ESPRIT Project in Confidence

ESPRIT Project 2072 ECIP2

European CAD Integration Project

Prepared by Cadlab W.Mueller

Data Flow for High-Level Design and Specification

D.2.1.Cadlab.24 December 31, 1990

Prepared by: *Wolfgang Mueller*
CADLAB
Bahnhofstr. 32
D-4790 Paderborn
F.R.G.

Phone: ++49-5251-284-123
Fax: ++49-5251-284-140
E-Mail: wolfgang@cadlab.uucp

December 31, 1990

This document was improved and enhanced by the ECIP 2 Workpackage 2 team, namely Bull S.A., IMT, INPG/CSI, Thomson-CSF, Thomson-SINTRA.

Contents

List of Figures

Chapter 1

Introduction

As some research discovering the design flow in several companies has been undertaken it was found that the current practice in industry is that there is no unique formal method applied covering the entire design process starting from the user requirement specification to the system's implementation and support. Actually, in practice there is not even one formal method established in the field of user requirement specification anyway. Often only an informal mixture of manually written text and human interpretable drawing equivalents in connection with the informal knowledge of some engineers are used. The level some formal methods appear first is dependent on the product to be developed. For instance, in chip design mainly the first formal description is an algorithm using a hardware description language or the drawings of a schematic entry only. Software design often starts with an idea of one or several software engineers that try to formalise their ideas by writing some pseudo code which is manually transferred to a programming language. Sometimes the ideas are directly implemented without any further formalisation. In military system design, the first formal description often appears as a simulation model in form of a Petri-net or some kind of control-, data-flow or state diagrams. There are still a lot of design or decomposition steps which are done manually, such as inspection finding faults. However, this shows that there is still are large gap between theory and practice. It might be that the theoretical methods are not used since they are very costly and time consuming or that there are still no complete commercial tools available supporting this methods.

Furthermore, it has been discovered that every company applies its own (formal/ informal) design policy which particularly depends on the tools which are used. There exist few internal papers describing very small parts of the design flow only. At present there do not exist any established complete standardisation for methods and their interfaces used in early design phases (user requirement specification, functional specification, conceptual design). First attempts are made by ECIP in [24].

Beside these problems there are a lot of ambiguities in using the term 'system'. However, the term is defined by the IEEE Standard Dictionary as "An integrated

whole ...", this definition does not clarify any ambiguities anyway. The usage of the term particularly depends on the field of enquiry. For a hardware designer 'system' might be a complete chip (hardware) only. For a software engineer 'system' might be a program or several processes (software), while a military expert use the term 'system' thinking of a model of real things (entities and relationships) and their behaviour, like cities, people, weather, etc.

This document tries to combine the theoretical and practical experience in high-level system design. This report is particularly focused on the engineering domain of electronical digital system (hardware, software, chip, board, computer, computer network). The term 'dataflow' has decided to be the identification of actors, data representation, activities, and reports within the designflow in high-level design and specification.

As many publications refer to software development only, most of the methods can easily be adopted to system design as well. Thus, some parts of the IEEE Software Engineering Standards [46] are introduced in this report.

Most complex systems are developed by several contractors which tend to be a most complicated design process to describe. Thus, defining the interfaces becomes a most important task [45]. It is not in the scope of this report to mention interrelated activities of multiple companies. In this report the specification process is described in its most general form.

Furthermore, it has been decided to structure the description of the concurrent design in a sequential way. The concurrent design process seems to be more understandable if it is been described step by step according to the decomposition of a specification though the activities of the different steps are highly interrelated and have to be managed simultaneously.

This document contains 5 major chapters. The next chapter gives an introduction to a general design methodology called Concurrent Engineering. The following chapters are organised according to the 4 different major transformation processes described in [30]:
User Requirement Specification to Formalised Requirement Specification, Formalised Requirement Specification to User External Specification, Formalised Requirement Specification to Internal Engineering Specification, and the further Design Synthesis.

Chapter 2

Concurrent Engineering

2.1 Definition

To manage the entire system development process 'Concurrent Engineering' is recommended to get a multidisciplinary approach as it is described by the following chapter (see also [16, 17, 31, 43]).

The term 'Concurrent Engineering' (CE) first was introduced by the CALS/CE [1] working groups founded by the U.S. DoD. CE has been applied to system design by several major companies gaining tremendous benefits in the fields of costs, schedule, and quality [31].

CE was defined by the Institute for Defense Analyses (IDA):

> "... a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. This approach is intended to cause the developers, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements."

Thus, it is defined that CE has to include all elements of the product life cycle, neither a listing of action points nor a detailed explanation of the methodology is given by the definition above. Therefore it might be necessary to give a further explanation of the concepts involved in CE.

