Architecture Assessment*, DRIVE II Project QUARTET (V2018), Deliverable Nº 53, 1995.

[Rechtin 1991]*
E. Rechtin, *Systems Architecting: Creating and Building Complex Systems*, Prentice Hall, 1991, ISBN 0-13-880345-5.

[Rumbaugh 1991]
Rumbaugh J, Blaha M, Premerlani W, Eddy F and Lorensen W, *Object-Oriented Modelling and Design*, Prentice Hall, 1991, ISBN 0-13-630054-5

[SATIN]
The SATIN documents are listed at the end

[Sommerville 1992]*
I. Sommerville, *Software Engineering (4th Ed.)*, Addison-Wesley, 1992, ISBN 0-201-56529-3.

[Thomé 1993]*
B. Thomé, *Systems Engineering: Principles and Practice of Computer-based Systems Engineering*, Wiley, 1993, ISBN 0-471-93552-2.

[USA ITS]
ITS America, *The National Architecture for ITS*, see “www.rockwell.com/itsarch/” or “www.itsa.org/public/archdocs/national.html”.

[Wiener 1993]
Wiener L, *Digital Woes: Why We Should Not Depend on Software*, Addison Welsey, 1993.

[Zimmerman 1980]
Zimmerman H, *OSI Reference Model - the ISO Model of Architecture for Open Systems Interconnection*, IEEE Transactions on Communications **COM-28**(4) pp. 425-432.

SATIN Documents

The following documents were produced by the SATIN Task Force and are available from the Commission. The relationship between each other is shown in Appendix S.

[SATIN AC13-PT1]
Bonora S, *ATT System Architecture Developments: the Automatic Debiting System (ADS) Area*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT2]
Both M, *ATT System Architecture Developments: the Freight and Fleet Management and Hazardous Goods Monitoring (FFM and HGM) Areas*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT3]
Casimir C and Helcmanocki N, *ATT System Architecture Developments: the Traffic and Travel Information (TTI) Area*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT4]
Roach H, *ATT System Architecture Developments: the Public Transport Area*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT5]
Giezen J and Blonk J, *ATT System Architecture Developments: the Inter-Urban Traffic Management (IUTM) Area*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT6]
Wrathall C, *ATT System Architecture Developments: the Urban Traffic Management (UTM) Area*, SATIN Task Force, DRIVE II Project CORD (V2056), 1994.

[SATIN AC13-PT7]
SATIN, *Proposals for Urban, Inter-Urban and In Vehicle Architectures*, SATIN Task Force, DRIVE II Project CORD (V2056), 1995.

[SATIN AC13-PT8]
SATIN, *Interfaces Between the IRTE Areas*, SATIN Task Force, DRIVE II Project CORD (V2056), 1995.

[SATIN AC13-PT9]
Gaillet J-F, *Highlights of Transport Telematics Architecture Concepts and Results*, SATIN Task Force, DRIVE II Project CORD (V2056), 1995.

[SATIN-AC20]
Giezen J, Blonk J and Jesty P H, *Reference Models and their use in Architecture Development*, SATIN Task Force, DRIVE II Project CORD (V2056), 1995.

[SATIN D007-PT1]
Wrathall C, *High Level User Requirements and Quality Factors for a European IRTE*, SATIN Task Force, DRIVE II Project CORD (V2056), 1995.

[SATIN-AC18]
Blonk J, Blachère J and Gaillet J-F, *Review of the DATEX Dictionary, a SATIN contribution to DATEX*, SATIN Task Force of CORD (V2056), November 1995.

APPENDIX A PROPOSED SYSTEM ARCHITECTURE DELIVERABLE CONTENTS LIST

1 System Concept

1.1 Vision Statement

A brief statement of WHAT the system should achieve, clearly defining the system boundaries (refer to Section 6.2.1).

1.2 Identification of Users

All persons affected by, or who will have an effect on the final system (refer to Section 6.2.2).

1.3 Mission Statement

A brief statement of HOW it is proposed to produce the system (refer to Section 6.2.3).

2 System Characteristics

A description of all properties that the final system should exhibit. This should be generated by all those who will have an affect on, or will be affected by, the system (refer to Section 6.3).

3 System Requirements

3.1 Context Requirements

An expression of assumptions about the system environment together with policy statements and strategic and tactical considerations for the development and/or deployment of the system (refer to Section 6.3 and Appendix E).

3.2 Functional Requirements

Requirements defining the type of service expected from the system, fully covering all functional elements in the system (refer to Section 6.3 and Appendix F).

3.3 Non-Functional Requirements

Requirements referring to intrinsic quality requirements of the proposed architecture (refer to Section 6.3 and Appendix G)

3.4 Context Diagram

The first engineering drawing of the system showing the information exchange with terminators representing data sources and sinks. This will allow completeness and consistency checks to be performed (refer to Section 6.5).

4 Level 3 and 2 System Properties - Reference Models

Models concerned with the control domain allowing zones of jurisdiction to be delineated. Level 2 refers to single Authority control whilst Level 3 refers to multiple Authority control in which a consensus must be obtained (refer to Section 6.6).

5 Level 1 Architectures

5.1 System Structure - Functional

Functional Architecture

Describes the various sub-functions of the system and the data flow between them (refer to Section 6.7.2).

Control Architecture

Describes the flow of control signals between sub-systems (refer to Section 6.7.2).

Information Architecture

Comprises data models for the sets of data that have been identified (refer to Section 6.7.2).

Enterprise Architecture

For a system having many types and layers of management this ensures compatibility with the allocation of responsibility shown in the level 2 and 3 reference models (refer to Section 6.7.1).

Physical Architecture

Describes the allocation of physical units, and the communication paths between them, required to perform the functions of the Functional Architecture (refer to Section 6.7.4).

Communication Architecture

Describes the channel characteristics identified within the Physical Architecture (refer to Section 6.7.4).

5.2 System Structure - Behavioural

Describes those features which are concerned with the "manner" in which the system operates and which together create the "workable" attribute of the System Architecture (refer to Section 6.8).

APPENDIX B PROPOSED SYSTEM ARCHITECTURE ASSESSMENT DELIVERABLE CONTENTS LIST

1 User Needs

A re-statement of the User Needs which have been identified for this system.

2 Review of System Concept

A summary of the results of the review of the System Concept:

- Vision Statement (see Section 7.4.1).
- Identification of the Users (see Section 7.4.2).
- Mission Statement (see Section 7.4.3).
- System Boundary (see Section 7.4.4).

3 System Characteristics

A summary of the results of the review of the System Characteristics (see Section 7.5).

4 System Requirements

A summary of the results of the review of the System Requirements (see Section 7.6):

- Context Requirements
- Functional Requirements
- Non-Functional Requirements

4.1 Preliminary Safety Analysis

A summary of the Preliminary Safety Analysis (see Appendix H); details may be placed in an Appendix.

5 Reference Models

A summary of the analysis (see Section 7.7.1) and review (see Section 7.7.2) of the Level 3 and Level 2 Reference Models.

6 Level 1 Architectures

A summary of the review of the Level 1 Architectures:

- Functional Architecture (see Section 7.8.1).
- Control Architecture (see Section 7.8.2).
- Information Architecture (see Section 7.8.3).

- Enterprise Architecture (see Section 7.8.4)
- Physical Architecture (see Section 7.8.5)
- Communication Architecture (see Section 7.8.6)

A summary of the analyses of the Level 1 Architectures:

- Traceability Matrix (see Section 7.8.1)
- Checks for Consistency (see Section 7.9)
- Functional Analysis (see Section 7.12)

7 Benefit Analysis

A summary of the Benefit Analysis (see Section 7.10)

8 Cost - Benefit Analysis

A summary of the Cost-Benefit Analysis (see Section 7.11)

APPENDIX C LEVELS OF ARCHITECTURE

The following sections state the differences between the four levels of architecture.

Appendix C.1 Level 0: (Sub-)System Design

Appendix C.1.1 Characteristics and Assumptions

- Single expertise domain for each component
- Standard development procedures
- Fixed goal
- Full set of specifications
- Applicable standards
- Full quality control and product certification

Appendix C.1.2 Objectives

- Marketable product
- Functionality (goal oriented)
- Testability
- Performance and reliability
- Component flexibility (to meet different goals)
- Component maintainability
- Application of standards

Appendix C.1.3 Methods, Techniques and Presentation

- Domain-specific education and training
- Standard, domain specific methods, techniques and presentations
- Use of widely available, standard, domain specific tools, some of which are sophisticated and require expertise

Appendix C.2 Level 1: System Development

Appendix C.2.1 Characteristics and Assumptions

- Related domain expertise
- Semi-standard development procedures
- Single non-conflicting goal

- Possible growth of data, traffic, users etc.
- Negligible evolution or expansion in any volatile dimension
- Full set of system requirements
- Distribution and/or replication of functionality
- Process certification

Appendix C.2.2 Objectives

- Functionality: goal oriented and supporting
- Integration of components
- Structuring
- Flexibility
- Maintainability
- Configurability
- Reliability and availability
- Performance, efficiency and effectiveness
- System testability
- Integration of data and messages
- Configuration management
- Problem management
- Consolidation of desired emergent properties
- Predictability of system behaviour
- System safety, security and EMC
- Definition of system dictionary

Appendix C.2.3 Methods, Techniques and Presentation

- Education, experience and training in system engineering
- Methods, techniques and presentations as used in the system engineering domain
- Use of some dedicated system engineering tools requiring moderate expertise, which produce results which can be understood by all participants

Appendix C.3 Level 2: Single Authority Control Regime Development

Appendix C.3.1 Characteristics and Assumptions

- Non-related expertise domains; multi-disciplinary
- Non-standard, partly described, dedicated procures
- Multiple and conflicting goals (intra-control regime)
- Single authority
- Tentative set of functional requirements
- System evolution (hardware, functionality, knowledge)
- System expansion (functional, geographical, user types)
- Standards virtually absent, some to be defined
- Validation procedures

Appendix C.3.2 Objectives

- Functionality: goal oriented, then supporting
- Integration of systems
- Intra-goal conflict resolution
- Responsibility
- Complexity management
- Development of Reference Model(s) for structuring and simplicity
- Intra-consistency
- Flexibility (at Levels 1 and 0)
- Stability (especially for control systems)
- Controllability (for control systems)
- Maintainability; Operability; Manageability
- Degradability; Local autonomy; Recoverability
- Upgradability
- Effectiveness
- Predictable behaviour
- Integration of data definitions in system dictionary

Appendix C.3.3 Methods, Techniques and Presentation

- Education, training and experience in system architecture
- Methods, techniques and presentations in system engineering and system architecting (especially layered models)
- Some dedicated tools for model description, coherence and consistency analysis, which produce results which can be understood by all participants

Appendix C.4 Level 3: Multi-Authority Control Regime Development

Appendix C.4.1 Characteristics and Assumptions

- Non-related expertise domains; multi-disciplinary
- Non-standard, scarcely (or not at all) described, dedicated procedures
- Multiple authorities, shared responsibility
- Superficial knowledge of ultimate functionality
- Conflicting goals (inter-control regimes, especially commercial isues)

Appendix C.4.2 Objectives

- Functionality: improvement of efficacy of individual goals; emphasis on behaviour
- Development of Reference Model for conflict resolution, inter-operability and inter-control regime consistency
- Integration of data definitions or interpretation/transformation rules
- Upgradability
- Flexibility (at Levels 2, 1 and 0)
- Maintainability (after the integration of the control regimes)
- Allocation of responsibilities, liabilities and rights
- Financial and commercial arrangements

Appendix C.4.3 Methods, Techniques and Presentation

- Education, training and experience in system architecture and human factors
- Primarily textual and elementary figures and diagrams
- Elementary tools

Appendix C.5 Supra-system Level (Context)

Appendix C.5.1 Characteristics and Assumptions

- Non-related expertise domains: multi-disciplinary
- External influence outside control of current stakeholders

Appendix C.5.2 Objectives

- Identification and incorporation of laws, local regulations and other binding rules
- Identification of financial and taxation rules, and ways to deal with those issues
- Identification of standards applicable to the system and the way to apply them
- Identification of the information required by external authorities and how to deal with them
- Identification of future inter-operability with emerging systems, and the determination to deal with it

Appendix C.5.3 Methods, Techniques and Presentation

- Education, training and experience in system architecture and legal issues (pertaining to the subject)
- Primarily language oriented
- No tools available

APPENDIX D SYSTEM CHARACTERISTICS

The system characteristics are derived from the User Needs and the System Concept. The following is a checklist of those issues that should be considered when creating or assessing the system characteristics.

- System overview - a brief statement of what the system will look like, and how it will operate.
- Functional - the principal functions of the system.
- HMI - the way that the Users will use the system.
- Institutional - e.g. the authorities associated with the system.
- Organisational - e.g. the relationship between the operators of the system
- Social - e.g. the acceptability of the system to the final users.
- Legal - any laws that apply or that are needed.
- Data and Information - the main items needed by, and produced by, the system.
- Communications - the main communications needed by the system.
- Safety - obvious safety hazards.
- Security - obvious security issues.
- Electrical and electronics, including power supplies - any special requirements.
- Electromagnetic compatibility - any special requirements.
- Mechanical, including packaging and size - any special requirements.
- Degraded modes of operation - e.g. after a failure, or during maintenance.
- Maintenance - e.g. any special restrictions.
- Future expansion, both functional and geographical - both planned and possible.
- Financial (system deployment) - costs and benefits expected; who will pay for it.
- Financial (payment for services) - e.g. how the user will pay for the service.
- Risk - e.g. technical {can it be done?}; financial {how much will it cost?}

It should be noted that at this stage in the assessment the most important attribute is completeness (e.g. that all the identified Users have been considered). The System Characteristics may not be consistent and will almost certainly be written at differing levels of detail; this does not matter.

APPENDIX E CONTEXT REQUIREMENTS

The context requirements specify the reaction to the constraints imposed by the environment in which the system is to operate. The following is a checklist of those issues that should be considered when creating or assessing the context requirements.

System openness - the architecture should permit the provision of equipment and services from various suppliers within an open, modular and incremental plan.

Temporal - the architecture should support an evolutionary development strategy that enables the continuous upgrading of the system.

Geographical - the architecture should support an evolutionary implementation strategy that foresees an orderly geographic growth of the system.

Institutional - the constraints determined by the allocation of responsibilities, liabilities etc. to parties.

Financial - the development costs need to be balanced against maintenance and operation costs. Overall costs need to be balanced against the expected benefits of the system that can also be expressed in monetary terms.

Social - the acceptability of the proposed system by both users and non-users.

Technical - the availability and suitability of technologies.

Risk - the reduction of the risk of delay or non-completion of critical paths in the development plan.

Infrastructure - the architecture should permit the provision of services on the existing road network, and should be compatible with existing road infrastructures.

Appendix E.1 Examples of context requirements

- The architecture shall permit the provision of equipment and services from various suppliers within an open, modular and incremental development plan.
- The architecture shall support an evolutionary development strategy that enables the continuous upgrading of the system.
- The architecture shall support an evolutionary implementation strategy that foresees an orderly geographic growth of the system.
- The architecture shall support the provision of services in various topographical domains:

APPENDIX F FUNCTIONAL REQUIREMENTS

The functional requirements define the type of service that will be expected from the system and the functions that will provide this service. The functional requirements will be written at various levels of detail. Initially many of them will be at a high level and then, as the life-cycle proceeds, they will be expanded into lower levels of detail; this should, of course, be done under version control. Functional requirements should always be written in a manner such that they are:

- understandable, concise and self-explanatory.
- consistent between each other.
- testable - written formally using 'shall'.
- traceable during the life-cycle.

Functional Requirements can be divided into two main categories

a) **Primary Requirements (or main service provision)** - The services identified by the User Needs and/or the System Characteristics will be implemented by one or more functions. At the high level of definition road ITS developments are advised to use the CORD Function List [CORD D004-PT3] for this purpose.

b) **Derived Requirements** - these are requirements that are needed because of the particular design decision, and can therefore appear at any point in the life-cycle. A particularly important set of derived requirements are the Safety Requirements that have been identified by the Safety Analysis (see Appendix H)

Appendix F.1 Examples of functional requirements

- The architecture shall support the recording of the behaviour of the traffic and transport system.
- The architecture shall support methods for predicting near-future localised traffic conditions.
- The architecture shall support the monitoring of network traffic conditions.
- The architecture shall support the accurate and efficient identification of network incidents, including accidents, lane closures, demand peaks and infrastructure failures.

APPENDIX G NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify the performance and/or quality attributes. Since many high-level non-functional requirements will ultimately be implemented by low-level functional requirements, at this in the life-cycle it is far less important as to whether a requirement has been categorised as *functional* or *non-functional* than that it has been recognised to exist. The following is a checklist of those issues that should be considered when creating or assessing the non-functional requirements (see also [ISO 9126] and [SATIN D007-PT1]).

- **Continuity** - an ITS system is likely to be safety-related. It should therefore operate continuously or at least ensure that unsafe operation does not result from degradation. Non-safety-related systems may need this attribute for commercial reasons.
- **Evaluability** - the ability of the architecture to facilitate the evaluation of the effects of the working system upon its environment, and upon its own operation.
- **Expandability** - the ability of the architecture to support new functions or data items.
- **Extendibility** - the ability of the architecture to support the addition of new locations.
- **Flexibility** - the ability of the architecture to support modifications to its components in order to satisfy changing user requirements. In particular with respect to (see also [QUARTET D53]):
  - functional modules
  - data items
  - message composition
  - changes in the network
  - changes in the infrastructure
- **Human Factors**: the ability of the architecture to produce systems that are easy to use. Projects are referred to the project CODE (TR1103) for support on Traffic Safety and Human Machine Interfaces (HMI) for assistance.
- **Interoperability** - the ability of the architecture to interface harmoniously with other ITE systems outside of the system boundary (see also [QUARTET D53]).
- **Maintainability** - the ability of the architecture to support timely changes to the software or hardware at any location.
- **Proprioception** - the ability of the system to monitor its own performance [CORD D004-PT6].
- **Robustness** - the ability to withstand environmental stress or infrastructure failures.

- **Safety** - to eliminate the possibility that the system might have a negative effect upon its environment (see Appendix H and [Jesty 1997])
- **Safety Integrity** - the measures specified to ensure that all safety functions are performed in order to meet the Safety Integrity Level. They are divided into the measures to avoid, and to control, the effects of systematic and random faults.
- **Security** - the measures specified to protect the system from malicious attack and to maintain the security, integrity and privacy of the data during use, storage and transmission.
- **Survivability** - the ability of the architecture to continue the performance of critical functions when a part of the system is inoperable.
- **Testability** - the ability of the architecture to support verification tests during the development of the system.
- **Time and Space Criteria** - with particular reference to:
  - date transfer quantities
  - computational complexity of functions
  - storage space
  - access times
  - transmission times

Appendix G.1 Examples of non-functional requirements

- The architecture shall ensure the continued provision of benefits under environmental stress or infrastructure failures.
- The architecture shall ensure system safety during degraded mode operation.
- The architecture shall ensure system availability during degraded mode operation.
- The architecture shall allow small changes of enhancement, extension or adaptation without necessitating large redevelopment.
- The specification, design and documentation of the system shall permit the effective application of normal rigorous validation and verification procedures when changes have been made.
- The architecture description shall include a carefully studied risk analysis as a reference for managing the development phase.

APPENDIX H PRELIMINARY SAFETY ANALYSIS

The following is a summary of Preliminary Safety Analysis (PSA) from the Framework for Prospective System Safety Analysis (PSSA) produced by the EC DRIVE II Project PASSPORT (V2057) [PASSPORT D9]. A PSA should be carried out before the detailed design is underway and is based on the information that is available at the time. Since most new systems are extensions of, or make use of, existing systems, the information available at even this concept stage can be quite considerable. The objectives of a PSA are:

- preliminary hazard identification.
- preliminary identification of the safety objectives.
- preliminary identification of the safety requirements.
- preliminary assignment of Integrity Levels.

A hazard is an undesirable effect by the system on its environment. It is therefore necessary to produce a model that shows the system in relation to its environment. One such model is the PASSPORT Diagram which is an extension of Yourdon data-flow diagrams. A PASSPORT Diagram (see Figure H.1) contains the nucleus of the system at its centre with all the possible interactions with its terminators surrounding it. The system boundary is immediately outside the terminators. The type of data passing between the nucleus of the Target of Evaluation is also shown, as are any known databases. The model is specifically designed for systems which contain computers and which interact with other systems. The diagram can be checked for completeness by ensuring that it will perform the functions specified in the system requirements, and also that all the influences on the system ("Static Data") are present (e.g. development process, standards, given data etc.). The diagram can also be checked for consistency by ensuring that "what comes in must go out" and "what comes out must have gone in".

A PSA should be performed by a team of persons with a variety of relevant expertise, including human factors, and once the team is agreed that the PASSPORT Diagram does indeed provide a true representation of the system the hazard analysis can begin. Initially a "what if..?" analysis should be performed on each 'box' in the diagram; this will enable the primary hazards to be identified, from which the safety objectives can be formulated. Each hazard can then be studied further by using a "what causes..?" tree analysis based on the information currently available; this will enable the preliminary safety requirements to be identified. A study of the controllability of each hazard [PASSPORT D9] will enable a preliminary Safety Integrity Level to be assigned for the design and development of the corresponding sub-system.

