Template for IDA Project (Project Id)

Template for specific development (Contract Id)

Technical Design Document

Issue 1

TABLE OF CONTENTS

0 PREFACE

0.1 PURPOSE OF THIS DOCUMENT

#1 This document is a generic Technical Design Document document for use by IDA Projects. It provides guidance and template material which is intended to assist the relevant management or technical staff, whether client or supplier, in producing a project-specific Technical Design Document document. It is also useful background reading for anyone involved in developing or monitoring the IDA Management System (IDA-MS).

0.2 USE OF THIS DOCUMENT

#1 This Preface is addressed to the users of this generic document and is not meant to be retained in any project-specific Technical Design Document documents based on it.

#2 The remaining sections (numbered 1, 2, 3,...) constitute a template that should be used to construct the project-specific Technical Design Document document.

- *Text in normal case is in the most part "boilerplate" that can be retained, amended or deleted in the document.*
- *Text in* italics *provides instructions on how to complete a section and should be removed once the section is written.*

#3 The template should be used pragmatically, that is - where a section is not relevant it should be omitted. Conversely, the material contained in this document is not necessarily exhaustive; if there is a subject that is relevant to the IDA Project, but is not included in this document, it should still be included.

#4 This document has been prepared using MS Word 97. The following variables are currently recorded as File "Properties" under MS Word. They may be modified by that means or overwritten directly at each occurrence in the document, at the discretion of the user.

a. "Summary" Properties

Title	Type of document (i.e. Technical Design Document)
Author	Author(s) of document
Keywords	Document reference (i.e. IDA-MS-TD)

b. "Custom" Properties

Proj Id	Short mnemonic of IDA Project (set, in this document, to "Project Id")
Project	Full name of IDA Project (set, in this document, to "Template for IDA Project")
Contr Id	Short identifier of contract (set, in this document, to "Contract Id")
Contract	Full name of contract (set, in this document, to "Template for specific development")
Version	Issue number (currently Issue 1)
Date	Date of document (currently 17 January 2001)

0.3 OVERVIEW

#1 *This preface is for information only.*

#2 *This preface will therefore not be retained in the project-specific document.*

#3 *The remaining sections (numbered 1, 2, 3,...) constitute a template that should be used to construct the project-specific document.*

- *Text in normal case is in the most part "boilerplate" that can be retained, amended or deleted in the document.*
- *Text in* italics *provides instructions on how to complete a section and should be removed once the section is written.*
- *Text in* blue italics *includes discussion and specific recommendations on the approach to Technical Design in the IDA context.*

#4 *The template should be used pragmatically, that is - where a section is not relevant it should be omitted. Conversely, the material contained in this document is not necessarily exhaustive; if there is a subject that is relevant to the project, but is not included in this document, it should still be included.*

0.4 BASIS OF THIS DOCUMENT

#1 *This following introductory sections set out an approach to designing systems that may be developed under IDA. It attempts to set standards and create a consistent approach to the design and development of systems across the IDA Programme. It will enable the Programme to benefit from 'economies of scale' and a consistency in the approach to building and deploying systems. Important issues that need to be considered include the architecture of systems, links to legacy systems, contemporary approaches to design (Object Oriented Design), aims for code re-use and the need to develop systems that will work on an operational basis over many years and the associated desire to make such systems easily supportable and affordable.*

#2 *A key point will be to build on the work already carried out in IDA and its predecessor programmes, where a large number of specific 'technical' developments were undertaken looking at, for example, standards for data exchange, such as GESMES, and the introduction of contemporary technologies and infrastructures.*

#3 *The concept of a **Reference Architecture** is also introduced as part of the process of creating an interoperable environment which facilitates the exchange of information between administrations through setting out a number of standard building blocks around which solutions can be assembled. These building blocks or 'components reflect the emerging technologies that should form the technical basis of the IDA developments. These include the Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI) and Enterprise JavaBeans (EJB) technologies that highlight code re-use, scalability and the creation of interoperable architectures around legacy environments.*