2.2 Concepts

The term CE has recently appeared, but some of the concepts embodied have already been used for a long time. System development was mainly characterised by a pure sequential relationship of development activities (obsolete waterfall

[1] Computer-Aided Acquisition and Logistic Support/Concurrent Engineering

model), but in reality the design process is highly interrelated in a cyclic way for a long time - for instance, tests and manufacturing constraints can cause several redesigns. It lies in the nature of system development that a practical model has to contain a lot of feedback loops to revise decisions made in former phases, as it is shown in [45]. A typical design and analysis flow is an iteration between a schematic editor and a design analysis tool.

CE involves concepts as, design for reliability and maintainability (R&M), supportability, manufacturability, producability, testability, safety, and quality assurance. It is based on the idea that it is more effective to design the 'ilities' into the products rather than they are installed afterwards.

CE is based on the cooperation of multiple engineering disciplines starting from the beginning of the design process. Wrong decisions made in first design phases caused by multidisciplinary contradictions should be eliminated in early phases to shorten the design cycle and to approve quality. Multidisciplinary design is the design of mechanical systems as well as the design of electronical systems. The engineer who specifies a complex systems has to consider also some physical issues, such as thermal parameters, which can have a strong influence on design considerations.

2.3 Specification & Design Processes

Considering different design processes three different design-methodologies can be identified as they are listed hereafter. The last one builds up the kernel process that is necessary to establish CE. They are ordered to their increasing demand in data and tool integration (see [43]).

Post Design Review/Analysis. A design is specified, implemented, and informally reviewed afterwards. The design is empirically reviewed/analysed to improve quality applying rules of thumb.
The review can deal with simple human interpretable drawings and images represented on paper which should not lack substantial completeness.

Simulation-Optimisation Oriented. After the major parts of the design are completed the design is checked by a detailed discrete simulation. After a completion of the model, optimisation of critical parameters are done by tools to finalise the model. The design is empirically checked during the simulation against quantitative data, such as statistical tables and rules.
Files and other complex computer-interpretable representations have to be handled. A common database is needed to handle different kinds of representation.

Direct Design Synthesis. The specification/design is checked against consistency, reliability, maintainability, manufacturability, testability, etc. from the very beginning of the design process. Quality assurance and project management are as well involved as simulation and optimisation during each step of the design synthesis to decompose the model.

Tools should be fully integrated. A common object-oriented database management system should handle different tools to ease their communication and data exchange.

Considering the above design methodologies it seems to be most practical to mix them up to one unique design process addressing some kind of a 3-phase prototyping. The three design methodologies should be applied during three major specification/design cycles:

- Informal Review/Analysis,
- Analysis & Dimensioning,
- Design Synthesis.

1. Phase: Informal Review/Analysis
A rough design can be developed out of the user requirement specification appearing in the contract. This approximate design should be 'post-reviewed' and checked against the specification. The major goal should be to detect and to eliminate most of the obvious electrical, physical, mechanical, and financial contradictions at a very early phase. This is, to check the electrical constraints against the physical, the mechanical, and the financial constraints and vice versa. This should be done by a multidisciplinary domain expert who is able to apply rules of thumb known from former projects. The field of requirement specification is still very untouched by automation (and no changes are expected for a long period). Thus, especially human interpretable data are handled.

2. Phase: Analysis & Dimensioning
The first step of decomposition has to provide a formalised specification at a very high functional level. The user requirements have to be synthesised to get a first model. This model should be input for some analysis procedures to check the model by consistency. Thereafter the dimensioning of the specification can take place. The specification has to be completed, functional parameters have to be adjusted, and behavioural considerations have to be checked. Thereafter, an approximate optimisation can take place, such as the optimisation of the system's layout. Only design properties that have influence on some contractual considerations should be optimised at this level.

3. Phase: Design Synthesis
A complete synthesis of the system can now be applied. The synthesis has to consider multiple engineering domains as well as the different life-cycle phases, like specification, design, test, support, etc. Approximate data to start the synthesis with can be extracted from the first two development cycles.