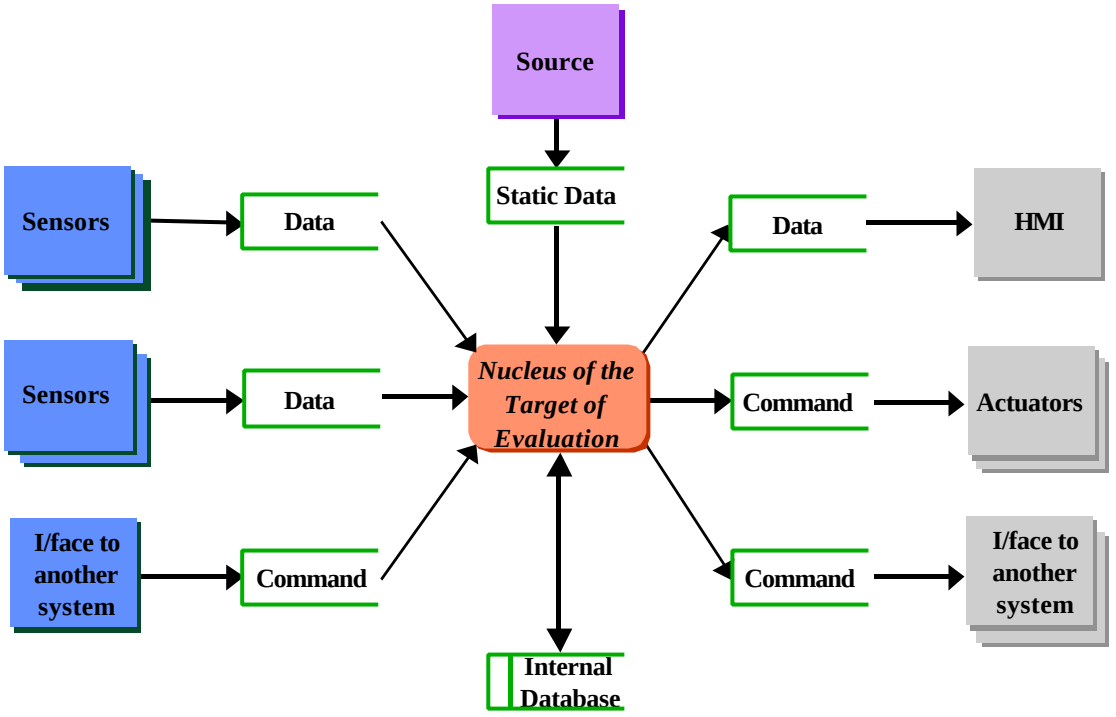


Figure H.1 - PASSPORT Diagram

Appendix H.1 System Boundary

The PASSPORT Diagram in Figure H.1 identifies the system boundary, with the terminators on the Left and Right locating the points where the system interacts with its environment (i.e. at the boundary of, but within, the system). It is important that the system boundary is identified correctly and exactly since it defines what is, and is not, under the influence of the system life-cycle. Not only does this location have commercial consequences, but the precise statements of the safety hazards, and hence the safety requirements, may vary if the system boundary is moved.

APPENDIX I SYSTEM ARCHITECTURE: THE DEVELOPMENT OF THE REFERENCE MODEL

This paper was prepared by Jan Giezen for the 4th World Congress on Intelligent Transport Systems.

Appendix I.1 Introduction

CONVERGE is one of the horizontal projects in the Advanced Transport Telematics (ATT) programme. One of the tasks of CONVERGE is to contribute to the development of the system architecture (SA) for the European ATT systems by means of methodology definition, advice to the projects and activities to amalgamate the architectures of the projects.

One of the theoretical contributions of the CONVERGE-SA to the methodology is a four-level model to describe some of the various issues an architecture has to address, see the Architecture Guidelines, as produced by this project. For an overview, see [Jesty 1996b].

The highest level (level 3 architecture) describes the multi-authority domain. In a large system, especially in the ATT environment, the overall system is not under one authority, but many. Each of these authorities has to agree on certain issues of the system, like the production and availability of data, apportionment of revenues, responsibilities and rights etc. This has to be established in a model and serves as a basis for the system.

The next level (level 2 architecture) represents one single authority, a control regime. This control regime is, within the boundary set by the level 3 architecture, free to develop further structures. However, it must be clear that similar structures for each of the control regimes are advisable. To this end CONVERGE proposed the level 2 reference model, which the projects are advised to apply, especially to structure the functionality of the system and to improve the anticipated behaviour of the system [SATIN AC20].

Unfortunately, it turned out that a considerable number of projects had problems to apply the structure implied by the reference model. Further investigation showed that there were various reasons for this: inadequate theory, insufficient guidelines for development and difficult incorporation in the traditional development life-cycle.

The theory in the guidelines left something to be desired. Partly this was intentionally, to keep the guidelines concise. But this situation was also caused by the then yet incomplete notion of what was needed by the projects, which insight emerged later. This theory should address the rationale and advantages of the proposed structure, compared to the more traditional functional decomposition methods, for instance following data flow diagrams.

The development guidelines should explain how such a reference model can be developed, following the control paradigm, and how this functional result can be enhanced to cater for a variety of organisational and behavioural issues.

Also the incorporation of this activity in the more traditional development life-cycle seemed to be difficult. Current theory books on system development do not normally mention a phase for architecture development and those that do, do not mention this specific reference model, because it is a CONVERGE-SA concept.

This paper is a collation of existing and newly developed theory and practical experience to address this deficiency.

Appendix I.2 Level 2 Reference Model: Disadvantages of Decomposition Techniques

Normally, software systems are designed following a decomposition technique, of which data flow diagrams (DFDs) is one of the best-known examples. (For the case of the subject: object oriented design (OOD) is not fundamentally different in this context, although OOD in itself embodies a wealth of new technology.)

There are, however, quite a number of disadvantages of straight functional composition. These are: the prime focus on functionality, the likelihood of the creation of a fait accompli for many design details, the combinatorial explosion' and the potential of unstructured results (although this contingency is less likely with OOD).

The prime focus on functionality means that many organisational or behavioural issues are not considered. Both DFD and OOD lack adequate accommodation for these issues; these methods are based on functionality, and attention to other things is just lip-service in many cases or is just left to the designer to handle. Yet organisational issues have to be addressed and behavioural issues are a prime concern: even if the system is functionally correct, the system can be rejected because of abject behaviour. This is not just a theoretical contingency; in many cases of system rejection the reason was not the amount of deficiencies in the functionality; these were undoubtedly present but were expected to be correctable. The reason for rejection however was unworkable behaviour, with no prospect of fundamental improvement.

Progression following the functional decomposition paradigm, without proper attention to other issues may easily lead to a fait accompli in various respects: the purely functional approach may obstruct better solutions. For example, whilst a DFD starts with a context diagram without further restrictions, it can be questioned whether at that point a constraint-free start is really the best, or even a realistic, position.

Standard functional decomposition is also haunted by the phenomenon of combinatorial explosion', a fundamental characteristic, not to say measure, of complexity. Regularly, a function is connected with a range of other functions. Needless to say that this adds to the complexity of the result and that this outcome is not at all desirable. Moreover: it is unnecessary, as the following will show. Simple computation will demonstrate that the interactions increase rapidly, more or less exponentially; in the layered model this increase is just linear and obviously more easy to handle.

To people familiar with decomposition techniques it is well-known that the outcome of the process can be quite unstructured. This seems to be a paradoxical situation: whilst the production of DFDs follows a structured method, meaning: with certain rules for diagram production and internal consistency, this does by no means imply that the result, after some layers of decomposition, is really showing any structure at all, apart from the decomposition itself. Partly this can be attributed to the non-existence of refined rules on how to apply decomposition; any division is all right, as far as the method goes, resulting in largely varying results between different designers. Apparently, if the output of the process needs to present a higher-level structure, this structure has to be input to the decomposition process.

The level 2 reference model is meant to address this issue: to feed the decomposition process with a start situation, which already encompasses some solutions to the various issues that have to be addressed. Put in another way: the level 2 reference model compels the designer to consider some issues prior to starting the realisation of the functional side of the system. It seems somewhat strange that at some point in the development process the realisation of functionality is of a lower priority than organisational or behavioural issues, but because of the existence of overriding external constraints, this is often the case. The correct sequence of design decisions, of which decomposition is just one, will lead to the optimal result. The question is: what determines the correct sequence? At least a part of the answer is to be found in the four-level model, and in the level 2 reference model in particular.

Appendix I.3 The Level 2 Reference Model: Rationale

The level 2 reference model represents the structure of a certain control regime; many control regimes in one system are possible, which have to be addressed at the level 3 model. The current question is: what can the reference 2 model mean during the design of a system in general, and a control regime in particular? However, before answering that question first the following question has to be posed: what are the rules that determine the foundation of system development and that apply to this reference model? The main subjects here are: (1) simplicity of structure, (2) description rather than prescription, (3) incorporation of a number of organisational and behavioural issues and (4) accommodation of system evolution.

In the past many systems have presented its users with functional or behavioural deficiencies, its owners with maintenance or operational problems, and both parties sometimes with straightforward failures, sometimes even accompanied by human injuries, although the exact amount is the subject of much debate. These phenomena are investigated by a variety of authors and from many different perspectives, but one conclusion stands out: complexity is an evil, almost ubiquitous to all the investigated systems. The lesson is: the basic structure of the system has to be simple. This is reflected in the KISS principle: Keep It Simple Stupid. Complexity renders the resulting system incomprehensible, difficult, error-prone and costly to adapt, and unpredictable to its users. And if things do go wrong, the staff usually encounters great difficulties to localise the error.

The reference model, especially its structure is advisory, is descriptive, rather than that it prescribes how things are to be done. It should be considered seriously when starting functional decomposition, and should only be changed with articulated and good reasons, but it is not the final word. Functionality has also something to say on these matters, but only after due considerations.

The reference model is meant to incorporate a number of organisational and behavioural issues. These come from the level 3 model or straight from the list of non-functional requirements, in particular those that apply to one or more of the individual authorities. To this end first a structure for the prevailing system functionality has to be defined, applying KISS, and subsequently this structure can be enhanced to incorporate especially behavioural issues. The prevailing idea here is that, after refinement, this model has to lead to predictable system behaviour in as many situations as can possibly be achieved.

System evolution is the phenomenon that a system starts with a relatively simple, localised or limited implementation and that the system, over the years, is enriched in functionality, grows in a geographical sense and so forth. This phenomenon is occurring more and more, due to the investments necessary and other shortages. For comparison: think of the motorway network itself. Pure functional decomposition has difficulties to cater for this phenomenon: it is just no part of the scenario. However, the reference model is able to capture the future, evolved system and by supporting this, it creates a stable basis throughout the system's life-cycle.

The rationale behind the reference model is expressed in [SATIN AC20] and can be summarised in the following: (a) similarity and simplicity of the basic structure, (b) input to the functional design process and (c) the foothold for predictable system behaviour.

The reason for simplicity has already been illuminated, but an accompanying advantage has to be mentioned. Similar systems should preferably adopt similarity in fundamental structure, which, conceivably and without further proof, will contribute to standardisation, inter-operability, easy exchange of experience etc. Hence the reference model will admonish the

designers to produce similar results after the functional design, which would be unlikely if the design process would not start from comparable positions and would not apply the same

Following the former arguments: whilst the reference model incorporates behavioural issues, that remain obscure if a purely functional paradigm is followed and hence tend to be neglected or at least to be addressed poorly, this input is valuable to safe-guard many behavioural issues. Many designers are not trained to address those issues, and those that do understand how to do it, have not necessarily the budget and time to do it properly, if eventual incorporation is not obstructed by absence of, or non-compliant measures, elsewhere in the system. This shows the nature of many issues related to system behaviour: by definition they often belong at the system level, rather than at any lower level of decomposition: this influences priorities and sequence of implementation decisions.

Predictable system behaviour is an issue in many systems especially to the users, but it is a prime concern in control systems. The system operators should be able to rely on the system, to be confident that a control action or command should lead to the expected outcome - and nothing more than that: the absence of side-effects, which would render the system more difficult to control. This can only be achieved if this is incorporated in the design sequence and the consequences are thoroughly considered, at a moment where this is still possible and feasible: the level 2 reference model, and not after some layers of functional decomposition. By then the behavioural consequences are often intractable. The reference model makes the design process amenable for analysis of its future behaviour.

Hence the level 2 reference model incorporates a number of issues that usually remain unaddressed or, if addressed in a later stage, are by no means transparent.

Appendix I.4 The Functional Representation: The Control Paradigm

In functional decomposition the functions are defined in a rather haphazard way, and many data flows are defined, leading to a kind of network'. The term haphazard' refers to differences between developers in particular; whilst each developer may use his own rules, and in a reproducible way, other designers might do it differently. And also the reproducibility of the work produced by a single designer leaves something to be desired; this is demonstrated by the difficulties many designers have when reviewing their own work after a certain length of time. Concerning the result: when looking at such a network, especially after some layers of decomposition, the similarity with spaghetti becomes apparent. The negative connotation of this dish in the information processing world is wide-spread and not without reason. These designs are difficult to understand and even more difficult to maintain. And during maintenance this structure will become entangled more and more, resulting in early obsolescence of the system. Neumann formulated a kind of law' on this issue, which states that simplicity of

structure is retained over the years of maintenance, but that an already complex structure tends to increase in complexity [Neumann 1995].

However, the same functionality can be realised in a layered structure and the proof of this is relatively straightforward, as shown by Hitchins [Hitchins 1992], a ‘spaghetti' network can be presented equally well in a layered equivalent. There are two main differences.

First: data flows that go from one function to another that are straight one-to-one in the spaghetti case often need to traverse another function in the layered case. Actually, this reflects the background of the spaghetti' problem.

Second: because the functional structure has now become one-dimensional, a completely new characteristic can be incorporated in the system, which was almost absent, or at least obscure and unarticulated in the spaghetti case. There are some paradigms that can be applied to sequence the functionality, but the most obvious one in a control system is the control paradigm. An interesting point is that this paradigm explicates a hidden, but nonetheless vital and stable aspect of the system. Briefly summarised, the control paradigm expresses the following:

each higher layer controls the lower layer(s);

a lower layer is necessary for the higher ones to function;

a lower layer is, if so desired, able to function on its own (local autonomy), can sometimes partly be unavailable (degradability) and can be in a different location (system distribution).

The first rule is obvious. The second rule needs some more precision. The higher layers need the lower layers, but in some cases one of the lower layers can be empty in case of a simplified version of a more generic system, but this does not affect the principle. This may for instance happen at system inception, anticipating system evolution.

The big advantage is found in the third clause. It embodies local autonomy, degradability and system distribution: two behavioural and one organisational issue respectively. Local autonomy means that the lower level part is able to continue operation if the higher level is absent. Degradability means that some part of the lower structure can be lost, without loosing the prime functionality of the system, which is still governed from the top. Local autonomy and degradability are difficult to realise in any other way, than by the layered structure, because of the many entangled data flows that blur the situation. System distribution is addressed fundamentally: now the system is distributionally pliable: it is organised in such a way that later on, for instance during installation, the physical distribution can be chosen by putting functionality on various locations. If this is done according to traditional functional decomposition, the decomposition must be modelled later on to accommodate distribution, and then only one version is the outcome: a non-pliable, deterministic implementation.

Appendix I.5 The Level 2 Reference Model: Step 1: Functionality

The first step in the derivation of the reference model is to structure the functionality according to the control paradigm. This is relatively straightforward: list all the basic services, or functions of the intended system, and group those according to the control paradigm. Each group has to fulfil a certain sub-goal of the system, which maybe one of the many conflicting (sub)goals to be realised. The combined sub-goals should reflect the overall mission of the system. This requires some thinking, but is not necessarily difficult; consequences of changes can easily be seen, the best solution is quite often intuitively obvious, although later on many minor improvements will be added to the existing structure.

This goal-oriented derivation of the system functionality and transposition into a structure, also serves to arrive at a stable architecture of the system. The goal of the system is assumed to remain stable. Indeed, in some cases the goals do fluctuate and sometimes they even change. However, this cannot be an objection against the proposed approach; two arguments will make this clear. First of all: if the goal of the system is changing in a really noticeable way, it is questionable whether system production is feasible at all; for obvious reasons, stability of the system objectives is the prime assumptions underlying system realisation. Secondly, if the goals of the system change, and hence basically the system as such, then any method enters the danger area, not just the architectural approach using the reference model. Admittedly, most system show some creep in system goals during their life cycle, primarily caused by progressing insight': ideas do not stop developing at the moment the system is initiated or even later on when it is installed, but experience will provoke new insight and new ideas on how to conduct matters. However, this phenomenon has no relationship with system development methods.

The resulting structure will show five to ten layers typically. This result can be presented in a very simple format: a table summarising layer sub-goal, services, functions and main data stores, will normally suffice. Yet this provides a stable foundation for the next step.

During this approach it should be kept in mind that this table should precipitate future extensions of the system.

Appendix I.6 The Level 2 Reference Model, Step 2: Behaviour

The existing structures can be enhanced both in a functional or behavioural way, the last manner being more of fundamental interest to the system engineer. Enhancements can be implemented by inserting additional layers between existing ones, such a layer representing the realisation of a certain system requirement or part thereof. Insertion means that the data flows remain primarily unchanged; incidentally, new data stores can be inserted in the structure.

Functional enhancements are possible, especially to accommodate information exchange with the outside world, the system environment. The ATT environment, being open in many senses, does not only receive input from its sensors, but also from many outside sources, for instance from neighbouring inter-operable systems. By the same token information has to be delivered to the neighbours. The existence of this information exchange is a given and hence stable, but its detailed implementation may change over time, due to further information processing refinements. Also incorporation of the received information is not necessarily trivial and may require intricate computations. So the architecture should be ready to facilitate the exchanges, and not to obstruct future changes, which requirements can be met by the reference model.

Behavioural enhancements are more interesting and encompass a wide range of issues. The most interesting ones are mentioned here, but they all share the condition that incorporation is realised by defining an additional layer in the proper place in the existing structure.

System safety is a close relative of architecture definition and lack of attention to this subject can seldom be compensated later in the life-cycle, because of the many things that have to be considered: reliability to reduce the amount of failures; fail-safe measures to mitigate the consequences of failures; fail-soft measures to give the system users the facility to finish their task properly (limp home' facility); safety warnings, to inform the users timely that something nasty is going to occur in the system e.g. a partial failure, reduced functionality etc.; safety watchdogs to detect potential safety violations in due manner and at a proper time and to start countermeasures; and so forth.

Hence, the incorporation of the full set of safety measures may drastically change the scene of the reference model.

Safety watchdogs, ensuring a safe situation independent of external and internal positions and eventual errors somewhere in the system, can be implemented by a layer that just does that, separating the two layers and preventing undue influences from one layer to another. This can work both sides: unsafe commands can be corrected and unsafe situation can be filtered and can be corrected in a reflex manner: immediate response to correct the situation, without waiting for the higher level structure to react, as in the case of reflexes in the human body, which are often necessary to prevent or to limit damages.

Also the other safety issues can be addressed in a similar style, although it is difficult to describe this in a general way. Usually, each system has to solve these issues in its own way.

Security can be improved in the same way as is done for safety. Intrusion can be detected by authentication of entering information flows and screening of the contents.

Proprioception is another issue. To clarify the term: Current systems, and information systems in particular, are so complex, and certain failures can be so elusive, that only advanced logging of data files and memory, supported by a wide variety of tools, can trap errors.

To illustrate the point: in a number of accidents, some with human fatalities, software errors were suspected, but could not be proven; electromagnetic interference might also be the cause (or one of the threads in the causal net), or even some spurious or intermittent hardware error. Needless to say that this lack of analysability will occur more often with increasing complexity, if no measures are taken. This situation is compounded by the observation pertaining to information systems, that software is often failing in an unanticipated way, and hence requires potent tools for detection and further analysis. For this, among various other books [Wiener 1993]. The software situation is in sharp contrast to hardware failures, which can be anticipated by means of various techniques, although, regrettably, this is not always done. For instance, in the US Space programme, over many years, no catastrophic failures occurred in orbit that were not anticipated beforehand; see the section on diagnostics in [Rechtin 1991].

Proprioception is meant to denote an area that differs from what is normally understood under the term diagnostics. The term diagnostics often conveys the idea that the investigation is only started, and debugging aids are only activated, after presence of the malady has been detected. This may be too late or deadly insufficient. The most annoying errors are intermittent ones, which are difficult to trap even in relatively simple environments. Moreover, the intricacies of their analysis become insurmountable in more complex environments.

Proprioception is more advanced, in the sense that it is a prime objective of the derivation of the architecture, to enable early detection of the presence of a fault, and to supply sufficient information concerning the whereabouts and the nature of the deviation on a (semi-)continuous basis. Diagnostics can be implemented following the traditional approach; proprioception, and the implied link with analysis tools, needs a fundamentally different approach.

It must be clear that proprioception in the layered, and distributed, structure is fundamentally more convenient to implement and to use than in the ‘spaghetti’ case.

By doing so the behaviour of the system for its end-users is not really improved. Yet, to the system maintainers, especially the debuggers, the behaviour is improved and, by gradual system improvements, the end-users will benefit in the long run.

Appendix I.7 Conclusion

The main conclusion is that the level 2 reference model is an attractive, not to say indispensable tool to ensure that the control regime functions as expected, that it shows the desired behaviour, has no behavioural side-effects and that the implementation can be analysed for deficiencies.

Additionally, system evolution can be followed, without changing the basic structure of the system.

APPENDIX J SATIN REFERENCE MODELS

Three ITS reference models have been defined for use in Europe, one each for Urban and Inter-Urban Traffic Management and one for In-Vehicle systems [SATIN AC13-PT7]. The Level 3 and Level 2 issues have been combined into one reference model for each major area, and are summarised in Figure J.1 (see also [SATIN AC20]).