0.5 A REFERENCE ARCHITECTURE FOR THE IDA PROGRAMME

#1 *One of the greatest difficulties facing the IDA Programme is how to create an interoperable architecture across European administrations whilst allowing*

individual projects to manage their own IDA developments with sub-delegated authority. Under sub-delegation, it would be possible for sectors seeking to be selected to build elements of the overall IDA Programme to propose their own solutions, or the solutions proposed by their selected suppliers, thereby exacerbating the existing heterogeneous environment, which is in direct conflict with the aim of creating an interoperable architecture.

#2 *In practice the IDA Programme should seek to create what can be referred to as a* ***Reference Architecture*** *for systems – in effect a series of established building blocks on which a variety of systems should be assembled. The Reference Architecture should then become part of the* IDA Architecture Guidelines[1].

#3 The Reference Architecture would take an n-tier, client-server model as its basis, and state that the persistence layer would be implemented through a number of standard API[2] calls to a particular database system. This use of a Reference Architecture would provide some guidelines to organisations tasked with developing individual elements of the IDA Programme and would, in time, move solutions towards a common homogeneous standard – with all of its associated benefits.

#4 Recommendations could be made on the reporting tool that should be used in systems if one were required, the Web Browser could be standardised, as could any security features that needed to be built into the system. The Reference Architecture would facilitate re-use throughout the IDA Programme as individual projects could be arranged to support the development of libraries of components that can be re-used. This will be one of the major potential benefits that could arise from the creation of a Reference Architecture.

0.6 SPECIFIC DESIGN CONSIDERATIONS

#1 Perhaps one of the most crucial aspects of this is the level of concurrency that will be present in typical IDA projects. Thus far projects falling within the IDA Programme have tended generally not to involve large scale, transaction-based computing. The transmission of information from an MSA to the European Union, containing information on – for example – employment statistics, has focused on the need to create common approaches to the formatting of the data and to handling language issues. It is possible that future projects will involve more frequent updates from participating administrations. In this case attention in the Reference Architecture to the loading on computer systems will be necessary.

#2 OOD will also become increasingly used in the IDA Programme as developments seek to create systems that are more maintainable over their deployment lifetime. The knock on effect of this will be an increasing emphasis on the use of Commercial Off

---

1 The IDA Architecture Guidelines were developed in 1999 in co-operation with the TAG (Telematics for Administrations Group) Subgroup on Horizontal and Legal Issues, a group of experts from EU Member States and EEA states. They describe concepts and references for a well-defined, common architecture that supports network interoperability within and across different administrations. Article 4 of the IDA "Interoperability" Decision (No. 1720/1999/EC) defines the requirement to develop the Guidelines as one of the IDA Horizontal Actions and Measures. Article 5 of the IDA "Guidelines" Decision (No. 1719/1999/EC) requires projects to make use of them. An update to the Guidelines is expected in early 2001.

2 API: Application Programming Interface

The Shelf (COTS) packages in the software architecture in areas such as the database, middleware, reporting tools and in the Graphical User Interface.

#3 *IDA projects may well shift from a traditional waterfall development approach to the design and development of solutions towards Rapid Applications Development (RAD). This will have fundamental implications for the design process. IDA project leaders will need to consider the creation of Joint Applications Development (JAD) teams that are made up from software developers and people drawn from the user administrations with business experience of the application area in question. Staged development, through so-called 'timeboxes', offers many attractions to designers and developers of contemporary systems. IDA Project Leaders should give serious consideration to the adoption of this approach as a way of ensuring systems are delivered into operation that meet up-to-date user requirements.*

#4 *Specific attention will also have to be placed in the choice of language for the implementation of the system. Today various options exist, and it is necessary as part of the design process to give very careful consideration to a range of trade-offs that need to be factored into development of a consistent design for the architecture. Languages such as Java, C, C++ and emerging languages, such as PYTHON, each offer different benefits to the developer. Careful consideration has to be given to the choice of which language to implement the system to gain the appropriate long-term benefits through the full lifecycle of the system.*