Phase 1 is a very informal one and is necessary to be applied in complex system design, mainly. In a more or less sophisticated manner it is applied at the beginning of each specification process anyway. The second and the third phase have been formalised within ECIP2 Workpackage 2 (see [30]). Figure 2.1 shows the entire specification & high-level design process. Actually, there are feedback loops from each node assigning an activity to each activity applied during former steps. But for the matter of simplicity this feedback loops are omitted in Figure 2.1. Three different customer/supplier domains can be identified within this process: User Domain, Requirement Engineering Domain, and System Engineering Domain. Furthermore, four different contractual documentation steps can be fixed:

- User Requirement Specification (URS)
- Formalised Requirement Specification (FRS)
- (Manufacturer to) User External Specification (UES)
- Internal Engineering Specification (IES)

The URS is the input of the *Analysis & Dimensioning* process. The URS is transformed into a FRS that can be analysed and dimensioned. Analysis means to check the FRS by completeness and consistency. Dimensioning covers soft-, hard-, and firmware partitioning as well as the performance analysis of the selected parts. After several design iterations the UES and the IES will be submitted. The UES is delivered to the customer (user) to fix the enhancements and improvements of the user requirements and to give the customer a detailed view of the product that will be produced. The IES is a manufacturer internal specification that is used as an input for the third major design cycle the *Design Synthesis*.

2.4 Specification & Development Domains

A complete CAE environment has to manage at least four orthogonal dimensions [31]:

- **engineering domain** (electronics, mechanics, software, hardware, etc.)
- **life-cycle phase** (specification, design, test, maintenance, etc.)
- **system hierarchy** (system, machine, board, module, chip, etc.)
- **level of abstraction** (behavioural, structural, physical, etc.)

CE requires the integration of different tasks of various engineering domains into a common automation environment which is provided by a framework to maximise tool cooperation and communication. Some changes or restrictions made or recommended by one tool can highly influence engineering design decisions that are important for other tools. Different tools should be fully integrated to access common data and to exchange information via a common object-oriented database. The entire activities are to be guided by a unique design management system.

2.5 Decomposition

The ECIP approach (see [30]) uses an advanced *top-down methodology* extended by additional decomposition into hardware and software. This method progresses from the general to the specific, from 'what' to 'how'. Items at former levels treat item at later progress as 'black boxes'.

Nevertheless, sometimes it might be necessary to use a bottom-up approach for some subactivities, such as the utilisation of prescribed and/or reusable library components.

The process of decomposition should be based on the object-oriented paradigm. This is, the synthesis should be based on the decomposition of entities rather than the decomposition of functions. Some significant guidelines are proposed by Bailin [8] and Booch [12]:

- Identify classes and objects
- Identify the semantics of these classes and objects
- Distinguish between active and passive properties
- Decompose classes into sub-classes ('isKindOf' relationships)
- Decompose objects into sub-objects ('isPartOf' relationships)
- Identify methods and intercommunication

Other references to the object-oriented approach are [6, 8, 13, 14, 15, 58, 62, 67].

In object-oriented design there is a real-time trend represented by Hatley's [45] and Ward-Mellor's [66] strategies for real-time system specification [33], which is best-suited for no-real-time systems as well.

The approach of decomposition can be extended by contractual features introduced by Cohen in [18]. Though Cohen's approach is still dedicated to the obsolete waterfall model some of his ideas are still helpful. In his model the acting persons at each level of specification are considered as customer and supplier, respectively. The supplier at one level may become the customer at the next lower level.

2.6 Knowledge Integrated Engineering

Considering all these different domains a CE environment will contain a huge array of design and analysis tools. The engineer will be overwhelmed by the design and the output of multiple tools. If the decision-making and analysis process is not being structured and supported by any tools the entire design/analysis process can hardly be managed (see [43]). Thus, a lot of research has to be done in the development of decision support systems (knowledge-based systems). Some approaches are known, like Knowledge-Based Software Assistance (KBSA) [41], Knowledge-Based Requirement Assistance (KBRA) [20], and Knowledge-Based

Hardware Assistance [57]. Therefore CE can also decided to be called Knowledge Integrated Engineering (KIE).

By the use of Knowledge-Based Assistant (KBA) life-cycle and engineering activities are machine mediated and supported. All data are machine readable and manipulable from a very early phase. Specification can be made working on-line instead of using paper. This will be a most formalised approach to the entire specification/design cycle activities.
All decisions made are transparent throughout the entire system development and can be revised easily. This is, when any requirement is changed during a later evaluation all requirements and decisions depending on that requirement are changed automatically. Reusability can already be handled at the requirement level, for instance to complete the requirement work automatically (see [20]).

A decision support system can handle very abstract symbolical descriptions. Default values and parameters within the entire specification/design process can be assumed by the system so that the user/engineer can leave out some parts of the description he has not decided on yet [57].
KBA suggests plausible strategies and supports the decision between different design alternatives by comparing alternative designs restricted by some constraints. Thus, this kind of evaluation is also called *Constraint Propagation* [20]. Constraints are described by spreadsheet-like equations, inequalities, and relationships. They also can be defined by the use of metric tables that are only providing a different notation for the same relationships. Constraint networks which link participating parameters have to be created. Constraints can be manufacturing rules or financial and physical laws. The equations and inequalities can be used as starting point to apply linear or integer programming to find an optimal solution. Two different kinds of constraints can be identified [22]: *Quantitative Constraints and Qualitative Constraints.* Quantitative Constraints define inequalities over choice-set attributes. Qualitative Constraints describe relationships among sets of alternatives (see [22]).