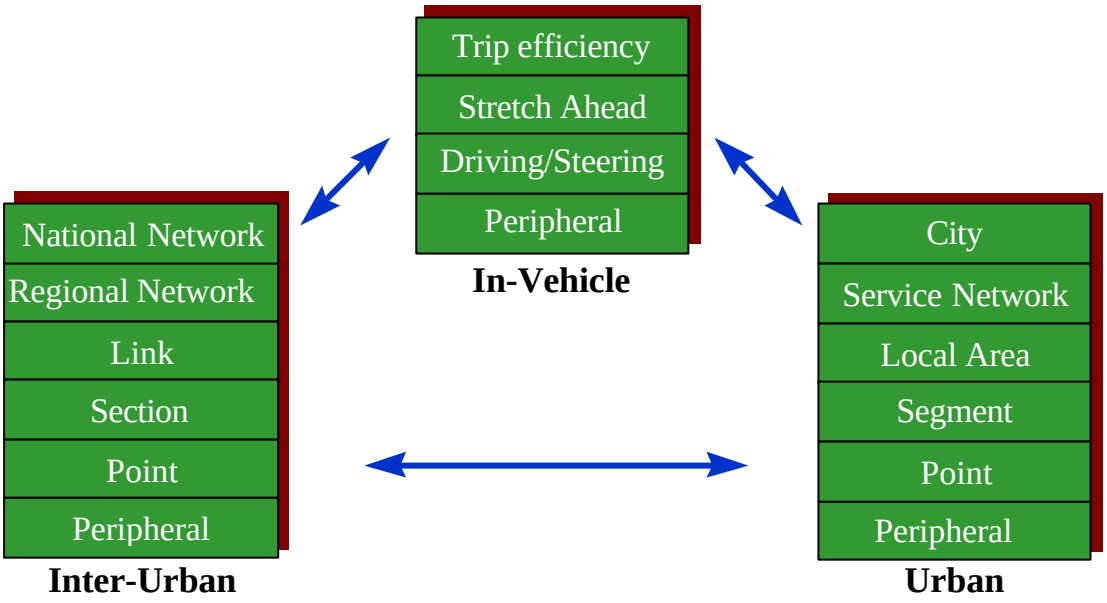


Figure J.1 - SATIN Reference Models

Figure J.2, Figure J.3 and Figure J.4 then expand these three reference models into their set of goals. The semantics of the models are based on the concept of a service; a user is not interested in difficult control algorithms and prefers to view the system as a set of black boxes each providing a number of services. Note that it may be necessary to define the higher layers with additional Level 3 Architectures when more than one authority is involved.

	Scope of Control	Objective	Example of Services
4	Trip Efficiency	Efficiency of trip subject to desired constraints	Route selection Circumvention of congestion Freight management Hazardous goods management
3	Stretch Ahead	Comfort of driver and passengers	Vehicle behaviour smoothing Comfort oriented information
2	Driving/Steering	Safety of vehicle, occupants, cargo and environment	Traffic rule enforcement Collision avoidance Safety oriented information
1	Peripheral	Data collection and command execution	Collection of data within, and in the vicinity of, the vehicle Control of vehicle systems

Figure J.2 - Reference model for In-Vehicle systems

	Scope of Control	Objective	Example of Services
6	City	Co-ordinated policy	Demand management O/D estimation
5	Service Network	Service quality Revenue collection	Public Transport management Parking management Road pricing
4	Local Area	Throughput optimisation	Public Transport regularity Signal plan optimisation Junction priority Individual route guidance
3	Segment	Emission control	Public Transport journey time estimation Dynamic speed control/advice Automatic incident detection
2	Point	Road safety Enforcement	Traffic counts Signal set control Pollution level detection
1	Peripheral	Physical contact with road and traffic	Actuator Device Control Sensor Device Control

Figure J.3 - Reference model for Urban Traffic Management

	Scope of Control	Objective	Example of Services
6	National Network	Network Efficiency	Network Control Hazardous goods Transit Management
5	Regional Network	Traffic flow Incident Management	Traveller Information Services Network Monitoring Services Re-routing Control Services Route Guidance Incident Management Services Hazardous Goods Monitoring
4	Link	Throughput Optimisation	Link Traffic Monitoring Incident Verification Speed Harmonisation Lane Closure Management Co-ordinated Ramp Metering
3	Section	Traffic Safety Management	Driver Awareness Warning Services Section Traffic Monitoring Services
2	Point	Logical contact with road and traffic	Local Ramp Control Services Local Lane Control Services Local Traffic Monitoring Services Local Monitoring of Road Conditions Local Traffic Surveillance Services Enforcement Services
1	Peripheral	Physical contact with road and traffic	Actuator Device Control Sensor Device Control Vehicle Flagging Services

Figure J.4 - Reference model for Inter-Urban Traffic Management

APPENDIX K ENTERPRISE ARCHITECTURE

The concept of an Enterprise Architecture was introduced when preparing the Reference Model of Open Distributed Processing [ISO 10746], which describes an 'enterprise language' to represent a system in terms of interacting agents, working with a set of resources to achieve business objectives subject to the policies of controlling objects.

Objects with a relation to a common controlling objective are grouped together in domains which form federations with each other to accomplish shared objectives (e.g. enabling it to determine security or management policies). Any such union mutually contracted to accomplish a common purpose is termed a community.

Policies set down rules on which actions of which objects are permitted or prohibited, and also which actions objects are obliged to carry out. Policies may relate to, for example:

a) the use of resources accounting for resource usage. For example, *resource usage rules*, defines the constraints that may be imposed by external:

   - regulatory bodies;
   - market demands;
   - environment;

depending on where the resources usage is:

   - public;
   - private;
   - third party

b) The ownership of resources. For example, *transfer rules* may be expressed in an Enterprise Architecture to state the exchange of ownership and/or obligations between resources.

c) The membership of federations. For example, and Enterprise Architecture can include *domain rules* that specify:

   - the membership of a domain;
   - the interaction rules between domains of the same type;
   - the domain naming rules.

d) The assignment of rôles to objects. For example, an Enterprise Architecture may need to define *organisation rules* that state:

   - assignment of rôles and responsibilities to resources within the organisation;

- how members of the organisation are structured (e.g. hierarchy, isocracies);

An Enterprise Architecture may also include business rules to express:

- an enterprise as a business entity;
- accounting requirements;
- strategic planning for the business in order to fulfil its objectives.

e) Permitted interactions between objects holding different rôles (i.e. access control). For example, *security rules* may define:

- the rôle-activity-object relationships, their integrity and confidentiality requirements for activities and objects;
- the rules for detection of security threats;
- the rules for protection against security threats;
- the rules for limiting any damage caused by any security breaches.

f) The responsibility delegated to objects. For example, *delineation of authority rules* are used to assign:

- privileges to agents (trust);
- permission and/or prohibition of performance actions of agents (obligation).

Actions that change policy (in that they alter the obligations, prohibitions or permissions of objects) are termed performative actions. For example, giving a user system's administrator privileges or the creation of an object can be performative actions. Objects that are liable to initiate actions have an agent rôle, whereas those that only respond to such initiatives have artefact rôles.

An example of an Enterprise Architecture can be found in Appendix T.

APPENDIX L FUNCTIONAL ARCHITECTURE

A Functional Architecture provides a structure for the main functions of the system, though attention should be paid to the behavioural issues as well as the goal oriented functions. The Functional Architecture is derived from the Level 2 Architecture, and should be developed by applying decomposition techniques to arrive at a sufficient level of detail.

Rules for decomposition can be found in many text books, e.g. [Hatley 1987 pp 130-8] and [DeMarco 1978]. They can be summarised as follows:

- Each function should be decomposed into a limited number of sub-functions (as rule of thumb 5 ± 2). This makes it easy for the developer to understand what is being stated, and to be able to check the completeness of the set of sub-functions. The process must remain intellectually tractable.
- The sub-functions should be at (approximately) the same level of abstraction, and the same level of detail.
- The sub-functions should be defined in such a way that the number of interactions (exchanges of data between functions) remains at a low level. A high number of interactions may sometimes be a symptom of a highly complex system, but more normally it is a symptom of insufficient effort being taken to keep the functional architecture simple and comprehensible. A high number of interactions will lead to a strongly linked 'spaghetti' system where fault analysis, maintenance and function changes will be difficult, and faults are frequent.
- It is **imperative** that there is hierarchical consistency between the levels of decomposition.

There are no hard and fast rules on how much detail should be given in the Functional Architecture, it will depend on its purpose. Since decomposition can continue down to the detailed design, a decision has to be taken as to when it stops being an architecture and starts to be a design. The more detail that is given in the architecture, the less flexibility will be available for the designers. Often the level of detail will be dictated by the needs of the Communication Architecture, especially for those system architectures that are intended to be open.

Sooner or later most high level non-functional requirements will be decomposed into low level functional requirements, in particular those connected with behaviour. The non-functional requirements should therefore be reviewed by the system architect to find those issues that can be realised by allocating them to a (set of) sub-function(s) of the Functional Architecture. By performing this allocation at this stage the designers will be less likely to forget the issues, and will know where to address them.

Appendix L.1 Proposed Semantic

A Functional Architecture may be made up of the following elements:

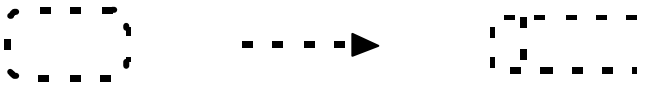
- A **data flow** is a conduit through which packets of information of known composition flow. Data flows can be represented by arrows:

- A **function** is a process that transforms one or more incoming data flows into one or more outgoing data flows. The function is executed on reception of incoming information. Functions can be represented in the functional diagram by, for example, "bubbles":

- A **data store** is a (time-delayed) memory function for information. A data store can be thought of as a database, or any other structure for data accumulation.

- A **terminator** is a net originator, or receiver, of system data. Its purpose is to mark the boundary of the model. Terminators are used to represent people, organisations and systems that interact with the modelled system and can be represented, for example, by rectangles:

- The **dynamic behaviour** of functions is modelled with the aid of control functions, control flows and control data stores. Control functions are those functions that synchronise, or co-ordinate, the data processing functions and/or other control functions. Control flows are flow of the signals that trigger, or control, the sequence of processing. Control stores have the same rôle as data stores. They can be represented by dotted bubbles, arrows and data stores respectively:



It should be noted that some CASE Tools use different symbols, and the case study in Appendix T does not follow the above exactly.

APPENDIX M ODP AND THESE GUIDELINES

ISO is preparing a standard for Open Distributed Processing (ODP): ISO/IEC 10746-1, *Reference Model of Open Distributed Processing (Committee Draft)* [ISO 10746]. Meanwhile, CONVERGE-SA is promoting an architectural approach that is described in these Guidelines, which is also targeted at systems which are distributed and that should be open. How do these two approaches relate to each other and where they differ?

The ODP Standard takes an object-oriented approach to the matter; these Guidelines do not do this explicitly, but leave it to the designers as whether to follow an object-orientated methodology, or the more traditional functional decomposition, or Structured Analysis methodology. Although Object-Orientation can, at times, have certain advantages, the authors of the Guidelines decided to promote Structured Analysis as being good practice and the basis for an architecture and a viable project, primarily because it is well understood by everyone, included those who now work in an Object-Oriented manner.

These Guidelines deal with the subject of architecture development only, and do not cover the more detailed design phases of the development life-cycle. The ODP Standard, however, aims to cover the same subject area but with more attention being given to the implementation. Is there a fundamental difference in thinking about architecture between the two documents?

Architectural Approach

These Guidelines define the term architecture, explains the rationale behind it and describes how to create one. the ODP document also uses the term architecture, but without a clear definition or an indication of its rationale (there is a reference to a list of definitions, including the word ‘architecture’, in another document). However, whatever the definition, the objective of an ODP architecture can only be deducted by inference; in the Guidelines it is stated. On reading the ODP Standard there seems to be no fundamental difference in thinking as to what architecture development embodies, although these Guidelines do pay more attention to system evolution and growth, in an attempt to avoid the static system viewpoint that has been detrimental for many systems, and is more or less ingrained in the usual development methods. For example, both the ODP Standard and these Guidelines acknowledges that many system requirements are not properly addressed in the standard development methods, and hence tend to be forgotten.

The ODP Standard uses five viewpoints, whilst these Guidelines make use of a number of different sub-architectures, including a reference model. The following is a discussion of the similarities and differences in approach under the headings of the ODP viewpoints.

The Enterprise Viewpoint

The ODP Standard recognises the situation in which large-scale, distributed systems, are used by a number of organisations to accomplish a common purpose; in ODP terms this is a 'community'. The Enterprise viewpoint is introduced to address these issues, and there are no fundamental differences between it and these Guidelines' Level 3 Architecture; both have the same function.

The Information Viewpoint

In ODP parlance this represents the information objects and their relationships, where information objects are abstractions in the real world, in the ODP system, or in other viewpoints. Information objects can be expressed in terms of lower level information objects, and, ultimately in atomic information objects, hence data elements.

The information objects can be positioned at the level of these Guidelines' Functional and Information Architectures; everything addressed by the information language can be found there. This should not come as a surprise, because whichever method is used, the functionality in the system needs to be expressed in terms of functions and information, although the ODP Standard keeps rigidly to the use of objects. However, at the level of a system architecture, where abstraction is at the highest level, the differences are minimal, as are the advantages of explicitly working with objects since, by definition, every item mentioned at this level can be called an object. (Differences do appear at the lower levels of design, where 'object' has a more rigorous definition because of its need to be supported by compilers, subroutine packages and databases.)

The Computational Viewpoint

In the ODP Standard this is where the interactions between the objects are captured, but without mentioning the details of the interaction. This is comparable to using levels of decomposition in Structured Analysis terms. The ODP Standard naturally follows a strict OO approach.

The Engineering Viewpoint

This captures the distribution of the system, one of the prime objectives of the ODP Standard. This is analogous to the Physical Architecture in these Guidelines, which addresses distribution of functions and information, their replication, and their interactions.

The Guidelines' Communication Architecture might be included in the Engineering Viewpoint, but this is unclear; maybe communication is taken for granted. This may be suitable for 'normal' communication, this is definitely unwise for something like EDIFACT, which needs special expertise.

The Technology Viewpoint

These are statements about the actual components, from which the distributed system is composed. These Guidelines state that an architecture would normally be technology independent.

Reference Models

The main difference between the two documents is the absence of Reference Models in the ODP Standard. ODP does not mention such an entity, and these Guidelines explicitly address them, indeed requires them. This difference may be caused by a difference in thinking about architecture, or at least a difference in emphasis. These Guidelines acknowledge the need for control, and also the evolutionary features of ITS, which necessitates some deep thinking about the fundamental structure of the system and its functionality. A Reference Model is a good way of describing these issues.

Moreover, although the ODP Standard mentions the issue of 'object behaviour' throughout, this does not fully work through to behaviour at the system level. Thus system security is mentioned, but system safety is not. Degradability, recoverability, local autonomy and related issues are not at all mentioned; maybe in ODP these issues are not expected to play a major role. The Reference Model of these Guidelines is able to ensure that the roots of these issues are planted for further germination during the development process. This cannot be achieved using the ODP method, and in the field of transport it is essential that such issues of system behaviour are considered to ensure proper operation under adverse conditions.

A Reference Model describes the manner in which the first level of decomposition of the system functionality is performed. This first decomposition, and the consideration of behavioural issues is of prime importance. Much of the success of the future system, be its functionality, behaviour, evolution, flexibility or maintainability depends on just this step.

Object Modelling Techniques

Object modelling techniques do not address the higher system levels, such as the Enterprise Viewpoint, and hence do also not address system behaviour and evolution. Object modelling is functionally equivalent, in terms of a *working* system, with Structured Analysis. It starts at these Guidelines' Level 1 Architecture and progresses down, in a structured way.

State transitions, as incorporated in object modelling, are not mentioned in these Guidelines. This is not necessarily a defect, because the concern of these Guidelines is the development of an architecture, in particular the higher levels. State transitions, representing control behaviour at the micro-level, are not fundamental to the architecture of a large scale system, although they may become relevant for the architecture of a smaller system.

APPENDIX N INFORMATION ARCHITECTURE

Data modelling embodies two different viewpoints: the user view and logical database modelling. These two are the highest two of the three level ANSI-SPARC model for information modelling; the third level represents the physical database structure. The objective of the Information Architecture is thus to describe the set of user views and to provide the top level structure of the information base.

A user view describes the needs of a particular user or group of users, and corresponds to part of the real world as perceived by them. It describes the purpose of a certain information entity, the meaning of the attributes, constraints, dynamic behaviour and anything else (e.g. privacy, security, timeliness, update rules, authorisation) that might be of importance, but not necessarily related to the structure of the Information Architecture. When the system is operational the user view corresponds to how the user can manipulate the information entity, irrespective of the logical and physical structure. The user views provide the foundation for the realisation of the User Needs.

The Information Architecture captures the user views in the top-level logical structures and the data dictionary. It is recommended that a formal representation such as Entity Relational Diagrams are used to show the top-level logical models.

For those systems which will be centred on a large database of information the creation and maintenance of the data dictionary is of vital importance. A **data dictionary** is a database used for data that refers to the use and structure of other data; that is, a database for the storage of the **meta-data** (data that defines and describes data elements) of an information processing system. Data dictionaries have been primarily used in database design and implementation. However for maximum benefit a data dictionary system can be used throughout the whole of a system development life-cycle, including functional analysis and operational management.

The process of controlling and co-ordinating data definitions is named data administration [Newton 1993, Holloway 1988]. In a computer-based system the data dictionary meta-data will be stored in a data dictionary database; the software that manages this database is named the “data dictionary system” , which is specifically designed to collect such definitions, to generate reports and to enable the performance of automatic consistency checks. Any inconsistencies brought about by incompatible user views will have to be resolved by the Data Administrator. All this makes a common data dictionary an important asset for corporate awareness of the existence and value of data. A data dictionary system can be a tool to spread knowledge on common data definitions.

A **data element** is a unit of data for which the definition, identification, representation, and permissible values are specified by means of a set of attributes [ISO 11179-x], e.g.:

- data element name, and short name (if allowed);
- data element alias(es) (if necessary, to comply with a predefined naming convention);
- data element definition;
- data element description (including examples of use, forbidden use, warning on ambiguities etc.);
- format;
- domain (discrete or continuous);
- meaning of domain values (if applicable);
- constraints;
- mandatory/optional conditions;
- privacy considerations, authorisation, classification etc.
- uniqueness (necessary for keys in a database), rules for the generation of unique identifiers;
- administrative information, e.g. when added, where used, owner of the definition, history of changes.

Appendix N.1 Information Repository

Many different projects develop large information processing systems. A lack of awareness of common data between these projects can result in different systems for the same tasks. A lack of understanding of the meaning of common data can also result in the incorrect use of data. This is the reason why, in a distributed information environment, a common information repository serves as a single tool for handling all definitions.

When combined, the Information and Functional Architectures provide a fundamental description of the real world environment for which the system must be developed. The **information repository** is a database application used to store meta-data about shareable data, and provides a means of creating and maintaining a consistent overview of all the data in a system, thus it could contain:

- an overview of the system functions, where each function is characterised by a name, purpose and reference to its inputs and outputs (the messages); the current list of function for road transport ITS can be found in [SATIN D004-PT3].

- the overview of the messages in the system, the composition of messages as a collection of basic data items and the cross reference between messages and functions; the current list of messages for road transport ITS can be found in [CORDEX AC23]
- an overview of the relevant entities in the real world, their interrelationships and the groups of basic data-items that describe them; these will describe the structure of the various databases of the ITS.
- the overview of the basic data elements, where each data element is characterised by a name, purpose and a description of domain.

Information repositories are themselves databases and therefore their contents can also be described in a data model. Figure N.1 illustrates the overall structure of the SATIN information system dictionaries in an ERD diagram.

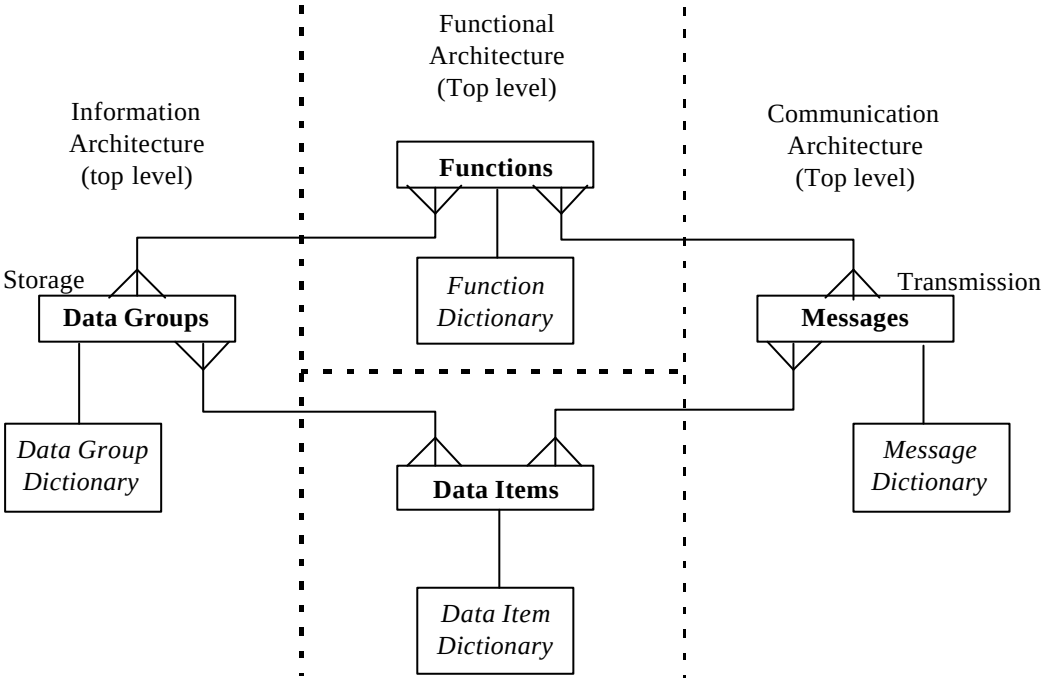


Figure N.1 - Overall structure of an information repository

Further information can be found in [SATIN-AC18].

APPENDIX O ASSESSMENT BY A REVIEWER

It is important to realise is that a project system architecture Deliverable may not contain the complete system architecture. If these Guidelines have been followed in detail for a large system then the total documentation produced will be very large indeed, and far more than is acceptable for an official Deliverable. In these circumstances the Deliverable will be a top-level overview of the complete work. Other projects, however, may not wish to put as much effort into their system architecture work, and the Deliverable will represent their complete work: this may be perfectly valid for certain projects, especially if the system is small (e.g. in-vehicle).