#5 *IDA has much to gain from adopting a standard approach to these projects based on, for example, the Unified Modelling Language (UML) paradigm, which is rapidly becoming recognised as an industry standard in software development. In the medium term this offers the exciting potential of creating classes or objects that can be agreed as standards to facilitate the creation of the interoperable architecture that is one of the key technical goals of IDA.*

#6 *It is therefore recommended that the IDA Programme also standardise on a number of tools that support development and deployment of software in the projects associated with the programme. To implement systems based on UML, it is recommended that the IDA Programme adopt the Rational Rose development suite and that Java be adopted as the development standard for all non-real-time projects. Where the system characterisation work shows a need for real-time delivery, we recommend the use of C++ as the development language. These will then become part of the IDA Reference Architecture.*

INTRODUCTION

#1 *This section should provide an overview of the entire document and a description of the scope of the system and its intended usage. The scope should also describe external interfaces to the system, external dependencies and provide a brief overview of the 'characteristics' of the system, commenting on aspects such as real-time use, security considerations, concurrency of users etc.*

#2 *The operating system, development language to be used for the system development and any COTS packages that will be used, such as databases etc. should also be referred to in the introduction. This should be sufficient for the casual reader to gain a good appreciation of the key building blocks of the system. The reader should also be introduced to the security measures that need to be included within the system. Where strong authentication models are required this may need to show how aspects such as authentication of users may need to be implemented using PKI for example.*

1.1 PURPOSE

#1 *This section should:*

a. describe the purpose of this document;

b. specify the intended readership of this document.

1.2 SCOPE

#1 *This section should:*

a. identify the products to be produced;

b. explain what the proposed system will do (and will not do, if necessary);

c. define relevant benefits, objectives and goals as precisely as possible;

d. define any security risks associated with the system;

e. be consistent with similar statements in higher-level specifications, if they exist.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

#1 *This section should define all terms, acronyms and abbreviations used in this document. Particular care should be taken to define terms that are specific to the application.*

#2 *Terms covering the development of software using the Unified Modelling Language (UML) approach – which is recommended if an OOD view of the system is to be developed – should be included for completeness.*

#3 *The following is a list of definitions for this template based on the UML approach to system design:*

Class Diagram	*Describes the structure of a system*
Object Diagram	*Expresses possible object combinations of a specific Class Diagram*
Statechart Diagram	*Expresses possible states of a class (or a system)*
Activity Diagram	*Describes activities and actions taking place in a system*

Sequence Diagram	*Shows one or several sequences of messages sent among a set of objects*
Collaboration Diagram	*Describes a complete collaboration among a set of objects*
Use-case Diagrams	*Illustrates the relationships between use cases*
Component Diagram	*A special case of a Class Diagram used to describe components within a software system*
Deployment Diagram	*A special case of a Class Diagram used to describe hardware within the overall system architecture*
System Block diagram	*A diagram showing the major components of the system with its interconnections and external interfaces*

1.4 REFERENCES

#1 *This section should list all the applicable and reference documents, identified by title, author and date. Each document should be marked as applicable or reference. If appropriate, report number, journal name and publishing organisation should be included.*

#2 *The* IDA Architecture Guidelines *(incorporating, if developed, the IDA Reference Architecture), which should provide the starting point for the system architecture, should be referenced. The elements of the recommended architecture which are used should be described.*

#3 *There should also be a reference to the document, which states how changes to this document are controlled. It is recommended that one of the references highlight the system development model that is to be followed in developing the system, such as Waterfall, V model or RAD. This may reference an internal document or an industry standard work on software development approaches. If an Object Oriented Design (OOD) approach is to be followed a reference to the guidelines to be adopted on developing the design should be provided.*

Num.	Title (Applicability & Reference)	Author	Date	Issue

1.5 OVERVIEW

/1 Section 1 is the introduction and includes a description of the project, applicable and reference documents.