After checking the constraints in most cases the data given by the user will lead to several possible valid design alternatives. The different alternative parts can be extracted out of a so-called *Acquisition Device.* Considering the history of previous decisions the system can choose one branch automatically.

The most capable representation for such a Acquisition Device seems to be a taxonomy based upon frames, as it is proposed by [57]. The taxonomy should contain knowledge like already existing components, principles of organisation, and methodologies. In its whole the taxonomy forms a collection of particular features which can be used for Constraint Propagation. One advantage taking a frame-based representation might be that it can easily be translated into an object-oriented language [57].

Beside handling already existing components the taxonomy should be flexible enough to be easily extensible by really new features though it is practically known that designs do not change radically. It is important that such a system

has to contain some kind of a self-learning component so that the knowledge gained from previous designs does not get lost.

It seems that in the field of general decision support there have to be solved some problems first. There is no general technique known yet making a choice between alternatives (conflict resolution) that is automatically provided by the system. The major problem is how to feed the knowledge base with all the knowledge needed for the entire design process. A lot of common sense knowledge is needed to handle the entire design process automatically.

There are two different concepts to conflict resolution:

- **user driven**
  The user has to choose between different alternatives by himself only guided by the system.
- **system driven**
  The system takes the choice according to internal metric tables, rules, and formulas.

In the present state of research a mixed approach seems to be the best solution since it seems to be a very critical task to set up any metrics for the entire specification/design domain. Furthermore, some decisions depending on common sense knowledge should be left to the user.

2.7 Documentation

Military standards set by the DoD have got a lot of influence in the standardisation domain especially in the field of document support and document exchange. There are many tools providing documentation, such as the DoD Standard 2167.

Recently CALS seems to have a major impact on the standardisation efforts, because CALS was set up to coordinate the increasing flow of paper between the DoD and the different contractors. Since August 1988 DoD contractors are asked to submit their documentation using the standards described below.

Thus, CALS has made decisions on the following documentation standards (see [16, 17]). The letters MIL denote the according military standard.

- **SGML** (Standard Generalized Markup Language) - MIL-M-28001 -
  Standard on Textfile exchange, automated publishing of technical manuals (TMs).
- **CGM** (Computer Graphics Metafile) - MIL-D-28003 -
  Standard on 2-D vector graphics used in technical manuals.
- **IGES** (Initial Graphics Textfile Exchange Standard) - MIL-D-28000 -
  Standard on 3-D vector graphics used in TM illustration, engineering drawings, mechanics, electronics, and numerical control.

- **GRP 4 Raster** - MIL-R-28002 -
  Standard on raster scanned images used in technical manuals and engineering drawings.

- **MIL-STD-1804**
  Standard on automated interchange of technical information providing rules for composing files into a complete document. The individual files are to be in compliance with the standards mentioned above.

Figure 2.1: The ECIP Specification Process

Figure 3.1: From the URS to the FRS

Thus, this process can be logically subdivided into a structuring and a transformation phase (see Figure 3.1). First the URS has to be structured and informally analysed by consistency, completeness, and checked against the intended user considerations. Thereafter, the data and information gained from the prescribed contract, interviews, and questionnaires have to be transferred to a clear, unambiguous, and complete specification. It should be clear that these two action

points are highly interrelated. In most cases the two procedures are completely mixed up to one unique activity.

The requirements should not be checked by electronical aspects only, rather than an interdisciplinary check has to be established to verify mechanical, physical, and financial conditions as well.

Beside the specification of the system the specification of its environment and some contractual conditions are to be considered.

Guidelines for the further design process should be evaluated and some recommendations for the further progress should be fixed. The system's documentation can be obtained from the system specification. The test strategy can be gained from the environment specification.

3.2 Actors

Customer: External Customer
Supplier: Requirement Engineer

3.2.1 External Customer

The customer can be classified according to his engineering expertise.

- **Expert**
- **Advanced User**
- **Novice**

According to this classification the external customer delivers different kind of information.

The *novice* usually makes very vague problem statements. He may have only a short impression of the user interface and the system's operations, for instance. He may probably refer to an existing design he has seen somewhere before. He particularly gives some unclear and ambiguous information usually provided by unstructured natural language statements and informal graphical outlines of the problems. This kind of information needs a most detailed analysis.

The *expert* has usually very detailed ideas of the system's behaviour, the user interface and what can efficiently be realised. Partly he can give a more or less formalised specification using more structured language statements, algorithms and clear references to existing standards. He knows formal definition methods. He is able to use structured diagrams and handle tools for high-level analysis.