Whilst the assessment process described in Section 7 will take a project some time to perform in detail, a reviewer will be asking the same types of questions of the Deliverable but have much less time to perform the task. However the objectives of the two parties are different. The project will wish to know whether that particular system architecture is suitable for their particular needs, or maybe it is trying to make a choice from a variety of possible options; a reviewer, on the other hand, is more concerned with being able to advise the Commission that the project is proceeding in a sensible manner.

General Issues

There is a good engineering 'rule of thumb' that says if something looks right then it probably is right; the corollary is also true, namely that if something looks wrong then it probably is wrong. When reading a system architecture Deliverable one can very quickly get a feeling for whether the authors have understood what they were doing, and have proceeded in a systematic and logical manner.

Specific Issues

By looking for answers to the following questions it is possible to build up an opinion as to the overall effectiveness of the system architecture Deliverable.

- Has the project taken a systematic approach to the creation of its system architecture?
- What is the system concept?
- What are the main functional requirements?
- Is there evidence that a full list functional requirements exits?
- What are the main non-functional requirements?
- Is there evidence that a full list non-functional requirements exits?
- Is there a context diagram (e.g. PASSPORT Diagram)?
- Is the system boundary clear?

- Has a preliminary safety analysis been performed? If not, why not?
- What types of architectures have been developed (e.g. Level 3 & 2 Reference Models, Level 1 Functional, Information, Physical, Communication and Enterprise Architectures - they may not all be necessary)?
- Do the Level 3 and 2 Reference models satisfy the functional and non-functional requirements?
- Is there consistency between the Level 3, Level 2 and Level 1 Architectures?
- Does the Functional Architecture make sense (e.g. what goes in must come out, what comes out must be created from what goes in)?
- Does the Functional Architecture satisfy the functional requirements?
- (Road mode only) Does the Functional Architecture make use of the CORD Function List? If not, why not?
- Is there hierarchical consistency in the decomposition of the Functional Architecture?
- If the system needs a 'database' is there an Information Architecture? (Note: not all systems will need a full Information Architecture)
- Is there a Physical Architecture?
- Does it look as though the Physical Architecture is consistent with the Functional Architecture?
- Is there a Communication Architecture?
- Does the project need specific standards? Do these standards exist?
- Is the architecture open? If not, why not?
- Is the project likely to produce the system that they are intending to produce with the system architecture presented in the Deliverable?

APPENDIX P THE PASSPORT CROSS

Appendix P.1 Introduction

The PASSPORT Cross model is used to relate four groups of system elements using matrices. When being used to check the consistency between a Functional Architecture and a Physical Architecture they are as follows:

Functional Elements (FE) - The lowest specified level of (sub-)functions (including terminators) contained within the Functional Architecture.

Information Sets (IS) - The information that passes between the Functional Elements.

Physical Elements (PE) - The lowest specified level for the units contained within the Physical Architecture.

Communication Facilities (FE) - These cover all the means of transmission of data between the Physical Elements.

The four necessary matrices may be positioned in the following "cross" format to aid various consistency and safety analyses, as shown in Figure P.1.

The contents of each of the matrices of Figure P.1 are discussed later.

Clearly there are four explicit matrices to construct, namely:

- Functional Elements/Information Sets (FE-IS),
- Communication Facilities/Information Sets (CF-IS),
- Communication Facilities/Physical Elements (CF-PE),
- Functional Elements/Physical Elements (FE-PE),

The matrices FE-IS and CF-PE are sometimes referred to as connection matrices, whilst the matrices CF-IS and FE-PE are sometimes referred to as projection matrices. It is essential that the axes of these four matrices reflect the ordering presented in Figure P.1 (e.g. the matrix FE-IS has "Functional Elements" as its horizontal axis and "Information Sets" as its vertical axis). It is also useful to arrange the axes on each matrix so they align when placed in their "cross" formation (e.g. the matrix FE-IS has "Functional Elements" at the bottom and "Information Sets" at the right). This conformance will help when these matrices are compared for consistency.

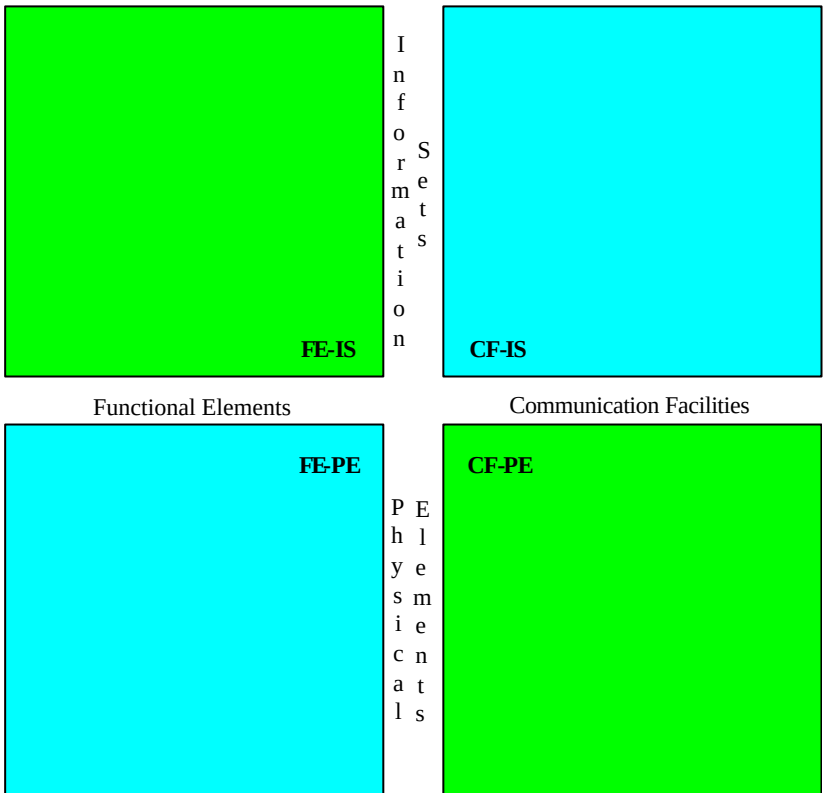


Figure P.1 - PASSPORT Cross Model

Each of the matrices should be constructed independently even though any one may be generated from the other three (see also Note below). This independence will be essential for the consistency checks. Entries in the connection matrices should record the *direction of flow* using an Input and Output notation. For the FE-IS matrix this indicates the source and sink of data, for the CF-PE matrix this refers to the property of the communication facility (transmitter (O), receiver (I) or dual communication (IO)). Entries in the projection matrices should take the form of a tick (✓).

Thus a typical matrix would appear as shown in Figure P.2.

	C1	C2.1	C2.2	C2.3	C3.1	C3.2
P1	O					
P0.1.1	I	O				
P0.1.2		I	O			
P0.1.3			I	O		
P0.2.1				I	O	
P0.2.2					I	O
P0.3						I

Figure P.2 - Example CF-PE

In Figure P.2 we read the Input/Output nomenclature as, for example "physical element P0.1.3 takes its input from communication facility C2.2 and places its output on communication facility C2.3".

NOTE - This Appendix only describes the very basic set of facilities offered by the PASSPORT Cross, and for this reason a slightly different approach to that described in [PASSPORT D9] needs to be taken when creating the Cross itself.

For example, in the situation when many (sub-)functions are performed by one Physical Element there will be no visible Communication Facilities down which to transmit the Information Sets that pass between these Functional Elements. It is therefore necessary to create 'dummy' Communication Facilities down which all these 'internal transfers' pass. There can either be one *dummy* Communication Facility for all internal transfers, or one for each Physical Element containing more than one Functional Element.

Appendix P.2 Connection Matrices

The two connection matrices describe the connections of the Target of Evaluation. The first connection matrix defines the functional model in terms of *Functional Elements* (FE) and *Information Sets* (IS) upon which they operate. The matrix, FE-IS, is completed by indicating which information sets are being used by each functional element. The second connection matrix defines the physical architectural model in terms of *Physical Elements* (PE) and *Communication Facilities* (CF) that join them. The matrix, CF-PE, is completed by indicating which communication facilities permit the physical elements to communicate with each other.

Thus the FE-IS matrix describes the exchange of information between functional elements at the logical level and the CF-PE matrix describes the various physical characteristics associated with the connections.

Appendix P.3 Projection Matrices

The two projection matrices relate the elements of the functional model to the elements of the physical architectural model.

The matrix FE-PE identifies which physical elements are being used to implement each functional element. The matrix CF-IS identifies which communication facilities are being used to transmit each information set.

It is the ability of the PASSPORT Cross model to relate the two different models of the system that makes this modelling technique so valuable in architecture analysis.

For architecture analysis it is important to know the allocation of functional elements and information sets to physical elements and communication facilities because a functional architecture can be implemented in various ways. Examples are:

- *Functional replication* - when this redundancy technique is applied one functional element is implemented more than once to increase reliability and availability. For software, diversity and other precautions are necessary. (See hardware duplication below.)
- *Data replication* - with this redundancy technique some of the data can be replicated, to improve response time and to increase the reliability and availability of the data involved. The disadvantage is the potential inconsistency between the replicated data. (See hardware duplication below.)
- *Physical replication* - when this redundancy technique is applied some of the physical elements can be replicated, to improve reliability and availability of the system. (See hardware duplication below.)
- *Communication replication* - when this redundancy technique is applied some of the communication facilities can be replicated, to improve reliability and availability for the system. (See hardware duplication below.)
- *Hardware duplication or alternative pathways* - these redundancy techniques are used to maintain operation of the system during hardware break down.
- *Hardware sharing* - when this re-use technique is applied one hardware component performs multiple functional elements.
- *Distributed functional elements* - at one extreme all the functional elements can be concentrated on one computer, at another extreme the functional elements can be distributed on various computers with considerable distance between the systems.
- *Distributed information sets* - one, a number, or all of the information sets can be centralised or decentralised, dependent on the requirements of the system.

Appendix P.4 Consistency Checks

There are two classes of check:

- *Intra-matrix* - these checks are performed upon each matrix independently to confirm that it is well-formed.
- *Inter-matrix* - these checks are performed across the set of matrices as a whole to confirm that they fully relate to each other.

Appendix P.4.1 Intra-matrix Consistency Checks

Connection Matrices

Consistency checks can be applied to the connection matrices according to the following rules:

The FE-IS matrix

The following consistency checks can be applied to the FE-IS matrix:

MCC1: Each internal functional element must have at least one Input information set and at least one Output information set.

MCC2: Each external functional element (system terminator) must have either an Input information set or an Output information set.

MCC3: Each information set must provide an Input to a functional element and an Output from a functional element.

The CF-PE matrix

The following consistency checks can be applied to the CF-PE matrix:

MCC4: Each internal physical element must have at least one Input communication facility and at least one Output communication facility.

MCC5: Each external physical element (architectural terminator) must have either an Input communication facility or an Output communication facility.

MCC6: Each communication facility must provide an Input to an physical element and an Output from an physical element.

Projection Matrices

Consistency checks can be applied to the projection matrices according to the following rules:

The FE-PE matrix

The following consistency checks can be applied to the FE-PE matrix:

MCC7: Each functional element should be implemented by at least one physical element (allowing more than one in the case of architectural redundancy).

MCC8: Each physical element should support at least one functional element (allowing more than one in the case of architectural re-use).

The CF-IS matrix

The following consistency checks can be applied to the CF-IS matrix:

MCC9: Each information set is carried by at least one communication facility (allowing more than one in the case of architectural redundancy).

MCC10: Each communication facility sustains at least one information set (allowing more than one in the case of architectural re-use).

Appendix P.4.2 Inter-matrix Consistency Checks

The basic property of the inter-matrix consistency checks (for a simple system) is that for each entry in FE-IS there should be an entry in CF-IS on the horizontal, this should have an entry in CF-PE on the vertical, which itself should have an entry in FE-PE on the horizontal. This final entry should be vertically below the original entry in FE-IS (see Figure P.3). Here we present several different forms of inter-matrix consistency checks each requiring a different amount of resources and each providing a different level of confidence in the soundness of the Target of Evaluation modelled by the matrices of the PASSPORT Cross.

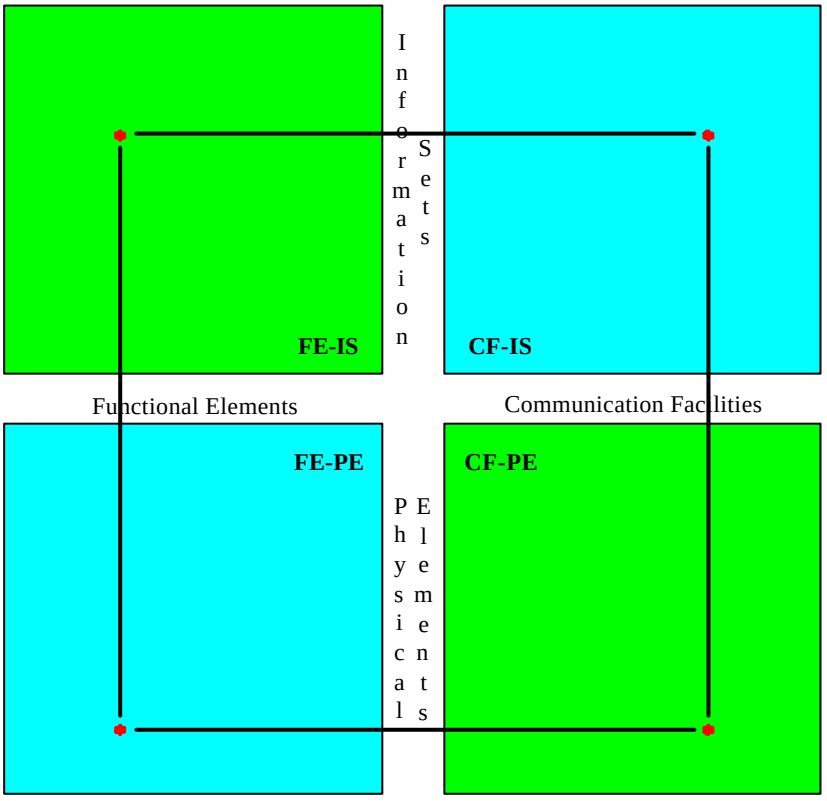


Figure P.3 Inter-matrix Consistency Checks within the PASSPORT Cross Model

Clockwise Consistency Checks

The theory here is that, from the functional element view point, in a well-formed design, the physical elements which support a given functional element, will be connected to those communication facilities which sustain the information sets flowing into and out of the functional element. The following 5 steps should be undertaken:

Step 1: Select a functional element.

Step 2: Select all information sets which either act as Input to this functional element or act as Output from this functional element or both.

Step 3: Select all communication facilities which sustain the information sets selected in step 2.

Step 4: Select all physical elements which are connected by the communication facilities selected in step 3.

Step 5: Select all functional elements embodied by the physical elements selected in step 4.

Notice that each communication facility, sustaining an information set which either flows into or out of the given functional element, is associated with at least two physical elements. However, in general, only one of these physical elements will embody the chosen functional element. This means that the set of functional elements resulting from the application of the above 5 steps should contain the original functional element but will invariably contain other functional elements. Thus the following check constitutes the clockwise consistency check:

11.1 The list of functional elements identified in step 5 above must contain the function chosen in step 1 above.

It is also possible to begin the clockwise tracing by selecting any of the system elements and tracing around the PASSPORT Cross model in a clockwise direction. Thus the generic 5 step plan would be:

Step 1: Select a system element.

Step 2: Using the matrix FE-IS for a chosen functional element, CF-IS for a chosen information set, CF-PE for a chosen communication facility or FE-PE for a chosen physical element select all system elements related to the system element selected in step 1.

Step 3: Following around the sequence of matrices FE-IS, CF-IS, CF-PE, FE-PE, in that order, select all system elements related to the system elements selected in step 2.

Step 4: Following around the sequence of matrices FE-IS, CF-IS, CF-PE, FE-PE, in that order, select all system elements related to the system elements selected in step 3.

Step 5: Following around the sequence of matrices FE-IS, CF-IS, CF-PE, FE-PE, in that order, select all system elements related to the system elements selected in step 3.

MCC11.1.1 The list of system elements identified in step 5 above must contain the system element chosen in step 1 above.

Anti-clockwise Consistency Checks

The theory here is that, from the functional element view point, in a well-formed design, the information sets which provide input and output for a given functional element, will be sustained by those communication facilities which are connected to the physical elements which support the functional element. The following 5 steps should be undertaken:

Step 1: Select a functional element.

Step 2: Select all the physical elements which embody this functional element.

Step 3: Select all the communication facilities which are connected to the physical elements selected in step 2.

Step 4: Select all the information sets which are sustained by the communication facilities selected in step 3.

Step 5: Select all the functional elements which use, as either Input or Output, the information sets selected in step 4.

Notice that each information set, sustained by a communication facility which is connected to any physical elements embodying the functional element, is associated with at least two functional elements. This means that the set of functional elements resulting from the application of the above 5 steps should contain the original functional element but will invariably contain other functional elements. Thus the following check constitutes the anti-clockwise consistency check:

MCC11.2 The list of functional elements identified in step 5 above must contain the function chosen in step 1 above.

It is also possible to begin the anti-clockwise tracing by selecting any of the system elements and tracing around the PASSPORT Cross model in an anti-clockwise direction. Thus the generic 5 step plan would be:

Step 1: Select a system element.

Step 2: Using the matrix FE-PE for a chosen functional element, CF-PE for a chosen physical element, CF-IS for a chosen communication facility or FE-IS for a chosen information set select all system elements related to the system element selected in step 1.

Step 3: Following around the sequence of matrices FE-PE, CF-PE, CF-IS, FE-IS, in that order, select all system elements related to the system elements selected in step 2.

Step 4: Following around the sequence of matrices FE-PE, CF-PE, CF-IS, FE-IS, in that order, select all system elements related to the system elements selected in step 3.

Step 5: Following around the sequence of matrices FE-PE, CF-PE, CF-IS, FE-IS, in that order, select all system elements related to the system elements selected in step 3.

MCC11.2.1 The list of system elements identified in step 5 above must contain the system element chosen in step 1 above.

Clockwise and Anti-clockwise Consistency Checks

The theory here is that since both the clockwise consistency checks and the anti-clockwise consistency checks produce a set of system elements which should contain the original system element but that each check invariably produces a different set, then it makes sense to strengthen the accuracy of the individual checks by combining their output sets.

MCC11.3 The intersection of the list of system elements identified in step 5 of the clockwise consistency checks and the list of system elements identified in step 5 of the anti-clockwise consistency checks must contain the original system element chosen for both their step 1s.

Clockwise and Anti-clockwise Input and Output Consistency Check

The theory here is that, from the functional element view point, in a clockwise consistency check, in a well-formed design, the physical elements which support a given functional element, will be connected to those Input, or Output, communication facilities which sustain the information sets acting as Input to, or Output from, respectively, the functional element. Thus in step 4 of the clockwise consistency checks the extra, undesired, physical elements which were selected because each communication facility is associated with two physical elements, one for which it carries an Input and one for which it carries an Output will be ignored. Once again we will invariably produce lists of functional elements which contain the original functional element but the list produced by selecting just the Input information sets will be different to the list produced by selecting just the Output information sets. Thus by combining these lists we may strengthen the accuracy of the individual checks. A similar argument can be applied to the anti-clockwise consistency checks. Again it makes sense to combine the lists produced by the respective anti-clockwise Input and Output checks firstly with each other and then with the

list produced by the clockwise Input and Output checks. The following 19 steps should be undertaken:

Step 1: Select a functional element.

Step 2: Select all information sets which act as Input to the functional element selected in step 1.

Step 3: Select all communication facilities which sustain the information sets selected in step 2.

Step 4: Select all physical elements for which the communication facilities, selected in step 3, act as Input.

Step 5: Select all functional elements embodied by the physical elements selected in step 4.

Step 6: Select all information sets which either act as Output to the functional element selected in step 1.

Step 7: Select all communication facilities which sustain the information sets selected in step 6.

Step 8: Select all physical elements for which the communication facilities, selected in step 7, act as Output.

Step 9: Select all functional elements embodied by the physical elements selected in step 8.

Step 10: Select all the physical elements which embody the functional element selected in step 1.

Step 11: Select all the communication facilities which act as Input to the physical elements selected in step 10.

Step 12: Select all the information sets which are sustained by the communication facilities selected in step 11.

Step 13: Select all the functional elements for which the information sets, selected in step 12, act as Input.

Step 14: Select all the communication facilities which act as Output to the physical elements selected in step 10.

Step 15: Select all the information sets which are sustained by the communication facilities selected in step 14.

Step 16: Select all the functional elements for which the information sets, selected in step 15, act as Output.

Step 17: Take the intersection of the set of functional elements selected in step 5 and step 9.

Step 18: Take the intersection of the set of functional elements selected in step 13 and step 16.

Step 19: Take the intersection of the set of functional elements selected in step 17 and step 18.

The consistency check here can now be stated as:

MCC11.4 The list of functional elements identified in step 19 must contain the original functional element chosen in step 1.

Unfortunately, especially when system terminators are included in the selection process, there will be instances of empty sets of elements created during the above scheme. This will mean that in some situations the sets of functional elements identified in steps 17 - 19 will be empty. In these cases it will be useful to trace back through the relevant steps to identify the last non-empty list of functional elements and use this list as a substitute for the empty list identified.

Matrix Construction Consistency Checks

In the previous forms of inter-matrix consistency checks there was lots of replication in the tracing process. For example, since an information set is passed between two functional elements then the tracing of each of these functional elements will include this information set and in turn the communication facility that carries this information set and so on. Here we suggest an alternative which will only compute each interrelationship once and thereafter appeal to this computation for the required validity.

The theory here is that from the four basic PASSPORT Cross matrices, namely FE-IS, CF-IS, CF-PE and FE-PE it is possible to construct two new matrices FE-CF and PE-IS. The information embodied in these two matrices can then be scrutinised for well-formedness of the original PASSPORT Cross model.

There are however, four ways to construct each of the two new matrices.

The matrix FE-CF can be constructed by:

1) Having the functional elements as the focus and using the matrices FE-IS and CF-IS.

2) Having the communication facilities as the focus and using the matrices CF-IS and FE-IS.