/2 Section 2 provides a system overview.

/3 Section 3 contains the system context.

/4 Section 4 describes the system design method, standards and conventions.

/5 Section 5 contains the component descriptions.

/6 Section 6 includes the Requirements Traceability Matrix.

SYSTEM OVERVIEW

#1 *This section should briefly introduce the system context and design, and discuss the background to the project. This section may also summarise the costs and benefits of the selected architecture, and may refer to feasibility studies and prototyping exercises. This section should also describe how the design proposed aligns with the* IDA Architecture Guidelines[3] *and makes use of the outputs of IDA Horizontal Actions and Measures (HAMs).*

2.1 SYSTEM CHARACTERISTICS

#2 *The description of the system should be given in terms of the Architecture of the solution that is being implemented with high level data flows described to set the context of the system, i.e. to look at its external interfaces. This section should also set out to 'characterise' the system describing aspects of its operation that indicate if the system has, inter alia:*

- *to operate in real-time or in bursts, linked to month-end reporting, for example*
- *the nature of the interface to the users of the system*
- *a large number of concurrent users*
- *to be highly resilient or fault tolerant*
- *to provide security features to protect data*
- *to be scaleable and easily maintainable in the future*
- *to have any special back-up facilities to protect important data.*

2.2 SYSTEM ARCHITECTURE

#1 *This section should describe the Architecture of the system, based where feasible on the* IDA Architecture Guidelines.

#2 The system under consideration will perhaps be based upon an n-tier, client-server architecture. It is possible that the architecture will have secure managed interfaces or proxies to isolate systems from illegal access. The Architecture may be a simple client-server system in which web technologies are used to provide forms from a simple server that can be filled in remotely by someone, for example, at a border post. With a simple requirement to collect basic information, which can be entered as it is captured by the user, this two-tier architecture may well be quite sufficient.

#3 The access layer (client) may have to cater for a range of terminals that may eventually need to access information. It is not impossible to foresee the day when WAP access to information may be required to allow the citizens of Europe direct access to certain pieces of information stored within the overall data systems operated by the European Union. With enlargement, IDA systems may have to handle a wide range of so-called 'client' systems ranging from simple database applications (Access) to information being collected from large-scale mainframe applications.

3 Note that the design MUST not contradict the IDA Architecture Guidelines

#4 *Given this likely wide range of means of access, the current emphasis in client-server architectures of breaking down the architecture into several layers will be factors that will have to be addressed in the course of defining solutions that can be rolled out across the increased European Union. The architecture proposed is a tried approach that, when implemented across the European Union, will provide the flexibility to support ever-changing business (often legislatively driven) requirements. Its key feature is that it separates out business logic, client access technology and centrally held data into discrete layers with standard, open interfaces. Today n-tier architectures can often be divided into several tiers as follows:*

a. *client or 'access' tier – which facilitates access by both external and internal clients through technologies such as web browsers*

b. *presentation tier – the layer that accesses information from the other systems as required, and can contain its own 'business rules' for simple processes*

c. *business tier – contains the business information layer*

d. *persistence tier – the database layer, which provides the facilities to store data.*

#5 *Depending upon the solution proposed for the system in question there may be a balance of functions placed into each of these layers. We believe that in the design process it is vital, however, to have a layered architecture as the basis of the solution to facilitate change and adaptation of the system in the future – in other words to protect the investment that has been made in the system as technologies develop in the future. The degree of isolation afforded to elements of the system through adopting a layered approach is significant and can help immunise the project from having to be replaced in full in years to come – some key elements will survive technological developments.*