The *advanced user* has got practice in using computers and tools rather than an expertise in system development.

3.2.2 Requirement Engineer

The requirement engineer tries to clarify and to complete the customer's problem statements. The requirement engineer is an analyst and a model builder [45].

Particularly, he gets some notes by the customer describing the requirements. He has to obtain more information by an interview. Some check lists can be used. The interview can be accompanied by the usage of an expert system. The requirement engineer must be able to handle formal description methods, such as entity-relationship diagrams, control/data flow diagrams, state diagrams, decision tables, petri nets, high-level specification language(s), etc., to build a formalised model from the user requirements.

The requirement engineer makes a first requirement analysis to clarify and to complete the user's statements and to detect logical errors. Usually, the problem statements are manually transformed to a formalised requirement specification to check the statements for consistency and logical errors.

3.3 Data Representation

The representation highly depends on the expertise of the customer in the specification domain.

Mainly, human interpretable data are handled at this stage.

Human interpretable data are passed using plain ASCII files or using de facto document standards, such as PostScript[1], TEX, and roff. On the PC domain MS-WORD[2] is a widely used de facto standard.

The graphical representation mainly depends on the tools that are used for the further process. Examples of widely used methods/techniques are [32]:

- SA (Structured Analysis) [37],
  SADT (Structured Analysis and Design Technique) [56, 21],
- JSD (Jackson System Development) [47],
- Real-Time Modelling [45, 66],
- ERAE (Entity Relation Attribute Event)
  developed within the ESPRIT project ATMOSPHERE, and
- Petri-Net Based Notation.

These methods do not only provide different notations they also provide different ways of thinking.

A mixture of the following more or less formal representations appear at the URS level: Natural Language, Structured Natural Language, High-level Specification Languages, Equations, State Diagrams, Entity-Relationship Diagrams, Petri-Nets, and Control- & Dataflow Representations.

[1] PostScript is a trademark of Adobe Systems, Inc..
[2] MS-WORD is a trademark of Microsoft Corporation.

3.4 Activities

The major activities during this first phase can be classified by structuring and transformation. In most cases these two action points are completely mixed up to one unique activity.

3.4.1 Structuring

3.4.1.1 General

The most complete modelling scenario structuring the URS is described by Hatley-Pirbai and Ward-Mellor called Real-Time Modelling (see [45, 66]).

According to this methodology the technical specification can be classified by: *specification of the system, specification of the environment.* To consider the entire user requirements *contractual considerations* have to be evaluated as well.

The input of this action point are informal and partly formal statements delivered by the customer. The information included in the prescribed contract and gained from interviews and questionnaires have to be structured according to the different domains, such as specification of the system, specification of the environment, and contractual features, as they are mentioned thereafter. Though the structuring of information depends on the unique project this section tries to fix some more detailed guidelines.

A paramount task structuring the URS is to fix the concepts by analysing the language statements and the diagrams delivered by the customer. Conflicts caused by contradictory statements have to be resolved, contractual considerations have to be checked. During this process informal test procedures are established by the requirement engineer applying rules of thumb and applying his experience gained from former projects.

Beside this present state of requirement engineering the use of decision support systems will have an increased influence in this early analysis phase. The customer then will be able to communicate directly with an expert system to fix the requirements in details. This will make the URS analysis machine-mediated to get a most formalised approach to requirement engineering (see also Section 2.6).

Decisions highly depend on a network of relations and constraints that have to be considered. All considerations can be expressed by constraints and equations. Constraints and equations build up a spreadsheet-like network among specification and design parameters. As it it shown in [22] three different classes of considerations are to be identified:

- **Quantitative Constraints**
- **Qualitative Constraints**
- **Equations**

Quantitative Constraints. They can be expressed by inequalities. That is for instance, the total costs should be less or equal than a particular constant amount of money.
Example:

$$HW_{cost} + SW_{cost} + DEVELOPMENT_{cost} \leq 1.000.000$$

Qualitative Constraints. They can be expressed by relationship between different alternatives. That is for instance, the decision taking one user interface that is ported to a particular set of operating systems only.
Example:

$$UI_{Name=X11R4} \rightarrow OS_{Name=UNIX} \text{ or } OS_{Name=VMS}$$

Equations. Setting up equations objective functions can be defined that have to be minimised or maximised.
Example:

$$TOTAL\_COSTS = HW_{cost} + SW_{cost} + DEVELOPMENT_{cost}$$

Using the constraints and equations above two kinds of considerations have to be defined:

- **Technical Considerations**
- **Contractual Considerations**

The technical considerations have high influence on the contractual considerations and vice versa. Thus, they both have to be checked together.