3) Having the functional elements as the focus and using the matrices FE-PE and CF-PE.

4) Having the communication facilities as the focus and using the matrices CF-PE and FE-PE.

The matrix PE-IS can be constructed by:

1) Having the information sets as the focus and using the matrices FE-IS and FE-PE.

2) Having the physical elements as the focus and using the matrices FE-PE and FE-IS.

3) Having the information sets as the focus and using the matrices CF-IS and CF-PE.

4) Having the physical elements as the focus and using the matrices CF-PE and CF-IS.

It might be useful to construct each matrix in the four different ways and then to compare the results.

Other Inter-matrix Consistency Checks

There may be other more complex inter-matrix consistency checks but these will certainly raise issues of advantage against the resources required. Certainly an automated checking tool would further this discussion and a CAE tool is the subject of the ESPRIT project COMPASS.

Conclusion

We have presented several different forms of inter-matrix consistency checks each providing a different level of confidence that the matrices of the PASSPORT Cross fully relate to each other. These checks are all syntactic and it would require some form of semantic check to provide 100% confidence in the validity of the PASSPORT Cross matrices. Dependent upon the degree of confidence required a choice must be taken as to which of the inter-matrix consistency checks should be applied.

APPENDIX Q BEHAVIOUR ANALYSIS TOPICS

The following topics should be considered when performing a behaviour analysis on the system architecture; not all topics will be relevant top all projects.

- **Non-functional requirements** - the objective of this analysis is to assess the degree to which the non-functional requirements in Appendix G have been achieved.
- **Communication System Capacity** - the objective of this analysis is to assess the Communication Architecture for its ability to serve both current and future needs. Issues will include:
  - expected size, complexity and number of messages
  - communications bandwidth.
- **Degraded Mode of Operation** - the objective of this analysis is to assess the support that the system architecture provides for:
  - system safety impacts of in-vehicle failures
  - system safety impacts of infrastructure failures
  - maintenance activities

Issues to be considered will include the provision of:

  - graceful degradation of functionality
  - fail soft
  - fail safe
  - fail operational
- **Feasibility and Risk** - the objective of this analysis is to assess the feasibility of deployment, and to identify the risk associated with it. This analysis must include:
  - risk identification
  - risk rating
  - risk mitigation

Issues to be considered will include (see also [QUARTET D53]):

  - non-provision due to technical, cost or institutional reasons
  - non-implementation of part of system architecture
  - services not purchased by the end user
  - technology placing limits on market penetration

- architecture not commercially viable
- limited take-up (e.g. of in-vehicle equipment)

- **Performance of Equipped Vehicles** - the objective of this analysis is to assess the benefit that the users will receive from in-vehicle equipment. Issues can include:
  - User travel time reduction
  - Safety
  - Convenience
  - Non-user travel time reduction
  - Non-user safety improvement

Particular thought should be given to vulnerable road users and disabled road users.

- **Accuracy of Traffic Prediction Models** - the objective of this analysis is to assess the support that the system architecture gives to the prediction of near future localised traffic conditions. Issues might include:
  - number of loop detectors
  - fraction of probe vehicles
  - fraction of routes known by the system.
- **Efficiency of Traffic Monitoring and Control** - the objective of this analysis is to assess the expected values of :
  - incident/accident detection time
  - demand peak detection time
  - infrastructure failure detection time
  - detection of lane closure time
- **Efficiency of Traffic Management Centre** - the objective of this analysis is to assess the time interval required for the traffic management centre to receive reports, compute reactions and disseminate information. Issues will include:
  - information sources
  - communication links (e.g. latency)
  - data accuracy and timeliness
  - strategy algorithms.

- **Accuracy of Position Location** - the objective of this analysis is to assess the support that the system architecture gives to accurately placing a vehicle's location in the network.
- **Effectiveness of Information Delivery Methods** - the objective of this analysis is to assess the support that the system architecture gives to :

  a) the delivery of time critical information (e.g. to vehicles, VMS etc.)

  b) easy to use user interfaces

APPENDIX R THE CONVERGE ANALYSIS TOOL

Appendix R.1 What can it do for you?

The CONVERGE System Architecture Analysis Tool has been designed to give:

- System designers
- System integrators
- System owners

the ability to estimate the potential performance of Integrated Transport Environment (ITE) system services while the architecture is being developed. In particular:

- The **impact** of a certain choice of architecture, or the certain choice of function, can be assessed
- The expected **performance differences** between various possible architectures can be investigated
- **Sensitive elements** of an architecture can be identified, i.e. when small changes in one section cause large variations in the expected responses.

The CONVERGE System Architecture Analysis Tool will give you confidence that the architecture you have designed will produce the effect that you desire.

The CONVERGE System Architecture Analysis Tool is also able to maintain a consistent view of your architecture at a number of levels of detail, with the ability to produce printed documentation if required

Appendix R.2 How does it work?

Appendix R.2.1 Summary

The approach taken by CONVERGE-SA is similar the that which has already been used successfully in many other engineering disciplines, where an "architecture level analysis" is used to evaluate the performance of a new design very early in the life-cycle. The CONVERGE System Architecture Analysis Tool can be considered to be made up of four different phases.

1) A Functional Architecture **model** is based on the functions in the CORD Function List. When complete it may consist of four levels of abstraction, which have been checked for their self-consistency.
   - Top Level
   - Area Level
   - Function Level
   - Sub-Function Level

2) By referring to existing systems, or the experience of experts, hypotheses can be made about the performances of each sub-function; alternatively the performance can be taken from the **results** obtained from field trial on real systems. The sub-functions, together with their described performance are being brought together into a central repository and being maintained by the CONVERGE-SA project. This repository is available for use by all Framework IV Transport Telematic projects.

3) The performance indicators described for each sub-function are used by the **simulator** to predict the performance of the current model.

4) Each application of the model can be **documented** on an application by application basis.

It should be noted that since the simulator depends upon both the current CORD Function List, and the results from previous field trials, it is not yet possible for the tool to be fully utilised by the current non-road transport mode projects.

Appendix R.2.2 The meta-model approach

If we were to use the dynamic flow of data as the basis for the simulation, it would be necessary to define precisely the input-output relationships for each function. The result would be far too complex to simulate successfully, even assuming that the relationship could be found. CONVERGE has instead taken a "meta-model approach" in which the simulation is done in the parameter domain rather than in the time domain. As a result the output of the model will be expressed directly in terms of the characteristics of the exchanged data, rather than being derived from the use of statistics in a classic time-related simulation. Figure R.1 shows the difference between a model and a meta-model. The top figure simulates the sub-function using the values of the inputs and outputs, whilst the meta-model transforms the characteristics of the inputs and outputs.

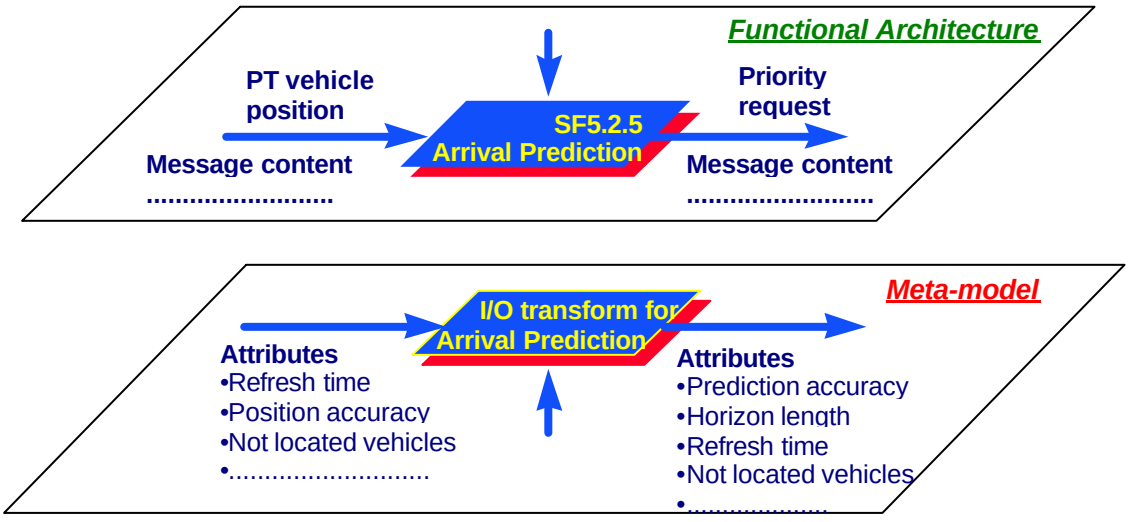


Figure R.1 - Model and Meta-Model

Appendix R.2.3 Building up the model

Although the model can be built up top-down, bottom up or middle-out, we will here describe the top down approach.

Top level Model

The top level model is very basic and is likely to be the same for most ITE systems. Figure R.2 shows a User interacting with a system within an environment.

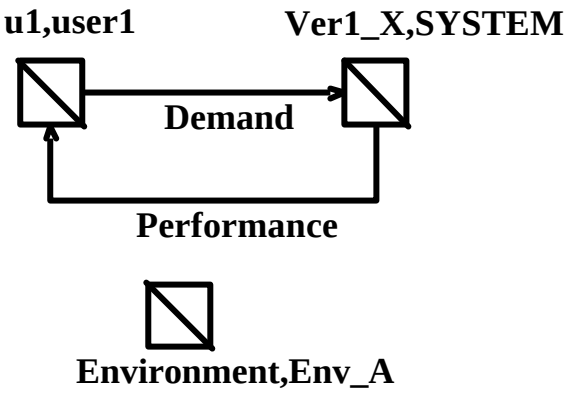


Figure R.2 - Top Level Model

Area Level Model

The ITE system is then broken down into its various Areas, as defined in the CORD Function List: in Figure R.3 we see that the System will consist of Traffic Management (Area 3) and Public Transport Management (Area 5). The various data that will be passed between these two areas are then added to the model.

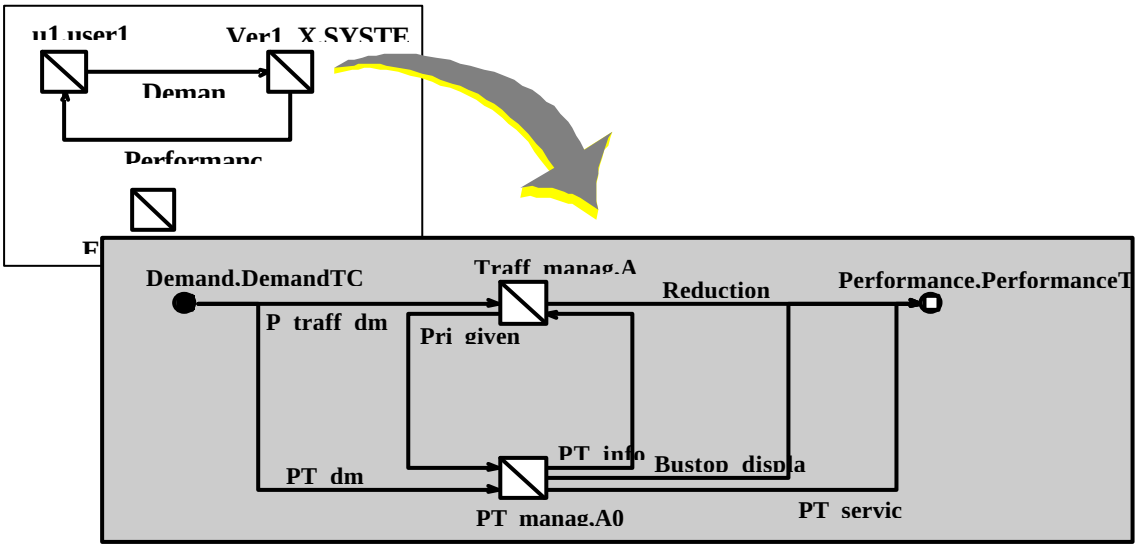


Figure R.3 - Area Level Modelling

Function Level Model

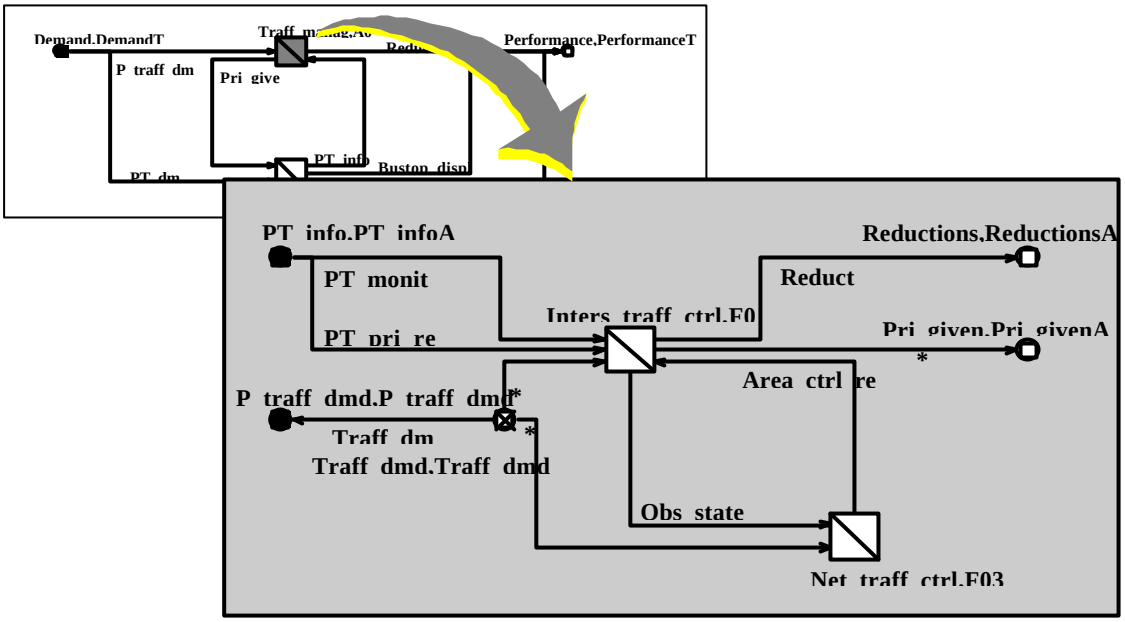


Figure R.4 - Functional Level Modelling

Each Area is then divided into the Functions that it will be making use of, as defined in the CORD Function List: in Figure R.4 we see that, for this ITE system, Traffic Management will be making use of Intersection Traffic Control (Function 3.2) and Network Traffic Control (Function F3.3).). The various data that will be passed between these two areas are then added to the model.

Sub-Function Level Model

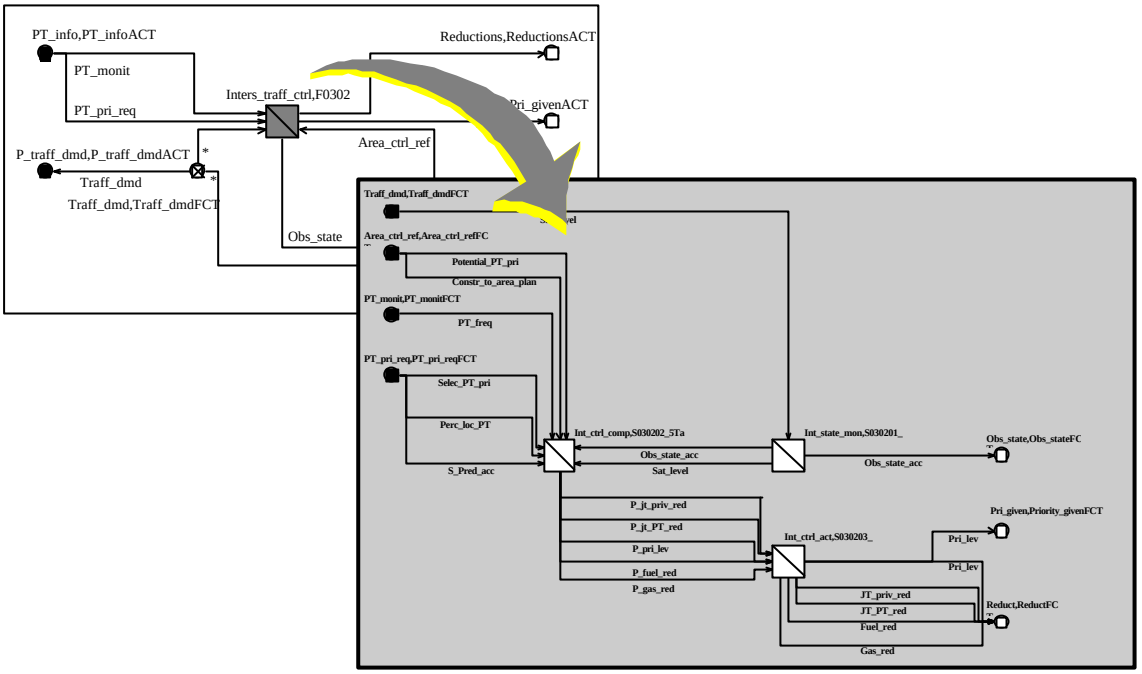


Figure R.5 - Sub-Function Modelling

Each Function is then divided into the Sub-Functions that it will be making use of, as defined in the CORD Function List: in Figure R.5 we see that Intersection Traffic Control will be making use of Intersection State Monitoring (Sub-Function F3.2.1), Intersection Control Computation (Sub-Function F3.2.2) and Intersection Control Actuation (Sub-Function F3.2.3).). The various data that will be passed between these two areas are then added to the model.

Appendix R.2.4 Field Trial Results

Whenever an hypothesis is made or an evaluation is done on an ITE system data should be recorded about the performance of each of the Sub-Functions. Experience has shown that the most effective method of recording this data is to maintain a card for each scenario under investigation. Figure R.6 shows the results from one scenario of Intersection Control Computation from the DRIVE II QUARTET 5T project. It should be noted that if the performance of this sub-function had also been defined for another project, then the designer could choose which of the two sets of data is likely to most resemble the circumstances that will be encountered by the new system.

Inputs		Corresponding outputs	
Sat_Lev	Medium OR High	P_jt_priv_red	16%
Obs_state_acc	Very Accurate	P_jt_PT_red	20%
Pred_accuracy	<20%	P_pri_lev	100%
Perc_loc_PT	100%	P_fuel_red	NULL
PT_freq	Medium	P_gas_red	NULL
Internal Parameters			
Roll_hor	120s		
Comm_reliab	100%		
Req_pred_hor	>90s		
Environment Parameters			
PT_res_lanes	Often		
PT_confl	Few		
Prediction accuracy additionally evaluated for 60s horizon =<10%			

Figure R.6 - Field Trial Data Card

The data from each scenario can then be collated and used to define the input-output relationships for each Sub-Function using a set of basic Building Blocks (BB) to form a flow-chart like structure, or by means of a decision table.

A **repository** of the BBs and Sub-Functions, complete with their field trial data, is being maintained by the CONVERGE-SA project. This can then be used by other projects to build up models of their own particular ITE system.

Appendix R.2.5 Simulation

Simulation takes place as a sequence of steps during which:

- Data is fetched for the input of each Sub-Function
- The Sub-Function processes the input data according to the BB flow-chart or decision table.
- The output data is placed on the Sub-Function outputs
- The state of the Sub-Functions is updated
- The state of the Functions is updated
- The state of the Areas is updated
- The state of the System is updated

During the simulation the user can navigate around the model, interact with the model, read parameter values and record output data.

Appendix R.3 How can I obtain the Architecture Analysis Tool?

The Architecture Analysis Tool is based on Object-Oriented techniques and consists of two parts:

- The CONVERGE-SA System Architecture Tool, which consists of the BBs, the Repository and documentation [Franco 1997].
- The software platform upon which to run it.
- The former is free of charge and may be obtained from the CONVERGE-SA project (Contact: Gino Franco at gino.franco@miz.it).

A license is needed in order to use the software platform. This can be obtained from PrimeSoft (Contact: bruno@polito.it).

Special conditions, reserved for TAP projects, are also available, such as a time limited licence. Contact PrimeSoft for more information.

APPENDIX S SATIN DOCUMENTS

Figure S.1 shows different sources of information developed or used by SATIN that could be used during the system architecture development, the full list of the documents produced by SATIN can be found in Section 8.

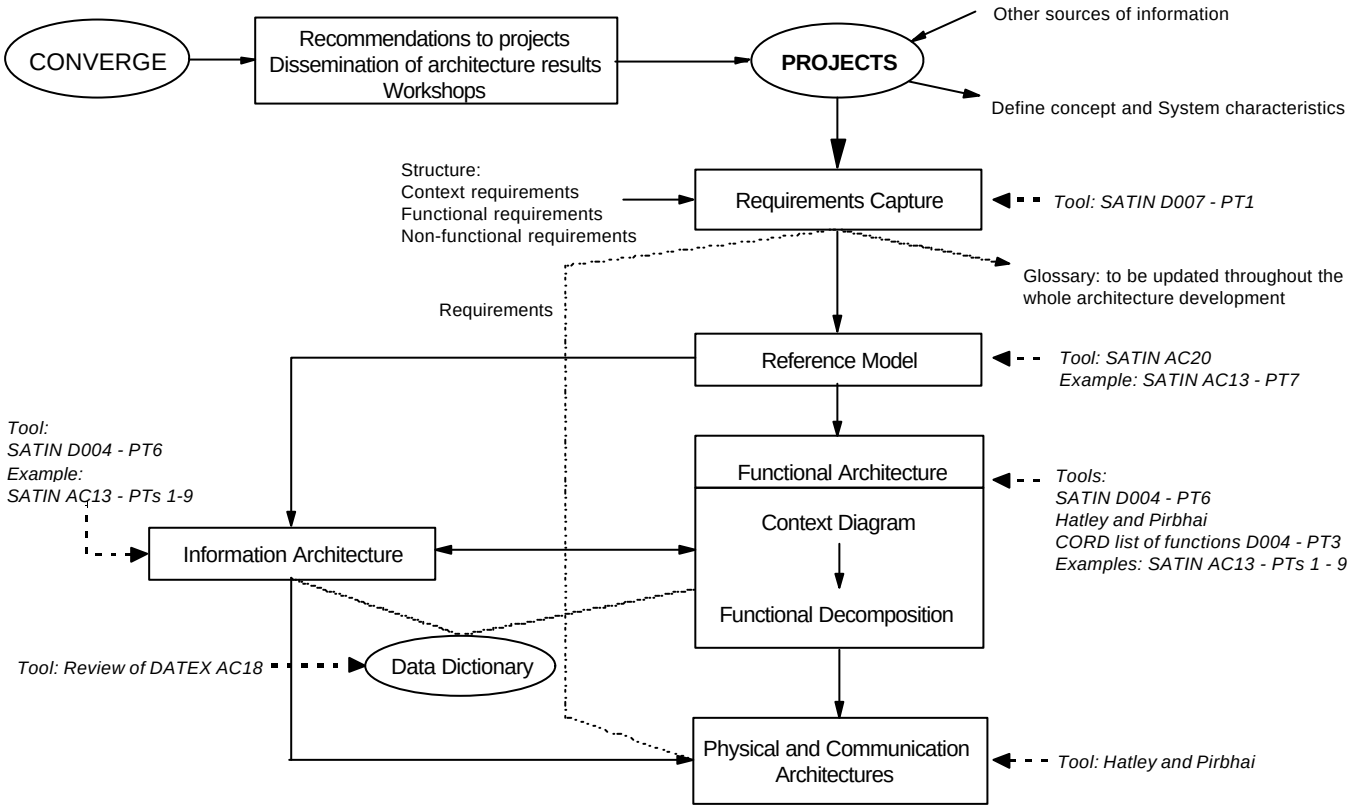


Figure S.1 - Information sources

APPENDIX T WHISPER CASE STUDY

WHISPER
System Architecture

NOTE: This case study provides examples of the outputs from the System Architecture life cycle; it is not intended to be complete.