#6 *Enlargement, both in the medium and long term, could be a crucial factor in the design of IDA systems. Scalability of the architecture could become an issue. With a layered approach the ability to scale to meet future requirements is relatively easy as additional server power can be deployed into the various layers of the system. It may be, for example, that with enlargement, solutions are devised with 'regional nodes' which connect administrations not directly into a central server but into a Regional Centre of expertise in a subject area.*

#7 *Such ideas are already common place in current Community systems such as those connected with EIONET. Here regionally based centres of excellence in the processing of environmental data collect and process information prior to it being forwarded into the European Environmental Agency (EEA).*

#8 *This illustrates a key point in the design of the architecture of the systems. Depending upon where the data is going to be processed and analysed, different physical layouts of servers may be required to provide the levels of service required by the users of the system. It may be that there are local nodes that collect information in each administration, and it is then routed to a specialist node and, from there, onto a central repository which contains the master files where regional factors are considered in order to help establish policy in an effective way.*

2.3 INFRASTRUCTURE SERVICES

#1 *Infrastructure Services should be provided to all applications with a view to reducing the time, cost and risks of development through re-use. To gain full advantage of infrastructure services the requirements of all current applications and the anticipated requirements from future applications should be analysed. These should be fed into the design of the services. The Infrastructure Services can be built incrementally, implementing the most common requirements first, followed by more specialised services. Topics covered in this section might include:*

a. Security

b. Audit

c. Performance monitoring and reporting

d. Error Handling

e. Debugging

f. Logging.

SYSTEM CONTEXT

#1 *This section should define all the external interfaces. This discussion should be based on a system block diagram or context diagram to illustrate the relationship between this system and other systems. The external interfaces should be defined in terms of:*

a. types of data and information flowing over the interfaces

b. the form that data flow takes, e.g. a defined message format such as GESMES

c. the physical medium of the interface, for example a X.25 link, TCP/IP link, a floppy disk, data entered by an operator etc.

d. rates of information flow, covering the frequency of information being updated, the volume of information sent in a message block, how often the information is updated[4]

e. what can happen if the interface fails for some reason, such as the loss of a communications link.

#2 *It is suggested that the reader give some serious thought to the use of the emerging XML standards to define external data flows into and out from the candidate system. The freedom XML offers in defining labels for data enable the user of the technology to create what is in effect their own data description language for the data that is encapsulated by the XML constructs. However, care should be taken to*

a. adhere to standards (de jure or de facto) where they exist

b. make use of the IDA Reference Architecture and the output of IDA HAMs where relevant.

4 In typical IDA projects the external interfaces are communications links that enable MSAs to transmit information into a central facility or to other MSAs. It is the case that information flows can be infrequent, you do not update employment, health or environmental statistics each day for an area of Europe – they occur monthly or less even less frequently. However some information flows, concerning for example Customs related data, may be required to move quickly between MSAs and, if required, to a central database. Time may be of concern if information is to be acted upon promptly.

SYSTEM DESIGN

#1 *This and the following section should provide sufficient information for a developer to produce the system. The detailed content will depend upon the approach to the design process that is to be used.*

#2 *According to one source there are over 30 different object design methodologies and notations of which UML is perhaps the best known.*

#3 *If the project is to follow the traditional waterfall approach, its documentation will differ from a development approach based on RAD or DSDM. Moreover, an OOD based philosophy for the design using UML would be different from one based on another design methodology such as SSADM or Yourdon. Depending upon the approach to be taken the software may be produced using coding statements or it may be automatically generated by an application development tool or a mixture of both.*

#4 *For the example presented in the rest of this section we will assume that new IDA projects move towards the use of UML and OOD based methodologies. One goal of OOD is to create a collection of reusable software objects that can be reused across other IDA projects, in effect offering some significant savings in development costs. With the proposal to create a IDA Reference Architecture a central library of objects could be reused across projects that have similar aims and objectives 'to collect information' but whose application areas (health, tourism, employment, environment, Customs, etc) vary.*