3.4.1.2 Technical Considerations

In view of technical considerations two different domains have to be identified.

- **Specification of the System**
- **Specification of the Environment**

3.4.1.2.1 Specification of the System Usually a lot of references to already existing standards or products are delivered by the user. The user often gives description by analogy. The attributes describing these properties can be classified as *pre-defined attributes*. Some of the requirements refer to new demands. They can be classified as *user-defined attributes* (see [30]).

There are three different kinds of technical description that are delivered by the customer [30]:

- **Behavioural Descriptions**
- **Architectural Descriptions**
- **Technical Constraints**

3.4.1.2.1.1 Behavioural Descriptions. The specification of product features describes rations a system has to perform successfully, often called the system's behaviour.

A system has a *perception space* and an *action space*. After objects are perceived by the perception space, the system acts upon the objects in the action space until the objects leave the action space [66].
Ward-Mellor [66] classify several system components which have to be identified: sensors, actors, processors, and communication links.
The object, their relations, and their interactions can be described by natural language, structured natural language, state diagrams, Entity-Relationship diagrams, Petri-Nets, and control- & dataflow representations.

3.4.1.2.1.2 Architectural Descriptions. Specifying a complex system a customer often describes an architecture as well. Then the system's behaviour is composed by the behaviour of its parts [18].
[45] shows how to adopt an behavioural description to the system's architecture. The following classification of the architecture might be useful: input/output processing, user interface processing, process/control model, maintenance/self-test/redundancy-management processing.
Architectural decisions are highly influenced by technical and commercial constraints.

3.4.1.2.1.3 Technical Constraints.

Technical Constraints can be subdivided by:

- Timing constraints (external response time)
- Performance (benchmarks, ...)
- Safety, Stability, Reliability & Maintainability, and Quality Assurance
- Standards to be met
- Technology
- Extensibility
- Reusability of parts
- Interoperability with other systems (environment)

The technical constraints usually are the most detailed kind of specification that are delivered by the customer. Especially standards to be met, timing constraints, reliability, stability, and safety considerations are paramount issues in critical real-time system design. The timing relates to external timing (response) only. Internal timing is handled in further design phases [45]. The timing specification must be fixed carefully in a timing table which must be part of the requirement specification.

Some of these constraints can be fixed by the use of a spreadsheet-like notation as it has already explained in the introduction of this section. Constraints can be formally fixed using equations, inequalities, and relations.

3.4.1.2.2 Specification of the Environment Every system is *embedded* in an environment. Modeling the environment the same methods can be applied to as for the modeling of the system.

The operational view of the environment should be modeled by a minimum-complexity representation of the relevant portions of the real world. The environment has a high influence on the interoperability, timing constraints as well as physical parameters.

The context of the system should be clearly defined, such as the definition of the system-environment interface and the identification of interactions between the environment and the system (identification of external and internal events) [66].

The specification of the environment and the system-environment interface has to be exhaustively analysed to develop a test strategy and to evaluate stimuli for further testing phases. No structured methods are widely used for the evaluation of test data at this level.

Apart from the operational view the behaviour of the system is affected by several environmental physical properties, such as temperature, humidity, electronical hazards, shocks, and vibrations, that have to be identified (see [30]).

3.4.1.3 Contractual Considerations

Contractual considerations which have a direct influence to the design are commercial constraints, such as special limits for the date of delivery, costs, and availability [30].

Financial considerations mainly cover the overall costs to develop the system. That is, the total amount of money to build, to install, and to support the system. Financial constraints have direct influence on the technical constraints, like technology.

Most of this constraints can be fixed by the use of a spreadsheet-like notation as it has already been mentioned before.

3.4.2 Transformation

The goal is to get a fully formalised specification from the pre-analysed user requirements. Thus, during this phase also ambiguities have to be removed and conflicts have to be resolved that are caused by the informal statements.

Considering the URS the unstructured statements have to be transformed into a structured representation first that can be transformed to a formalised require-

ment specification afterwards. The general actions points of transformation guidelines are listed below.

1. Ask the customer for information
2. Transfer the verbal statements into text
3. Transfer the unstructured natural language to structured natural language statements
4. Transfer the drawings to structured diagrams and structured language statements
5. Transfer the structured natural language statements and the structured diagrams to a formalised requirement specification considering the concepts
6. Repeat the process until some completion criteria is met

Verbal Statements → Text

A verbal statement should be clarified by the transformation into some kind of a textual representation. It is practically shown that verbal statements contain more ambiguities than written ones. A contract can hardly refer to any discussion. It is more convenient to fix all the requirements in textual or graphical representation.