Appendix T.1 System Concept

Appendix T.1.1 Vision Statement

The management of adjacent shopping precincts have observed that the accessibility of their areas for disabled persons is below what is desired for them; in particular blind people have great difficulty when they are on their own. Because the precincts wish to be open to everybody, they want a technical solution to improve accessibility of their areas, and provide a safe mode of transport for disabled persons, including those who are blind, throughout and between the shopping precincts, both within the shops and between the shops. (For the sake of simplicity the term 'shop' includes all other facilities that are open to the public within the precinct.)

Appendix T.1.2 Identification of Users

- Disabled Persons - who will be carried by the wheelchairs; disabled people may have mobility, visual, auditory or tactile limitations, necessitating the need for special equipment;
- Other Pedestrians - who will share the precincts with the wheelchairs;
- WHISPER Operator - who will supply and maintain the wheelchairs, and the central database of information;
- Precinct Managers - who will provide all the necessary information about the precinct to the operator, and install and maintain the infrastructure;
- Shop keepers - who will provide information to the operator, via their precinct management, about opening times and the good and/or services that they supply;
- Road Authority - who maintains the adjacent road infrastructure;
- Legal and standardisation bodies - legal and standardisation issues to be identified after the system architecture has been developed.

Appendix T.1.3 Mission Statement

To develop a WHeelchair for Intelligent and Safe Portage in Equipped Regions (WHISPER[1]) capable of guiding itself around a series of pedestrian precincts which has been provided with intelligent beacons at suitable locations. The wheelchair will not cause harm to the passenger or to other pedestrians, be capable of avoiding specified areas (e.g. work sites), and of crossing a road at suitably equipped locations without any assistance from the passenger.

[1] All ideas concerning, and the concept of the project WHISPER are exclusive and copyright to the DRIVE II projects PASSPORT I & II (V2057/8)

Appendix T.2 System Characteristics (examples)

Appendix T.2.1 System Overview

Description of the physical environment

The area where the WHISPER system will be applied, the WHISPER area, consists of a number of shopping precincts, and, from the viewpoint of WHISPER, these precincts are separated by public roads, on which motorised traffic is allowed. These roads are equipped with pedestrians crossings, each with a traffic light controller (PC-TLC). Each precinct may have shops located on one or more levels. The total area accessible for a wheelchair will be bounded and the boundary will have to be discernible by the wheelchairs.

The precincts will have special car parking places for disabled persons, at which points WHISPER wheelchairs will be made available. In addition, WHISPER wheelchairs will be made available at selected main entrances, with parking places for non-WHISPER wheelchairs.

Description of the WHISPER wheelchair

A WHISPER wheelchair will be equipped with batteries, equipment to communicate with PC-TLC and the WHISPER beacons, internal storage, processor(s), and a communication interface with the wheelchair user (who may have visual, auditory or tactile limitations). The internal storage will hold a database which contains all the relevant information about the shopping precinct(s), beacon locations and road crossings, the information provided by its user, and statistical information about the use of that wheelchair.

The wheelchair will be capable of navigating to any location within the WHISPER area, as commanded by the passenger, under the guidance of beacons and with the assistance of the PC-TLCs.

The wheelchair will be constructed in a manner that will produce a safe ride for the passenger, and be provided with sensors to detect any inadvertent contact with other pedestrians or objects in order to produce a safe outcome.

The WHISPER beacons

The beacons will provide location data to the wheelchair, together with some information about the immediate vicinity, including that necessary for the safe passage of the wheelchair. Temporary obstructions can be programmed into the beacons as ‘no go’ areas. Beacons are able to signal steps (which may be dangerous, especially to blind people), the status of any lifts in the vicinity, and the ultimate boundary of the WHISPER area.

The Pedestrian Crossing - Traffic Light Controller (PC-TLC)

The wheelchair will communicate with the PC-TLC to negotiate a safe passage across the road. Measures have to be taken that during the communication, which may last for some minutes, the actual wheelchair is not mixed up with another one. The PC-TLC has to ensure, by means of sensors, that the traffic has actually stopped.

Appendix T.2.2 Multi-Disciplinary System Characteristics

Functions:

The principal functions of the WHISPER system are described in the system concept (see above). The communication functions have to be defined, notably the one with the PC-TLC, which is under the ownership of an independent organisation.

Each wheelchair, beacon, PC-TLC and 'shop' will have a unique identification, to be issued by the system operator.

Each wheelchair is able to determine its route through the WHISPER area in a near-optimal way, at least in a manner such that the wheelchair user remains confident in its functioning. The route can be influenced by the user based on the sequence of shops to be visited, the shortest travel time, or a minimum number of road crossing and/or level changes.

Maintainability:

Except for the maintenance of the equipment, the normal operation of the system should not require additional staff. Each wheelchair must return to one of the official parking bays after use for battery recharge and database update (if necessary).

The wheelchairs and the beacons will be provided, and maintained, by the local authority.

The central database will be maintained by special staff. This maintenance will comprise all the changes in the WHISPER area, and the generation of maintenance and management information.

The timely update of the wheelchair databases, and the timely reprogramming of one or more beacons, is the responsibility of the system operator.

Organisation:

Each participating pedestrian precinct in the WHISPER area will provide and maintain its own set of beacons. The PC-TLCs will remain the responsibility of the Road Authority.

Judicial:

Each precinct manager will be liable for damage caused by the improper functioning of the WHISPER system or one of the beacons, including inaccurate information, under his responsibility.

The Road Authority controlling the PC-TLCs will be liable for damage caused by the improper functioning of the PC-TLCs. To this end both parties shall record information about the performance of their respective equipment, and analyse this information regularly for proper operation.

Information:

The WHISPER system will depend upon reliable and up-to-date information, especially about WHISPER area layout and temporary changes. Procedures will have to be defined to ensure this, independent of the availability of key personnel in the shops or the precincts.

The information which is of direct relevance to the passenger must be provided in a user-friendly manner, suitable for a person with the indicated disability.

The system will accumulate information for statistical, maintenance and managerial purposes, also with the intention of improving the operation of the WHISPER system. Information will also be collected about incidents and complaints provided by the wheelchair users, the shop keepers and the pedestrians, but they will not be stored in the main WHISPER database.

Communications:

The form of the beacons must be confirmed with a prototype, to ensure the absence of interference and to ensure the undisturbed communication with the wheelchair.

Communication of the wheelchair with any one of the beacons should not cause any change in the progress of the wheelchair (hence a smooth movement).

Synchronisation with the PC-TLCs is essential for a safe passage. This communication must ensure that the wheelchairs will not be mixed up, that the time for crossing of the wheelchair is adequate, and that waiting times do not become prohibitive.

Electronics:

Power Supply:

Safety:

Security:

Electromagnetic Compatibility:

Mechanical Stability:

Degraded Modes of Operation:

Testability:

Flexibility:

Future Expansion:

The system has to be prepared for future expansion in the sense that new wheelchairs, beacons, PC-TLCs, new shops and precincts may be added to the WHISPER area, with an additional increase in the amount of information in the WHISPER database and increasing demands on the navigation algorithm. A reasonable estimate is an increase by factor of two in any of the above parameters. Such an expansion should not lead to organisational, performance and maintenance problems and should not necessitate a fundamental change in one of the system components.

(Note: some of the factors in the example above are not addressed, but should be considered in any full architectural study.)

Appendix T.3 System Requirements (examples)

Appendix T.3.1 Context Requirements

- It must be possible to implement the WHISPER system in all the pedestrian precincts that are prepared to comply with the commercial, technical and operational requirements.
- The system must be capable of operating with equipment from a variety of manufacturers.
- The system architecture shall support an evolutionary implementation strategy.

Appendix T.3.2 Functional Requirements

- The wheelchair shall be able to communicate with passengers who may have visual, auditory or tactile handicaps.
- The wheelchair shall be able to determine a near-optimal route according to the sequence of destinations specified by the passenger.
- The wheelchair shall be able to communicate with PC-TLCs in order to synchronise a safe crossing of the road.
- The wheelchair shall contain a database, which will contain all the information about the WHISPER area. This database will be used during communication with the user, during route determination and during navigation.
- The wheelchair shall be able to transport persons up to 120 kgs in weight.
- A WHISPER beacon shall be able to communicate with any one of the wheelchairs. Malfunctioning of the communication with the beacon shall be detectable by the wheelchair and be reported to the system operator.
- The wheelchair shall be equipped with motion sensors which will permit dead reckoning over a distance of 100m with an accuracy of 0.5m.
- The beacon information shall be adequate to determine the location of the wheelchair with a precision of 1m.

- The beacons shall be separated from one another by a distance of between 20m and 100m.

Appendix T.3.3 Non-Functional Requirements

Security

- The database can only be modified by the system manager with data provided by authorised persons (only).

Performance

- Communication between the wheelchair and a beacon shall take no longer than 0.5 sec.
- The database in each wheelchair must be updated at least once every 24 hours (not continuously)

Safety

- The system shall prevent one PC_TLC from being confused with another, and the wheelchair shall only communicate with the PC_TLC at whose location it is.
- The PC-TLC shall ensure that the motor traffic has come to a complete stop on both sides of the crossing before permitting the wheelchair to cross. (This condition may need a pedestrian green period to be extended in order to permit the wheelchair to complete a crossing.)

System expansion

- The system architecture shall permit expansions in the spatial dimension and in the various other system parameters by a factor of two, without any degradation of system functionality or behaviour or any necessary change of system components.

Environment

- The system shall be capable of operating in the temperature range -10°C to 35°C, humidity range 10% to 90%, and under all potential rainfall conditions. Use of the system will not be permitted during conditions of snow or ice.

Etc.

Appendix T.3.4 System Boundary

Figure T.1 is PASSPORT Diagram of the proposed WHISPER system. In particular it identifies the system boundary, with the terminators on the left and right locating the points where the system interacts with its environment (i.e. at the boundary of, but within, the system). The diagram shows all the main flows of data necessary to produce the system requirements.

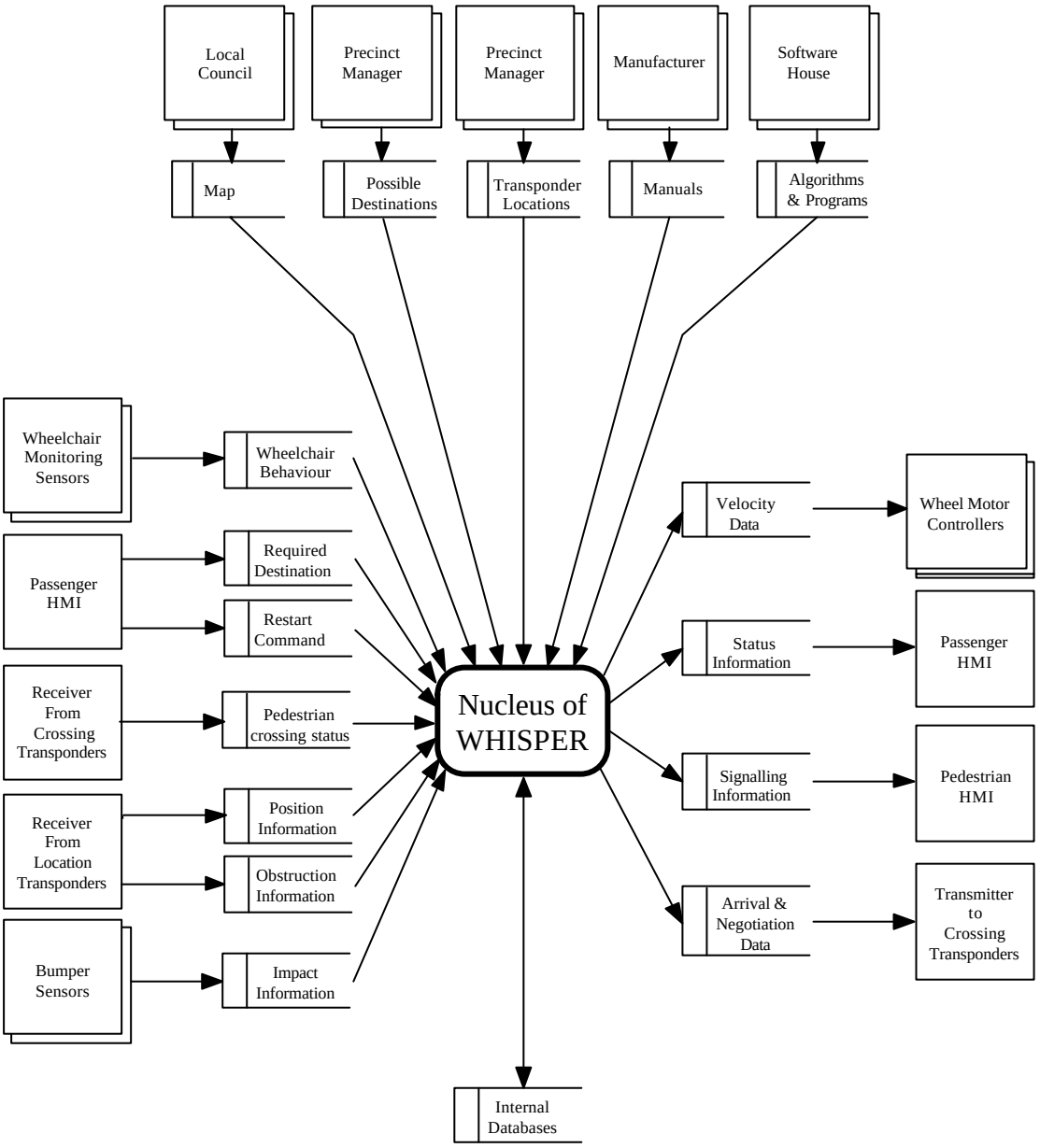


Figure T.1 - PASSPORT Diagram for WHISPER

A full Preliminary Safety Analysis for WHISPER can be found in [PASSPORT D9].

Appendix T.4 Level 3 and 2 System Properties - Reference Models

Appendix T.4.1 Level 3 Layered Reference Model

There are three separate authorities associated with the WHISPER system, with the following major concerns that need to be reconciled between them:

- WHISPER Operator (wheelchairs) - wheelchair navigation and safety
- Precinct Owner (beacons) - wheelchair navigation and safety
- Traffic Authority (PC-TLC) - road traffic safety and wheelchair safety

At a meeting of these three authorities it was decided that, should there ever be a conflict between road traffic safety and wheelchair safety, then road traffic safety would take precedence. It was also agreed that safety would take precedence over normal operation. This resulted in the following Reference Model for the Level 3 Architecture:

	Functions
Layer 2	Navigation
Layer 1	Wheelchair safety
Layer 0	Road Traffic Safety

Appendix T.4.2 Level 2 Layered Reference Models

Wheelchair Layered Reference Model

The functions of the wheelchair were divided into the following Level 2 Layered Reference Model, in accordance with the Level 3 Reference Model. Each layer must be able to perform it function without needing the layer above it. Note that it has been found necessary to split safety into two layers, one needing more facilities than the other.

	Functions
Layer 7	User interface - information display and passenger requests
Layer 6	Route planning and Route Control
Layer 5	Exception Handling - safety and emergency measures
Layer 4	Controlled crossing - with PC-TLC
Layer 3	Autonomous navigation - via beacons and dead reckoning, navigation errors
Layer 2	Database management
Layer 1	Basic Safety - system error detection, emergency stop
Layer 0	Communications - beacons, PC-TLC, wheel drive motors, sensors, data I/O

Beacon Layered Reference Model

The functions of the beacon were divided into the following Level 2 Layered Reference Model, in accordance with the Level 3 Reference Model.

	Functions
Layer 1	Database management - location and vicinity data
Layer 0	Communications - wheelchair, data I/O

PC-TLC Layered Reference Model

The functions of the PC-TLC were divided into the following Level 2 Layered Reference Model, in accordance with the Level 3 Reference Model.

	Functions
Layer 2	Synchronisation protocol
Layer 1	Communications - wheelchair
Layer 0	Normal operation - traffic and pedestrians

Appendix T.5 Level 1 Architectures

Appendix T.5.1 Enterprise Architecture

The various businesses associated with the WHISPER system are related to each other in the manner shown in Figure T.2.

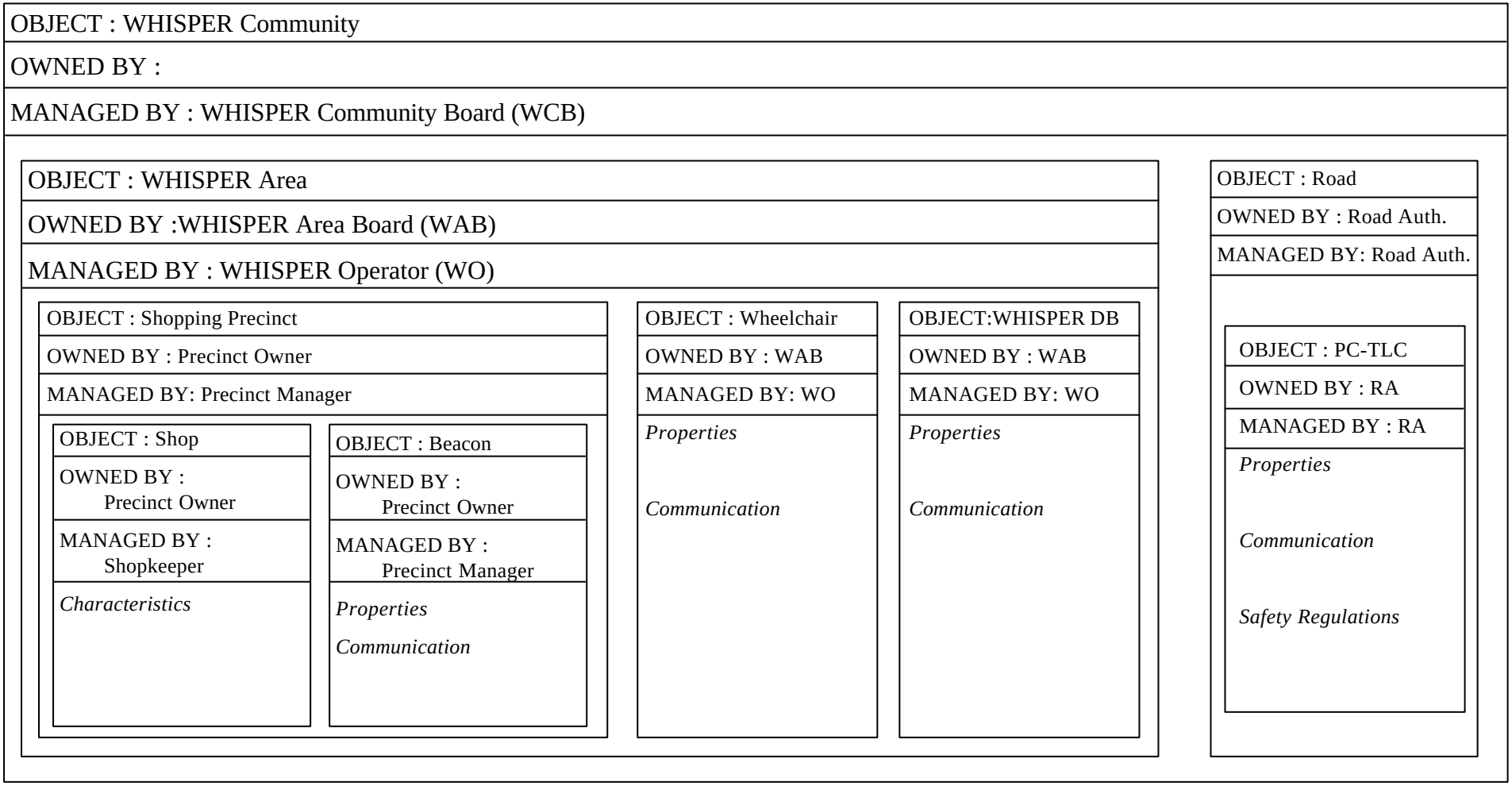


Figure T.2 - Enterprise Architecture for WHISPER

Appendix T.5.2 System Structure - Logical Model

Functional Architecture

Using the PASSPORT diagram (see Figure T.1) and the Reference Model for the Wheelchair, it is possible to produce the top level Functional Architecture for each of the layers. Figure T.3 shows the main flow of data up through the layers and the main flow of commands down through the layers. Each layer either takes in data from the environment, or receives data from the layer below it; sometimes this data is just being passed on from the layers below that one. Each layer performs one or more functions using this data which create output data and/or commands which may be used directly, or passed down for use by the lower layers.