2.1 DESIGN METHOD AND STANDARDS

#1 *The design method used should be named and referenced. A brief description may be added to aid readers not familiar with the method. Any deviations and extensions of the method should be explained and justified.*

#2 *In this paper we give an example of how client-server technologies can be developed on the basis of a given design paradigm that enables web technologies to be applied to IDA projects offering ease of maintenance and savings in life cycle costs.*

#3 *The design standard might need to be different if more than one method or programming language is involved: for example, if some Javascript design and programming takes place in an HTML project.*

#4 *IDA projects are vulnerable to legislative and political change. For example, an agreement might be reached to focus upon a specific health issue across Europe – where concerted action may be agreed – but which requires the data being collected to change. A key feature of moving towards an OOD-based approach is that it facilitates system change.*

#5 *A key feature of the design process is to attempt to isolate the various 'layers' that make up the overall architecture to ensure that changes in one layer have a minimal effect upon other layers in the system, such as for example in the business layer.*

#6 *In following an OOD-based approach it is vital that the design process is able to discriminate between 'objects' and 'classes'. An 'object' is a specific instance, containing a set of data and the methods to process the data. A 'class' is a*

generalised description of a group of objects that have the same data item(s) and method organisation.

#7 *In designing objects there are four important 'cardinal points' to be remembered. These are:*

a. attributes – what the object knows

b. methods – what the object does

c. states – the changes that occur due to process flow

d. events – responses to the outside world.

#8 *Object definitions should be annotated with these items. This will provide the beginnings of a detailed specification document. In addition, each object should be put into a UML diagram set, as this helps summarise the information into small, tight representations of each object in a standard format – there are nine UML diagrams as referenced in Section 1.3. In the course of the process the diagrams will be transformed into a class diagram that represents the relationship between the objects – in effect a blueprint for the structure of the business objects.*

#9 *One area where careful attention to design can reap rewards in the longer term is in the interface between the client and the so-called presentation layer. From its earliest inception in projects using the Smalltalk language the Model View Controller (MVC) paradigm has been one of the most implemented solutions in client-server computing. In implementations based on web technologies, such as JavaServer Pages (JSP) there is much to be gained from separating out elements of the application server architecture. One approach to this is given in an implementation of the MVC paradigm using Servlets or JavaServer Pages.*

#10 *For example, a Servlet (a Java class that operates on the server side to generate output in response to a client request) could be viewed as a 'controller' that is responsible for processing the request from the client – we may assume this request arrives in HTTP or HTTP (S) format. The 'model' is the representation of the data stored in the database or persistence store and the 'view' is the code that decides what to display next in response to the output from the controller. This is described here to encourage readers to look at the MVC paradigm as an example of a way to break out the components of the client-server architecture into building blocks that can be easily modified and maintained, and gain the benefits from an OOD-based approach to the design of the system.*

2.2 DOCUMENTATION STANDARDS

#1 *For a software implementation, this section should contain the standard module header (if necessary) and contain instructions for its completion. In addition this section should define or reference guidelines on the ration of lines of code to comment statements. It may be that these rules highlight specific areas of code where the commentary should be literally line-by-line, as this is a particularly difficult area. Other areas, which may be less difficult, could be commented on a ratio of five lines of code to one line of commentary.*

2.3 NAMING CONVENTIONS

#1 *This section should explain all naming conventions used, and draw attention to any points a maintenance programmer would not expect. A table of the file types and the permitted names or extensions for each is recommended for quick reference.*

#2 *Conventions for naming files, programs, modules, and possibly other structures such as variables and messages, should all be documented here.*

2.4 PROGRAMMING STANDARDS

#1 *This section should define the project programming standards. Whatever languages or standards are chosen, the aim should be to create a convenient and easily usable method for writing good-quality software. If an application development tool is used there may be other conventions that need to be defined, e.g. colour schemes.*

#2 *When programming in any new language, a standard for its use should be written to provide guidance for programmers. This standard may be referenced or included here.*

#3 *Where there are external interfaces, the programming standards for the interfaces required should be referenced.*

#4 *In general, the programming standard should define a consistent and uniform programming style. Specific points to cover are:*

a. modularity and structuring;

b. headers and commenting;

c. indenting and layout;

d. library routines to be used;

e. language constructs to use;

f. language constructs to avoid.