Natural Language → Structured Natural Language

Unstructured natural language statements have to be transferred manually to *structured natural language statements*. Presently, no company uses any formal method processing common sense language at discourse level which can transform the language to a structured representation. "Understanding natural language is an incredibly complex phenomenon" [68]. It is not easy to manage this transformation manually, either. Often a lot of the customer's background knowledge and concepts has to be added during the transformation. Statements have to be expanded by the customer's implicit considerations which are necessary for the complete specification. However, the same terms of phenomena are often used in many different ways when people talk about complex subjects. Some efforts are undertaken to reduce such misinterpretation using a common conceptual model by the ESPRIT projects ECIP1 and ECIP2, called the ECIP Conceptual Model [24, 25, 26, 27, 29] and by ISO/STEP setting up EXPRESS.

Drawings → Structured Diagrams

Often a description is much clearer and easier by the use of a graphical representation than using its equivalent textual representation. Usually, drawings are used to present concepts or layouts. As drawings can be presented in a pretty unstructured form it is necessary to analyse these representations and to fix the

context in a structured textual representation and some kind of diagrams, such as conceptual modelling diagrams, control/data flow diagrams or state diagrams.

Structured Natural Language → Formalised Representation

Structured Natural Language representation particularly includes common nouns, proper nouns, mass/abstract nouns and verbs [2]. A noun identifies an object or a concept, where concepts have to be defined in detail. Current practice shows that concepts are not fully thought through at this level (see also: the ECIP Conceptual Model [24]). Establishing a formalised requirement specification nouns identifying objects can be introduced as objects (data types) and verbs can be introduced as interactions (methods, procedures) or relations between these objects [2].

Transformation of Other Representations

Other representations,such as other formal representations and algorithms, are specified by customers at the expert level only.

Formal representations which are delivered by the customer should be unambiguous by definition. They are to be transformed to an application specific formalised representation as soon as the semantics are clarified.

Algorithms which are delivered by the customer are easy to handle since they should be unambiguous by definition. It is most practical to transform them directly to an executable representation to check their validity.

3.5 Reports

The following documents should be delivered during this first period (see also [46]).

- **System Contract**
  This document clearly defines the technical and management considerations, such as costs, and schedule. The contract should include important data of other documents, namely the Project Plan, Configuration Management, System Requirement Specification, and Environment Specification.
  Status: External Document set up by the customer and the supplier.

- **Project Plan**
  This management document should include the *Quality Assurance* and the *Configuration Management*. This is, the project organisation, the schedules, the resources required, the procedures, and methods to be followed. The inputs are the System Contract and other internal management informations delivered by the supplying company.
  Status: External Management Report.

- **System Requirements Specification** (SRS)
  The customer requirements have to be fixed in this document that is to be used as a fundamental document for the further documentation of the product. The requirements are particularly obtained from notes and from other kind of information delivered by the customer, like interview, questionnaire, etc.. Notes are received in every representation imaginable. It differs from customer to customer and from project to project [45]. The SRS report defines the function the system has to perform and the constraints that have to be met. The inputs are the system's contract and some additional informations delivered by the customer.
  Status: External Engineering Report.

- **System Requirements Review**
  A report on the System Requirement Specification.
  Status: Internal Engineering Report.

- **Environment Specification**
  This document has to describe a specification of the relevant parts of the environment the system interacts. Inputs can be the system's contract and some additional informations delivered by the customer.
  Status: External Engineering Report.

- **Environment Specification Review**
  A report on the Environment Specification.
  Status: Internal Engineering Report.

- **System Validation and Verification Plan**
  This document has to define test strategies and procedures which should be met in the further development to ensure at last the correctness of the system's implementation. Inputs are the Project Plan, System Requirement Specification and the Environment Specification.
  Status: External Engineering Report.

- **System Validation and Verification Plan Review**
  A report on the System Validation and Verification Plan.
  Status: Internal Engineering Report.

- **Decomposition and Methodology Guide**
  This report recommends the procedures and methodology to follow especially dedicated to the current project.
  Status: Internal Engineering Report.

- **User Change Request**
  The changes which are required by the customer or have to be made due to certain constraints should be fixed in this report.
  Status: External Engineering Report.

- **Task Reports**
  Intermediate documents describing the progress that is made during the entire project.
  Status: Internal Engineering Reports.

- **Anomaly Reports**
  Intermediate reports concerning anomalies in specification, schedule, cost, procedures, and methods which are used.
  Status: Internal Engineering Reports.

Figure 4.1: From the FRS to the IES

First the FRS has to be analysed by consistency and completeness. Thereafter design parameters have to be adjusted by dimensioning. During this activity software, hardware, and firmware components have to be identified. The resulting improved and enhanced requirement specification has to be transformed to an internal engineering representation (see Figure 4.1).