Each layer should then be developed separately and Figure T.4 shows the context diagram for Layer 4 “Controlled Crossing with PC-TLC”. The function has been further decomposed into its sub-functions in Figure T.5. This shows that once the arrival at the correct PC-TLC has been confirmed, the wheelchair makes a request to cross and then waits until it receives a signal that says that it is safe to cross. The act of crossing then begins by passing commands to the Autonomous Navigation layer; the details of the crossing, such as the width of the road at that point, are taken from the Precinct Layout Data. Once the crossing has been completed a signal is set to the Route Determination and Control function so that further commands may be given.

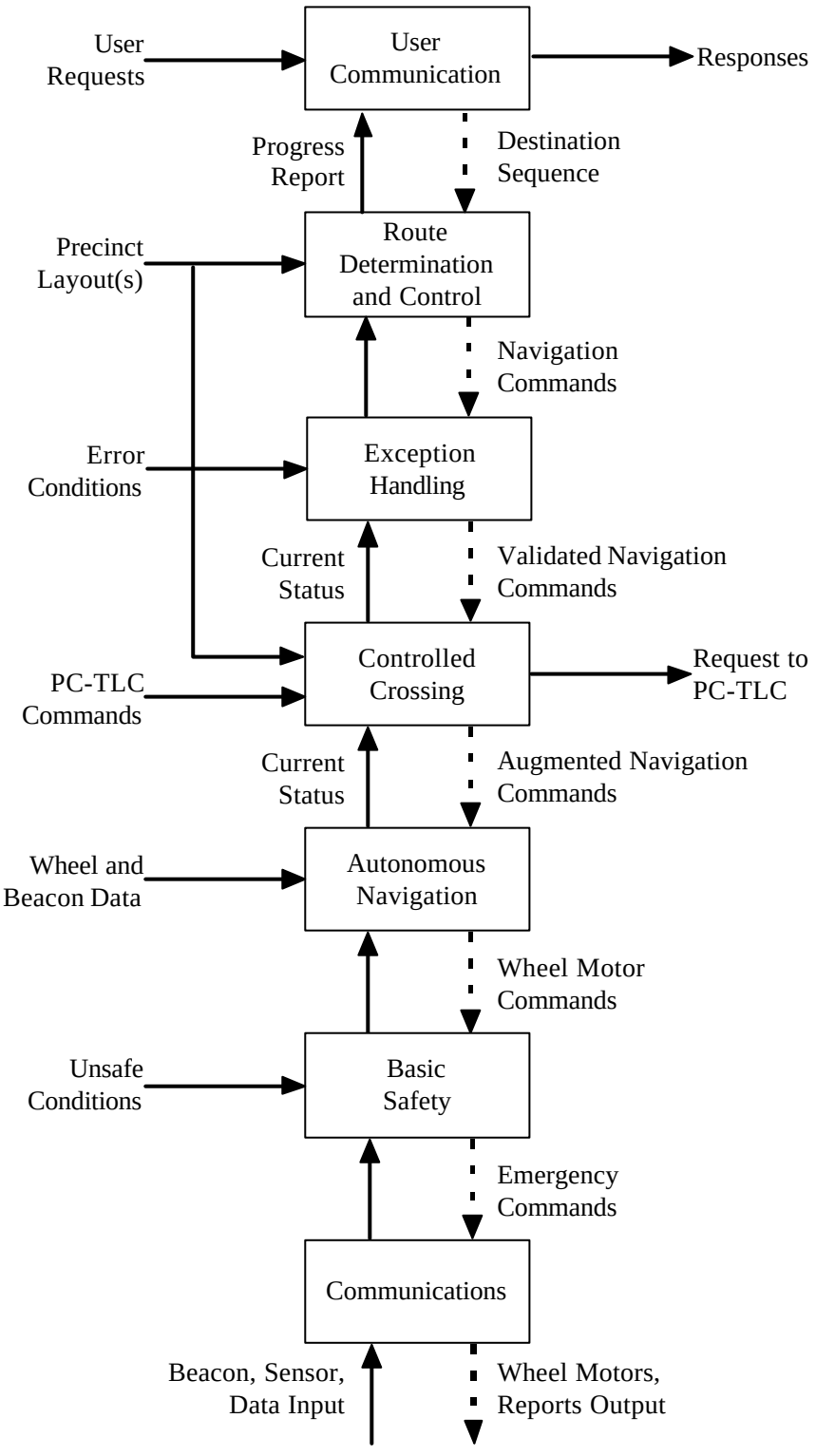


Figure T.3 - Top-Level Functional & Control Architecture for the Wheelchair

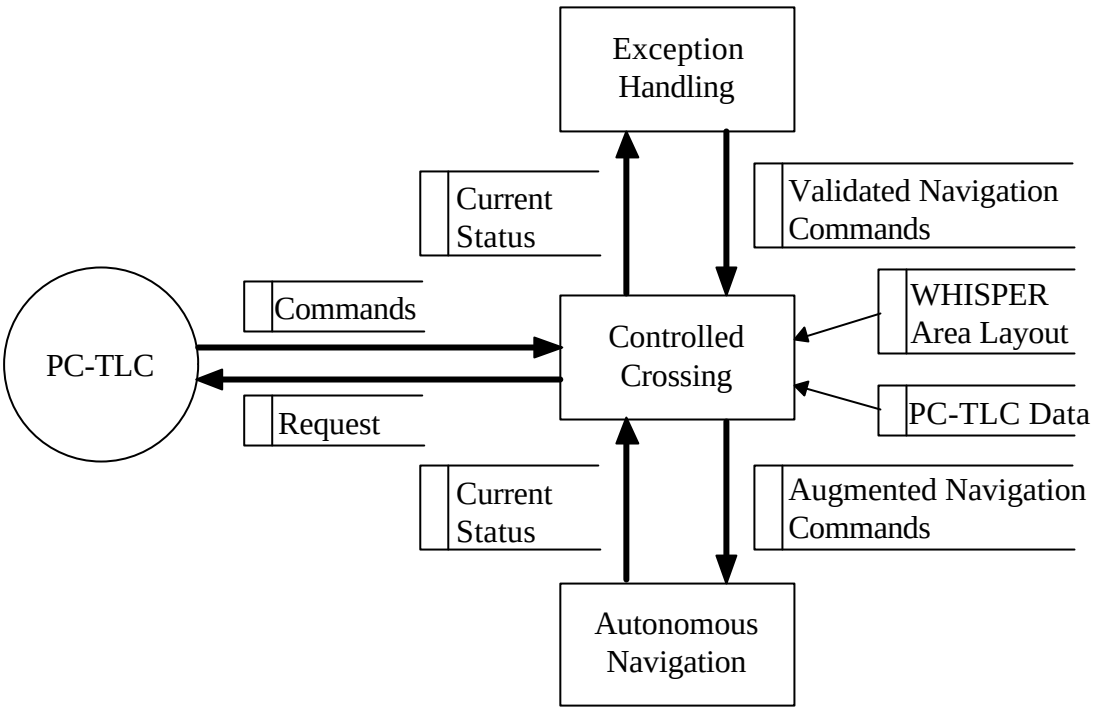


Figure T.4 - Context Diagram for Layer 4

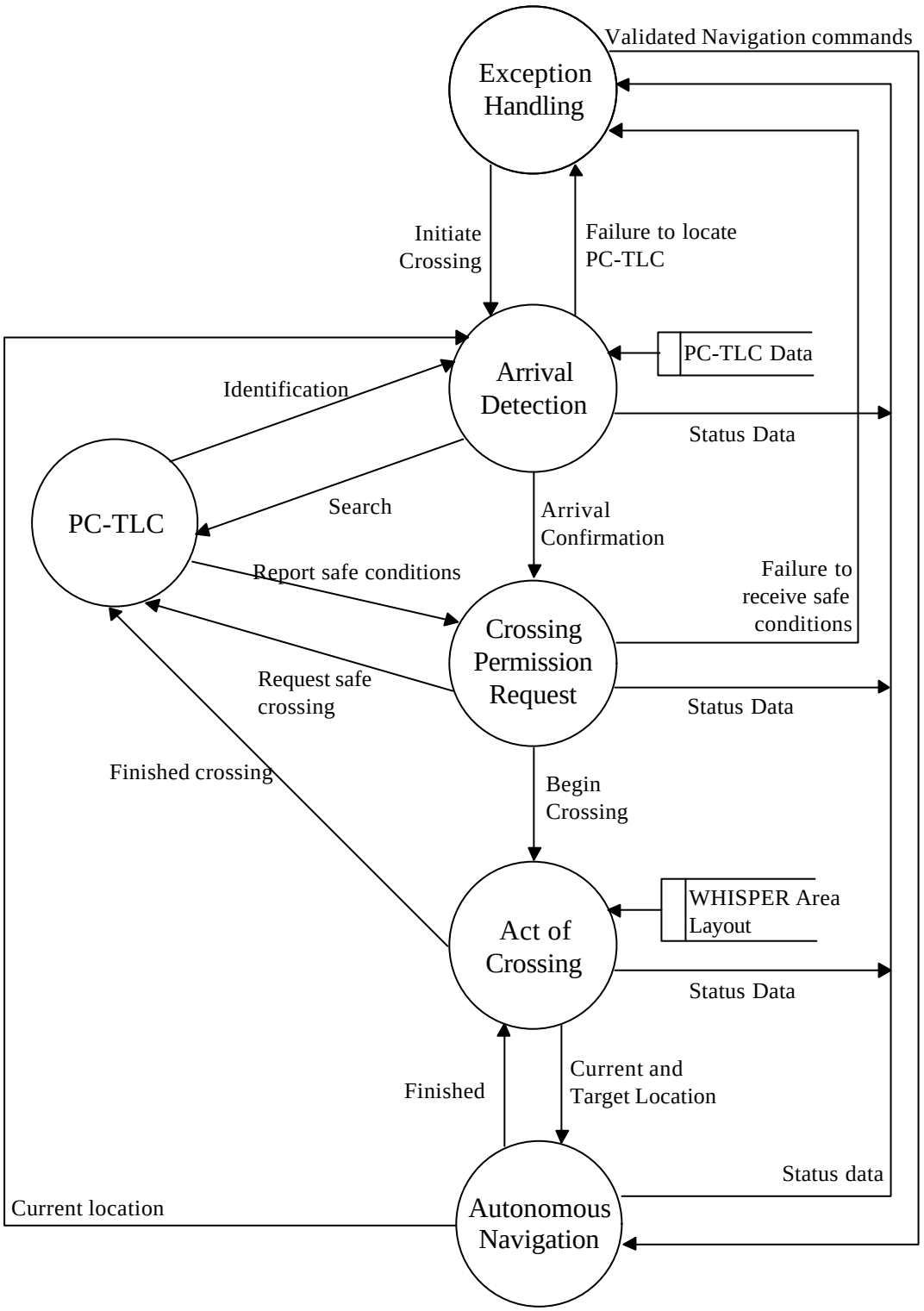


Figure T.5 - Decomposed Functional Architecture for Layer 4

Information Architecture

The main items of data needed for the WHISPER system will be stored in one or more databases. Figure T.6 shows the top-level relationships between the data items without considering their physical organisation.

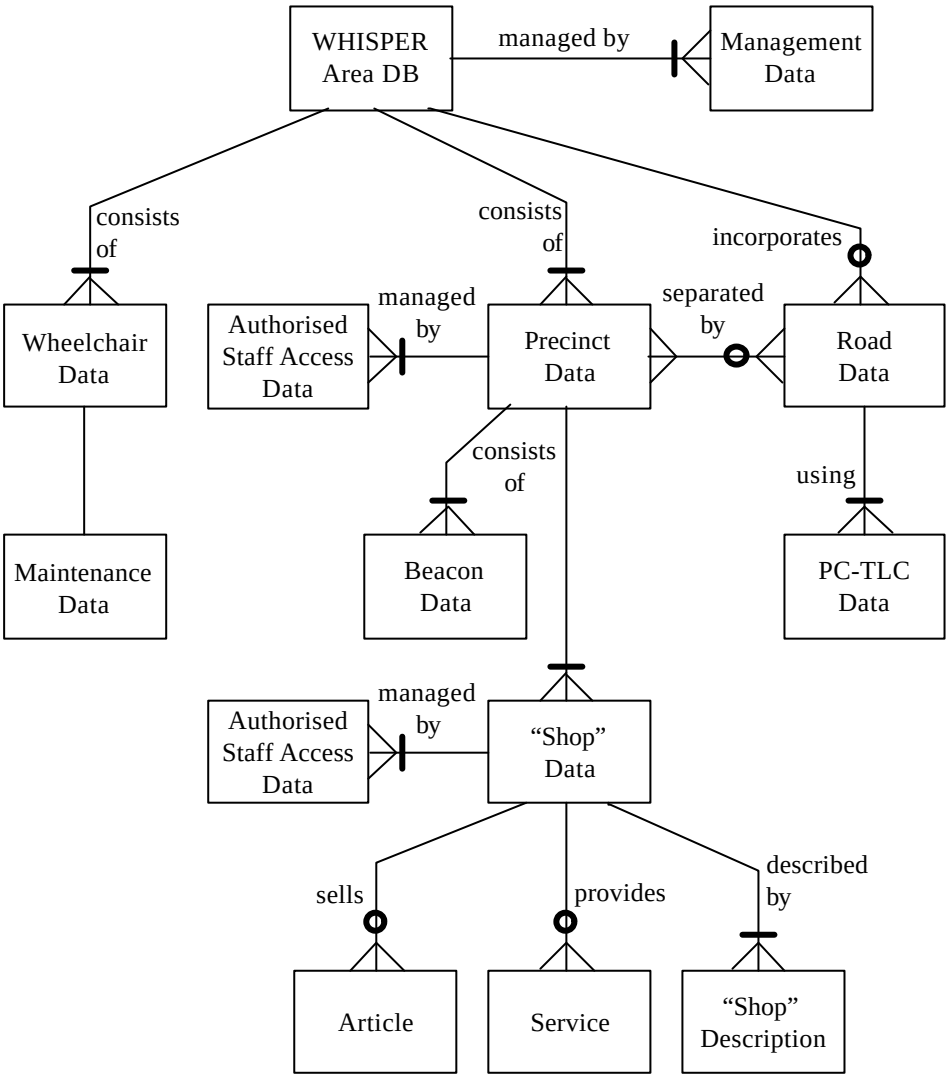


Figure T.6 - An Information Architecture for WHISPER

Appendix T.5.3 Physical Model

Physical Architecture

Figure T.7 shows the top-level physical architecture for the Wheelchair and the Beacons. It shows that the Wheelchair will use 5 processors connected to a bus, with each processor performing on or two of the main functions. The functionality has been split in this manner in order to provide the fail-soft attributes implied by the Wheelchair Reference Model.

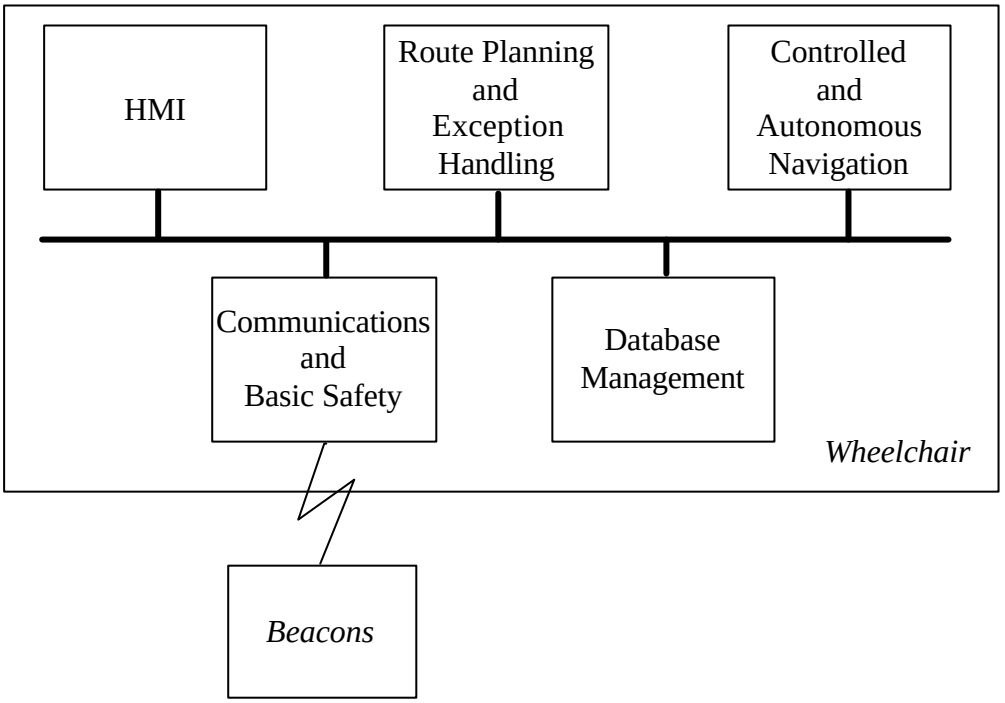


Figure T.7 - A Top-Level Physical Architecture for WHISPER

Communication Architecture

The communication link between a wheelchair and the beacons should include the following properties:

- wireless with a normal working range of 1-2 m; it should have a sharp cut-off at a range 10 m.
- a maximum transfer of 10 KB of data should take place in less than 0.5 sec.
- the link must be capable of operation in dry, wet, dusty and dirty conditions. It need not be able to work in conditions of snow and ice.

Appendix T.5.4 System Structure - Behavioural

The following are some behavioural characteristics that should be exhibited by the WHISPER system. Note that although they are listed here under the heading of "non-functional" requirements, many of these high-level behavioural characteristics will ultimately be implemented by low-level "functional" requirements. At this stage in the life-cycle, it is far less

important as to whether a requirement has been listed as "functional" or "non-functional", than that it has been at least recognised to exist.

Wheelchair passengers:

- a dialogue that is easy to follow and understand.
- the passenger must feel safe at all times. In particular slopes of greater than 10° must be avoided.
- the batteries must be capable of a normal use for at least 4 hours operation without charging.

Other Pedestrians:

- should the wheelchair inadvertently hit a pedestrian it must stop. The speed of the wheelchair must be such that no injury will be inflicted in this situation.

Wheelchair Operator:

- the safety-related sub-systems of the wheelchair should be designed to have either a probability of failure of less than 10^{-2} per year of operation, or less than 10^{-2} on demand. (Safety Integrity Level 2 - obtained from the Preliminary Safety Analysis [PASSPORT D9]).

Precinct Operators:

- the beacons should be designed to have a probability of failure of less than 10^{-2} per year·
- any new information given to the Wheelchair Operator must be available to the wheelchair passengers by the beginning of the next day.

Traffic Authority:

- drivers of vehicles must not detect any change in the operation of the pedestrian crossing when it is being used by a WHISPER wheelchair.

Security:

- each database management system must be able to validated the authority for any changes that are made.
- consideration should be given to the need for the traceability of modifications to the Central database.
- the contents of the Wheelchair and Beacon databases must remain consistent with that of the Central database at all times.

- consideration must be given to a possible conflict between the requirement for mutual consistency between the databases, with the requirement to update the Central database with new information.
- consideration must be given to the possible attempt by unauthorised persons to change the contents of the Wheelchair and/or Beacon databases - possible including the use of physical means of protection.
- it must be possible to detect the physical movement of a beacon without the corresponding change being made to the databases.

APPENDIX U BACKGROUND INFORMATION

Appendix U.1 Introduction

This appendix introduces concepts that have been partly or not covered in the main document. It aims at introducing new concepts that could stimulate the reader's interest.

For a proper understanding of the terms used during the development of an architecture, a set of definitions has to be given to facilitate a common understanding. Each definition is followed by an explanation about what is meant and how the term relates to other terms in the set.

The regular way to present a list of definitions is the alphabetical order, which permits easy access. This organisation has, however, the disadvantage that each definition is presented in isolation and full understanding of the definitions can only be achieved only by studying related definitions simultaneously. So the definitions are grouped in the following categories, which permit easier reading and understanding.

The following categories are used:

1. Traffic control organisation related definitions including:
   a) control regime;
   b) the traffic environment;
   c) reference model;
   d) service;
   e) system;
   f) the ITE.
2. System-related definitions including:
   a) system;
   b) instantiation;
   c) system concept;
   d) system characteristics;
   e) artefact;
   f) proprioception.
3. Functionality related definitions including:
   a) goal oriented functionality;
   b) supporting functionality;
   c) system factors.
4. System architecture related definitions including:
   a) construct.
5. Architectural aspects definitions including:

a) distribution, functional distribution;
b) replication, functional replication;
c) goal conflicts;
d) survivability;
e) local autonomy;
f) continuity of operation;
g) control exertion.

6. System deployment related definitions including:
   a) operations;
   b) management;
   c) maintenance;
   d) change management;
   e) capacity management;
   f) availability management;
   g) problem management;
   h) version management;
   i) performance management;
   j) configuration management.

7. Integration related definitions including:
   a) integration.

8. System requirements related definitions including:
   a) system requirements;
   b) functional requirements;
   c) non-functional requirements;
   d) quality requirements.

9. Additional system engineering definitions including:
   a) expertise domain;
   b) application domain;
   c) system administration;
   d) system archive;
   e) message;
   f) systematic, systemic;
   g) efficacy, efficiency, effectiveness;
   h) holistic view, ontological view.

Note: The European Procurement Handbook for Open Systems (EPHOS), issued by the European Union has a Glossary (volume 2, edition 1994.). Unfortunately, this

glossary is not geared towards system engineering and misses all the terms pertaining to system architecture. Hence the EPHOS definitions are just marginally taken into account.