2.5 SOFTWARE DEVELOPMENT TOOLS

#1 *This section should list the tools chosen to assist software development, including testing. The actual software chosen will be heavily dependent upon the language in which the system will be implemented.*

#2 *The list may include:*

a. an application development too;

b. a configuration manager / builder;

c. HTML authoring tools;

d. a word processor for documentation;

e. a tool for drawing diagrams;

f. automated testing tools.

#3 Prototyping projects might make use of an interpretative tool, such as an incremental compiler/interpreter/debugger.

#4 External interfaces may require some of the modules to be pre-compiled.

2.6 OUTSTANDING ISSUES

#1 Provide details of any design issues that remain unresolved at the date of issue of this document. Explain options, pros and cons, and give an estimate of which option is most likely. Outline impact of each option on the rest of the design.

2.7 DECOMPOSITION DESCRIPTION

#1 The software components should be summarised. This should be presented as structure charts or object diagrams showing the hierarchy, control flow and data flow between the components.

#2 If the UML paradigm is used then the decomposition description should make extensive use of the nine 'UML Diagrams' that in effect define the operation of the system.

COMPONENT DESCRIPTION

#1 *For a software implementation, this and the previous section should provide sufficient information for a programmer to produce the software, and for a maintainer, who may not be the developer, to make subsequent changes. The detailed content will depend upon the software tool to be used. The software may be produced using coding statements written by an application programmer. In contrast, it may be automatically generated by an application development tool, or indeed a mixture of both.*

#2 *It is worth reflecting for a moment on the term 'component' and giving some definition as to what a 'component' might entail. The following definitions are offered to the reader:*

- *Client-Based Components – User-centric graphical interface classes and widgets (e.g. Java Advanced Windowing Toolkit, Motif, Swing, Java Beans) implemented with automated tools like GUI builders and testers.*
- *Implementation Components – General-purpose language libraries or bindings that aid in the implementation of a design in a particular language (e.g. JDK, container classes, middleware wrappers, data portability streams), implemented through tools like UML code generators and repositories.*
- *Infrastructure Components – General-purpose processes built for a particular middleware architecture (e.g. CORBA, EJB, and TUXEDO) that can be customised for a specific task (e.g. logging, transactions, queuing), implemented through service and servant modelling tools (e.g. Rational Rose or BEA Webgain Studio).*
- *Architecture Components – Reusable architectural and configuration concepts that are documented and ready for reapplication (e.g. publish/subscribe, Store & Forward, Push) implemented through automated tools like UML modelling tools.*

#3 *Component-based design offers a great economy of effort by encapsulating functionality at the right level. Application components offer re-use and can easily be enhanced. Re-useable components capture the repeated functionality of common system behaviour (such as infrastructure services).*

#4 *The descriptions of the components should be laid out hierarchically. There should be subsections dealing with the following aspects of each component:*

- *5.n Component identifier*
- *5.n.1 Type*
- *5.n.2 Purpose*
- *5.n.3 Function*
- *5.n.4 Subordinates*
- *5.n.5 Dependencies*
- *5.n.6 Interfaces*
- *5.n.7 Resources*

- *5.n.8 References*
- *5.n.9 Processing*
- *5.n.10 Data*

#5 The number 'n' should relate to the place of the component in the hierarchy.

3.1 COMPONENT IDENTIFIER

#1 Each component should have a unique identifier. The identifiers to be used for components should be defined by the project and described elsewhere.

3.1.1 Type

#1 This section should describe the type of component, e.g. task, subroutine, subprogram, package, file.

#2 The contents of some component description sections depend on the component type. For the purpose of this template the categories: executable, i.e. contains computer instructions, or non-executable, i.e. contains only data, are used.

3.1.2 Purpose

#1 The purpose of a component should be defined by tracing it to the software requirements that it implements.

#2 Backwards traceability depends upon each component description explicitly referencing the requirements that justify its existence.

3.1.3 Function

#1 The function of a component must be defined in this document. This should a short description of what the component does and will depend upon the component type e.g. it may be a description of the process or of the data to be stored or transmitted.

#2 More detail will be provided in Processing (see below).

3.1.4 Subordinates

#1 This section should list the modules that are 'called by' this component. The subordinates of a database could be the files that 'compose' it. The subordinates of an object are the objects that are 'used by' it.

3.1.5 Dependencies

#1 The dependencies of a component should be defined by listing the constraints placed upon its use by other components. For example:

- *what operations have to have taken place before this component is called?*
- *what operations are excluded when this operation is taking place?*
- *what components have to be executed after this one?*

3.1.6 Interfaces

#1 Both control flow and data flow aspects of an interface need to be specified. Data aspects of 'non-executable' components should be defined in Subsection 5.n.10.

#2 The control flow to and from a component should be defined in terms of how this component is to be executed, e.g. subroutine call, and how it is to be terminated, e.g. return. This may be implicit in the definition of the type of component, and a description may not be necessary. Control flows may also take place during execution, e.g. interrupt, and these should be defined, if they exist.

#3 The data flow input to and output from the component must be detailed. Data structures should be identified that:

a. are associated with the control flow, e.g. call argument list;

b. interface components through common data areas and files.

#4 If a component interfaces to components in the same system then the interface description should be defined. If a component interfaces to components in other systems, the interface description should be defined in an interface document.

3.1.7 Resources

#1 The resources a component requires should be defined by itemising what the component needs from its environment to perform its function. Items that are part of the component interface are excluded. Examples of resources that might be needed by a component are displays, printers and buffers.

3.1.8 References

#1 Explicit references should be inserted where a component description uses or implies material from another document.

3.1.9 Processing

#1 The processing should be defined by summarising the control and data flow within it. For some kinds of component, e.g. files, there is no such flow. Techniques of process specification include Program Design Language, Pseudo Code and Flow Charts.

#2 Any specific algorithms to be used should be stated or referenced.

3.1.10 Data

#1 The data internal to a component should be defined. The amount of detail required depends strongly on the type of component. The logical and physical data structure of files that interface major components should be defined in detail.

#2 Data structure definitions must include the:

a. description of each element, e.g. name, type, dimension;

b. relationships between the elements, i.e. the structure;

c. range of possible values of each element;

d. initial values of each element.

SOFTWARE REQUIREMENTS TRACEABILITY MATRIX

#1 This section should contain a table that summarises how each software requirement has been met in this document. The tabular format permits one-to-one and one-to-many relationships to be shown.

System Req. Number	System Ref. Item	Component Identifier	Component Item

DOCUMENT CONTROL

Title: Technical Design Document

Issue: Issue 1

Date: 17 January 2001

Author: Dr Dave Sloggett

Distribution: EC DG Enterprise – Gavino Murgia
Project Team

Reference: IDA-MS-TD

Filename: Doc7e17.doc

Control: Reissue as complete document only

DOCUMENT SIGNOFF

Nature of Signoff	Person	Signature	Date	Role
Authors	Dr Dave Sloggett			Project Member
Reviewers	Mark Pillatt			Consultant

DOCUMENT CHANGE RECORD

Date	Version	Author	Change Details
02 January 2001	Issue 1 Draft 2	Dave Sloggett	First complete draft
08 January 2001	Issue 1 Draft 3	Mark Pillatt	Review and update
10 January 2001	Issue 1 Draft 4	Sue Turner	Updating format
17 January 2001	Issue 1	Mark Pillatt	Apply review comment and issue