Appendix U.2 Definitions

Appendix U.2.1 Traffic Control Organisation Related Definitions

a) **Control Regime** - a control regime is an independent part of the real world with a separate control objective and a separate responsibility. A control regime is mostly indicated by the term "management", which indicates the control regime in an enterprise. Examples of control regimes in the traffic environment are inter urban traffic management, urban traffic management, public transport (management), in-vehicle management, freight and fleet management, hazardous goods management, parking management, emergency management and possibly others.

The term control regime is introduced because of its meaning in the traffic context: one of the very fundamental characteristics of the ITE is, that many different control regimes have to be integrated into one system, each regime having separate responsibilities and optimisation criteria, which are sometimes conflicting with the criteria of other control regimes. In the ITE there is not one single top authority, as is normal in the business environment; this system is hybrid also in the control sense.

b) **The Traffic Environment** - the collection of all traffic related control regimes. This embarks upon the various management functions, sometimes with conflicting goals or approaches, that do exist in the traffic environment.

c) **Reference Model** (layered reference model for control regime description) - a reference model is a layered model that represents the control structure embedded in a control regime, each layer in the model fulfilling a sub-goal of the control regime. The structure represented in such a reference model needs to show stability during the life time of the system. Completeness is also a prime feature of such a model. Reference models are not necessarily layered, but in the case of a control regime they usually are.

d) **Service** (or: user service) - a service is a user oriented contribution to the realisation of the goal of one of the layers in a reference model. The collection of services in a layer is meant to consolidate the goal of that particular layer.

e) **System** (ITE (sub)system) - the system is an entity with a predefined objective, introduced to realise that part of a control regime, or collection of control regimes, that can be supported by some form of automation. A system consists of the system artefact and the system factors (see related definitions).

f) **The ITE** (Integrated Transport Environment) - the ITE is the integrated collection of all systems for traffic management and related control regimes.

Appendix U.2.2 System Related Definitions:

a) **System** - a system is a collection of elements, also called parts, that are interrelated and that possess properties different from the collection of properties of the individual parts and show behaviour that does only exist at the system level.

Other definitions emphasise the collection of parts and their interrelationship; the definition given emphasises the emergent properties. System behaviour has been included as well.

b) **Instantiation** (system instantiation) - an instantiation of an architecture is a system, small or large, that complies with all the structures, rules, etc. that are defined in the architecture for that system family. Hence also a subsystem of an instantiation can be an instantiation in itself. Normally the term is applied to those (sub)systems that realise a certain part of the objectives of the overall system. Also the architecture used in the system, the instantiation, can be an elaboration of the general architecture and hence can be seen as an instantiation of the architecture.

c) **System Concept** - the system concept is the basic description of the system as integrated from the perspectives from all contributing expertise domains. The development of a system concept is a basic requisite for the development of system characteristics and system requirements. If this step is not taken, the integration of the different ideas of the various experts will take place during requirement capture and it is much more difficult to resolve the issues there. The more articulated the system concept, the more proficient the following steps can be taken.

The system concept is primarily written from the view-point of the problem owner, the person or organisation that want a certain problem, or part of it, to be solved in a certain manner. The system concept is domain independent and has to be comprehensible for all contributing experts.

d) **System Characteristics** - system characteristics comprise of all the features, properties and inherent difficulties that may have relevance for any of the stages in system realisation, including requirements capture and architecture definition. System characteristics are mainly used to guide the requirements capture process: to warrant that the contributors to the requirements do have a full understanding of all the difficulties that will be encountered somewhere down the line and need to be considered prior to construction, or even prior to requirements definition. They are also used to determine the stable and volatile parts of the system and to support a holistic, multi-disciplinary approach to system development and as a preparation to system factor development.

System characteristics are collected to integrate the insight and knowledge from different disciplines, so that each individual expert understands the main contributions from all other disciplines and realises that all disciplines contribute to the same system; this does not imply that each individual experts understands all the facets of the system characteristics, although the system engineers need to be able to do this. In this way system characteristics represent an early stage of inter-disciplinary knowledge integration.

System characteristics are also useful to identify the necessary expertise domains during requirements capture and also to find goal conflicts in an early stage.

e) **Artefact** - the artefact (or artifact) is that part of the system that will be implemented by means of hardware, software and communication facilities. The artefact is a system engineering term, which is especially applicable during architecture development. It is an abstraction of the physical side of the system; it includes all hardware, software and other facilities for its operation, management and maintenance. In normal speech the term system is applied to the artefact, but this should be abolished because system engineering encompasses much more than just the construction or management of the artefact. This is for instance reflected in the Information Technology Infrastructure Library (ITIL), which deals with much more than just the technicalities of the system. See also system factors.

f) **Proprioception** (or: System Proprioception) - the in-built facility of a system to monitor its internal status and behaviour. It is meant to provide information to the system manager and system maintainer about its internal functioning to enable performance enhancements, for a timely signalling of bottlenecks, to find, localise and identify errors and to identify normal and abnormal behaviour of the system.

Appendix U.2.3 Functionality Related Definitions

a) **Goal Oriented Functionality** - the goal-oriented functionality is the encapsulation of what the providers and the (end) users of the system want the system to do. Based on the objective of the system, which is normally meant to solve a practical problem in whole or in part, the part of the system that is created to achieve this is the goal-oriented functionality, consisting of functions and data. A similar remark can be applied to the artefact.

In the traffic environment the goal-oriented part is that part of the system that deals with traffic management or meets the objectives of any other control regime involved. This is the domain of the traffic control operator, the traffic scientist or the traffic control operators of all possible facilities in the traffic environment, private toll operators and parking lot operators included for example.

b) **Supporting Functionality** - the collection of functions and data in the artefact that supports management or maintenance of the artefact. Because of the creation of the

system, or the artefact, some new problems are introduced because of complexity, size or any other volatile parameter of the artefact. Supporting functionality includes facilities for management and maintenance; it does not include facilities for operations, which belong to the goal-oriented functionality.

The supporting functionality comprises:

- the set of processes, procedures, methods, techniques, tools that allow a product system to be ordered, produced, transported and installed, and have its availability maintained;
- the set of processes, procedures, methods, techniques, tools that allow a living system to be adjusted to the users' needs and requirements.

c) **System Factors** - system factors are that parts of the system that do not belong to the artefact but are necessary to make the artefact operational in some sense or are necessary for full exploitation of the system. System factors do sometimes have some impact on the system architecture and hence cannot be neglected during the development of an architecture. System factors are: staff; organisation, procedures, documentation, office facilities, mathematical knowledge, help-desk, escrow services, etc.

This idea is also expressed by saying that a system consists of hardware, software, com-ware, people-ware, org-ware, proc-ware, doc-ware, etc. to indicate their relative importance to the more core parts of the system; without proper attention to the system factors an artefact, however well developed and meeting its technical objective, is likely to fail in some respect or to show less than optimal performance in some sense.

Appendix U.2.4 System Architecture Related Definitions

a) **Construct** - a construct is a structure built according to specific principles or using a particular set of construction units. An architecture, or its main body like the reference models, will be expressed in constructs. Constructs are built according to certain principles, considerations or other imponderable factors. The validity and feasibility of a construct can be assessed by means of 'what-if' questions and scenario analysis.

By its very nature a construct in the context of an architecture is an abstract entity; in later stages of implementation a construct may be transformed into a physical entity. A construct will be based on a model, typically a reference model which is a representation of some actual or contemplated thing; this model will be enriched with certain characteristics, derived from the requirements, to make it a construct and a candidate part of the architecture.

See also definitions of functional, information, physical and communication architectures in the main text.

Appendix U.2.5 Architectural Aspects Definitions

a) **Distribution, Functional Distribution** - distribution of functionality means that the functions and data in the system are present and can be used at different spatial locations. Functional distribution covers the phenomenon that one single computer system will not be adequate or feasible to meet all the demands for functions and data collections for retrieval or update. Distribution is an architecture related phenomenon, because it can influence both the system's functionality and its behaviour (in the sense of availability, accessibility, user-friendliness, performance etc.)

b) **Replication, Functional Replication** - replication of functionality means that the functions and data in the system can be multiplied and can be used at different spatial locations. Functional replication covers the phenomenon that one single computer system will not be adequate or feasible to meet all the demands for a specific function or data collection for retrieval and updates. Replication is an architecture related phenomenon, because it can influence the system's behaviour (in the sense of availability, accessibility, user-friendliness, performance etc.)

c) **Goal Conflicts** - a goal conflict is a situation in a system or between systems when some of the goals are mutually conflicting or require the same resources. Goal conflicts emerge in a multi-purpose system; the multiple goals in such a system are not necessarily congruent (if they are, the system is not really multi-purpose) or in an inter-operable multi-system environment when the systems chase different goals. If the goals are mutually exclusive priority rules have to be devised and implemented to guarantee the right goals to be realised under certain conflicts.

Goal conflicts are architecture related; they need to be resolved in a proper manner, because otherwise aberrant and undesired system behaviour might be the outcome.

d) **Survivability** - survivability is the continuation of some vital system functions if the full functionality of the system cannot be maintained. For some reason, e.g. absence of some vital sensor collection, or external source of data, the system may not be able to continue full functionality. However, the functionality that is not dependent or not necessarily dependent on these input or facilities has to be continued.

e) **Local Autonomy** - local autonomy is the continuation of some vital system functions at a certain location if the full communication with that location's environment cannot be maintained. Local autonomy is related to survivability, but incorporates the additional problem that the communication with other locations, especially the controlling one(s) are disrupted and that survivability of the local functionality is solely dependent on the ability of the location to act on its own behalf.

Local autonomy is an architectural issue, because it embodies one of the most stringent aspects of system behaviour, namely availability.

f) **Continuity of Operation** - continuity of operation is the ability of a system to permit maintenance, changes and upgrades without (substantial) interruptions of the availability of the system's functionality. Continuity of operation is especially relevant to those systems which exert control over another system which needs continuous, uninterrupted surveillance because of stochastic processes and unpredictable or sudden events and the need to keep the controlled system in the best condition possible. Continuity can be endangered if the system needs to be taken out of service for ordinary maintenance actions; the implementation of maintainability should aim to enable maintenance without system interruption.

Continuity of operation is related to the architecture, because it embodies one of the most stringent aspects of system behaviour namely availability.

g) **Control Exertion** - control exertion is the situation that a controlled system has to be kept or to be guided into a certain predetermined system condition. Control can be exerted by a person, the controller or operator, or by another system, which, in turn, can also be controlled by a person or another control system. Control is based on the extraction of some vital descriptive data from the controlled system and the application of one or more application devices. The accuracy, timeliness, availability etc. of sensors, interpretation and transformation processes and the actuators determine the efficacy of the controlled systems' development into the desired condition in terms of effectiveness, timeliness, accuracy and the absence of oscillations.

The exertion of control is relevant to the architecture of the controlled system, but especially to that of the controlling system, which in many cases will be based on an artefact. The architecture will reflect all the steps to be taken to effectuate the control implied in the process.

Appendix U.2.6 System Deployment Related Definitions

a) **Operations** (traffic management; application of the artefact) - the activities necessary to use the system, or more specific: the artefact, to effectuate the primary goal of the system. In the traffic environment this is the traffic control function, as performed by human operators.

b) **Management** (management of the artefact) - the supporting activities to ensure completeness, correctness, accuracy, timeliness, effectiveness, fine tuning and other qualities of the goal oriented part of the artefact. In the traffic environment Management

concentrates on the traffic related part of the system to keep that part in shape. The background of this function is traffic engineering.

c) **Maintenance** (maintenance of the artefact) - the supporting activities to ensure proper operation of the overall artefact. Maintenance encompasses: configuration management, problem management, change management, version management, availability management, performance management, capacity management, error detection, (hardware) preventive maintenance management, system recovery management, etc.

The background of the maintenance function is system engineering. The specific aspects of this task are indicated to clarify the differences, which are likely to lead to different experts. To support this activity, more dedicated expertise can be involved like hardware, software or communication knowledge. Also the wider concept above the artefact, which is the system, needs to be maintained. The artefact maintenance function has to monitor the functioning of the overall system and to signal deficiencies to the higher management level.

d) **Change Management** - change management is the management of all changes imposed on artefact or the overall system, to safeguard the integrity of the overall system, to limit the cost and effort and to limit the burden to the users and maintenance staff alike.

Information Technology is increasingly being applied in managerial, administrative and technical systems and the size of these systems is still increasing, together with their complexity. As a result, the frequency of changes increases and also the likelihood of adverse results of uncontrolled changes like change errors, the creation of inconsistencies or just postponement of changes because of insufficient tools and staff. Change management is meant to reduce the undesired effects of the need for changes.

e) **Capacity Management** - the management to assure the applicability of the available capacity and the availability of the necessary capacity. Examples of the capacities to be managed are processor usage (possibly related to various applications or over time (day, week, month, year) and the various processors in the system), disc usage (data volume, growth, number of disc accesses, possibly related to time of day, also related to the various discs in the system), network use (related to time of day and backups via the network), server usage (where processor, disc and network use may culminate).

In a system it may happen that available capacity is not properly applied and also that an existing or imminent capacity shortage is reported. Capacity management is meant to address both phenomena and to take appropriate measures that an appropriate capacity is available at any time. It is more concerned with planning and long-term satisfaction of the user needs for system capacity.

f) **Availability Management** - the management activities necessary to meet the required availability levels of the artefact. Availability management covers the aspects reliability, recoverability, serviceability. It is more concerned with the short-term problem solution and satisfaction of the user needs for system capacity. Availability can be improved by the reduction of full or partial system failures, or a faster, easier or more reliable system recovery and also continuation of the basic user facilities.

g) **Problem Management** - the management of problems in the artefact. Problems may comprise of faults (deviations from specifications), undesired effects, deviant behaviour, intermittent errors, unacceptable wait times and other phenomena. The problem management activity encompasses the collection of data about the problem, the analysis of the origin and background of the problem and the suggestion of ways to cope with the problem or ways to collect further information.

h) **Version Management** - version management is the maintenance activity that caters for an adequate management of versions of system elements. Both hardware and software elements may exist in different versions. These versions have to be known to the maintenance staff, because they may differ in maintenance needs and, more importantly, may differ in general behaviour and in fault exposure.

i) **Performance Management** - performance management is the activity to collect performance data of all the components in the artefact to signal performance shortcomings and to take appropriate measures. Performance management is related to capacity management and availability management. It is a short term activity, meant to detect performance bottlenecks, that are causing delays or waiting queues and to take measures to tune the system or to reallocate capacity.

j) **Configuration Management** - configuration management is the activity to keep the configuration fit to meet the changing demands of all the stakeholders and to administer all data pertaining to the artefact.

   Configuration management is the managerial and administrative foundation to various other maintenance activities. In its most simple form it consists of just the configuration database and the facilities to keep this database consistent with the implemented artefact. In this way it is the prime tool for availability management and so forth. In a more advanced way the configuration database can be used to simulate the system, to analyse problems (e.g. performance and other behavioural problems) and to predict potential capacity or performance shortcomings. In a more elaborate form configuration management uses a system administration, in which all data pertaining to the artefact are collected in an integrated form, instead of in isolated administrations, which situation would likely inhibit full exploitation of the available system data.

Appendix U.2.7 Integration

a) **Integration** - integration is the combination of some components, physical, logical or abstract, to realise a higher level structure. Integration comprises the following:

- the combination of the system factors with the artefact to constitute the system;
- the combination of commercial products (commercial off the shelf, COTS);
- the combination of tailor-made or customised products to one of the main functionalities of the system;
- the combination of the goal-oriented functionality with the supporting functionality and the infrastructure (new or existing) to constitute the artefact;
- the combination of independent control regimes into one communicating and interrelated structure;
- the combination of services to realise the goals of a layer in the reference model of a control regime;
- the combination of functional modules to implement a service;
- the collection of data definitions from different sources into one data dictionary;
- the development of one vocabulary.

Appendix U.2.8 System Requirements Related Definitions

a) **System Requirements** (or User Requirements) - a requirement is a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document. (Part of the definition in IEEE Standard Glossary of Software Engineering Terminology).

More specific: The user requirements are the collection of all requirements that apply to the system as a whole, be it the artefact or the system factors.

Warning: The term user requirements addresses the same as the more regular term in system engineering system requirements, but is adopted to comply with the jargon defined in the European Research Programmes.

During further development of the system the user requirements are allocated and realised by the artefact or the factors, or in combination. Two versions of the user requirements can be distinguished:

- the collection of requirements as expressed by each of the separate user groups as elicited during the requirement capture process. This collection is not necessarily complete and consistent, but expresses the main views of the users. This collection is

understandable for the users, but not directly fit for the development of an architecture or the system;

- the collection of requirements after the consistency, completeness and transformation steps for the creation of requirements that can be used for the development of an architecture. Some of the user requirements will affect the system architecture, others will be met during subsequent stages of the system development process.

b) **Functional Requirements** - the functional requirements define the fulfilment of the prime goal of the intended system: the goal-oriented functionality. In a deterministic environment the functional requirements fully cover and describe all the functional elements in the system. In a non-deterministic environment, e.g. when a system has to be functionally expandable or has to follow a certain evolution, then the functional requirements do not necessarily describe the functions at any moment of the life-cycle in the system, but are at a higher level of abstraction. Even if the functions of the start situation are fully known prior to design, this knowledge has to be applied with prudence because first the situation may change during design, and second the situation later on in the system will undoubtedly change (which turns out to be the case in many systems, also deterministic ones.)

c) **Non-Functional Requirements** - non-functional requirements are the requirements that are invoked because the intended system has to operate in a certain environment, has to show certain qualities or has to meet some demands for specific users. The non-functional requirements come from three sources: general quality considerations (quality requirements, source: system engineers, domain engineers, contribution from customers and users); the specific form the system has to adopt (source: the customers) and the requirements imposed because (some of) the users want to use the system in a certain way. Because of the existing road network and other infrastructure, some additional constraints may exist that do not follow from any functional consideration. This can be additional input to the non-functional requirements.

d) **Quality Requirements** - quality requirements define the set of attributes of the intended system by which quality is described and evaluated (adapted from ISO). See ISO/IEC 9126: Information technology - Software Product Evaluation - Quality Characteristics and Guidelines for their Use.

This standard is developed to prevent repetition of earlier shortcomings and to learn from experience. It enforces the consideration of certain aspects of the system early during the development process, because failure to do this can hardly be corrected during a later stage. (This standard applies to isolated information systems, but can also be used for systems with embedded information systems.)

Quality requirements are part of the user requirements and reflect some of the non-functional aspects of the system, which means: mainly the supporting functionality. The set of quality requirements is also indicated as the 'ilities' (or 'illities') because of the fact that all or almost all the names of the quality factors finish with 'ility'.

Appendix U.2.9 Additional System Engineering Definitions

a) **Expertise Domain** - this is the collection of knowledge and insight that exists in a separate branch of engineering. In any system that deals with information technology, except the ones solely dedicated to software engineering, there is the dedicated domain and the (information) system engineering domain. For the IRTE the dedicated domain is the traffic engineering one, possibly with a dozen of separate sub-domains. Proper system development requires the following:

- involvement of all relevant expertise domains;
- absence of expertise overstretch.

The first requirement is obvious, but the second one is often violated because of financial, organisational and time constraints. However, a limited or flawed understanding of an independent expertise domain may deteriorate the result, which deficiencies may not show themselves until at a very late stage. The requirement emphasises that only staff should be involved that has the relevant education and experience and knows how to bring this to the fore.

b) **Application Domain** (Application Area) - an application domain is a subset of the set of all the areas necessary to realise the intended system. An application domain attempts to fulfil the needs for knowledge and methods in a self-contained area.

c) **System Administration** (SAd) - the system administration is the construct in the system that supports all types of system management and maintenance activities, including evaluation, analysis, simulation and improvements.

The system administration is part of the information architecture; the functionality dealing with the SAd belongs to the supporting functionality. Whilst the system dictionary is instantiation independent to a very high degree, the system administration describes the specific artefact and some aspects of the system. However, its goals, structure and contents are to be defined universally to a high degree to guarantee the problem preventing and solving character of this construct. The SAd has no strong relationship with integration with adjacent systems.

d) **System Archive** - the system archive is the collection of all software, imagery, sound fragments etc. that is meaningful to any part of the system or for any of system related activities. Whilst the SD and SAd describe the system in various details, these databases

do not comprise all the various elements that are necessary for all activities around the system. Elements in the system archive are functional modules (software), video fragments, sound fragments, all linked to entries in SD and/or SAd.

e) **Message** - a message is a compound of information for the logical exchange of information between two functions. This definition implies that the message is possibly but not necessarily transported over the network, dependent on the fact whether the functions are on different locations or on the same. This approach improves flexibility: later on the decision of the actual implementation is not hampered by any predetermined configuration.

A message is a way to implement logical communication or logical transmission of data. To explain this point: physical communication reflects the way the information is transferred (audio, visual, braille paper, telephone, etc.) between people (may be systems as well). Logical communication describes the way the information has to be interpreted.

f) **Systematic, Systemic** - in system theory, the terms systematic and systemic have distinct and different meanings, and systemic being the more prominent one. Systematic means that there is a certain pattern or regularity behind a phenomenon, methodical, procedural. Systemic means that it refers to the general characteristics of a certain system, to a trait pervading and common to the whole system. Systemic in system engineering indicates the fact that the whole is more than the sum of the parts, that there are properties that are only discernible at the system as a whole. These properties cannot be allocated to any subsystem and hence are known as "emergent properties".

g) **Efficacy, Efficiency, Effectiveness**:

- efficacy (for: does the system work);
- efficiency (for: amount of output divided by amount of resources used by the system)
- effectiveness (for: is the system meeting the longer term aim)

Efficacy indicates whether the product is working and the immediate needs are met; effectiveness indicates whether the longer term and possibly evolving needs are met, this relates to whether the system is workable; efficiency indicates whether the system resources are used proficiently, which refers to a certain system quality.

h) **Holistic View, Ontological View**

- Holistic: concerning all the parts;
- Ontologic: concerning all the aspects;

These terms are introduced that at some moment in the development process the system has to be addressed as a whole (holistic, all the parts) but also in all its aspects (which is

addressed by ontologic). This is also to emphasise the behavioural side and the emergent properties of the system, not just its composition.