Analysis and dimensioning are the tasks overlapping with the process introduced

in Chapter 5. The only difference to the process mentioned in that Chapter 5 is the transformation to a different notation.

There are several iterations between the three different subtasks analysis, dimensioning, and transformation. Intensive feedbacks to former and simultaneous phases have to take place as well.

The input of this process should be a formalised specification that has a well defined semantic in order to achieve precision and conciseness. It should be a very high-level and very abstract representation of the system required, since the evaluation should be done at a functional level.

It should particularly describe what the system is intended to do, not how it should be realised. The formalised specification may include as few structural implications of the model as possible.

A formal specification can be represented by a mathematical notation or a executable operational specification. The mathematical specification can be checked manually applying rules of mathematical deduction. The operational specification can be analysed by simulation and by tools.

4.2 Actors

Customer: System Engineer
Supplier: Requirement Engineer

The actors which are involved in this process are the Requirement Engineer and the System Engineer. They usually belong to one unique company but different divisions.

They are both experts in handling tools as well as in handling formalised notation of the specification.

4.3 Data Representation

The notation of a suitable formalised notation should cover at least the specification of:

- behaviour of the system and the
- performance and/or the timing of the system.

The performance specification covers the constraints on speed, response, and the resource utilisation.

Mainly, machine interpretable data are handled during this process.

The representation of the data highly depends on the tools or methodologies that are used.

A formalised specification is a set of statements in a language of a certain formal system [18]. In this sense a language can be schematics as well as ASCII format or a mathematical representation. Nevertheless, in the following section a distinction will be made between language and graphics though each graphical representation has a formal semantics defined by the implementation of the particular tool.

The semantics of the specification is given by a formal system which can be a mathematical system, an engineering standard, or just simply the implementation of a tool. A formalised specification should be a functional description with no side effects.

Thus, notations can be classified by

- Analytical Representation
- Graphical Representation
- High-Level Language Representation

The first one can be decided to be an algebraic or simply a mathematical notation. The second and the third one are mainly used to describe operational specifications.

4.3.1 Analytical Representation

A analytical notation is used to specify the system by procedural abstraction. This is, the behaviour is not specified by a computation (algorithms).

There are two major approaches using a mathematical notation.

- Axiomatic Specification
- Input/Output Specification

The second one seems to be the more practical one. It has been already applied successfully during several industrial projects. The Input/Output Specification seems to be most practical for the formal specification of sequential systems.

The Axiomatic Specification seems to be practicable for the formal specification of abstract data types.

A third important notation is the specification language Z [61]. This language seems to be used in research projects mainly. Therefore it is not introduced by a separate section.

4.3.1.1 Axiomatic Specification

A axiomatic specification deals with specification of mathematical objects. Relations among the operations are defined over objects. Axioms are set up according to the operations. A complete set of axioms has to be defined to describe each

function. Syntactical properties are described by defining the domain and the range of each function. Semantics are defined setting up a complete axiomatic system. The notation was extented by [42] to specify concurrent systems as well.

The axiomatic notation is used to check abstract data types mainly. That is, the definition of types and operations acting on the types.

The axiomatic approach is well-suited for proofs. The axioms and the theorems which can be derived from axioms can directly be used applying a formal proof.

A very simple example defining the boolean **not**, **or**, and **and** relation is given thereafter (see [18]).

CONSTANTS
true : Bool
false: Bool

OPERATIONS
not: Bool → Bool
and: Bool x Bool → Bool
or: Bool x Bool → Bool

EQUATIONS
not(**not**(a)) = a
true = **not**(false)
a **or** (b **or** c) = (a **or** b) **or** c
a **or** b = b **or** a
a **or** true = true
a **or** false = a
a **and** b = **not**(**not**(a) **or not**(b))

4.3.1.2 Input/Output Specification

The Vienna Development Method (VDM) [48] is a well established representative using a mathematical Input/Output notation. It is now on the way to become an ISO standard. Substantial parts of the standard are ready.

The notation is called I/O specification since two conditions are specified. The first condition specifies restrictions and properties of the input and the second condition specifies restrictions and properties the output has to fulfil.

This approach is based on the denotational semantics. It first was developed to cover sequential information systems only. Later on the Input/Output Specification was adopted to concurrent systems as well. The extensions are made adding some operational notation. Since this extensions are not proofed by practical application yet, the Input/Output specification still seems to be applicable for the specification of sequential systems only.

The I/O specification is a most procedural abstraction. The major advantage using an Input/Output Specification is that no computation has to be defined. No algorithm must be declared how to map the input to the output. Only the preconditions that must hold before and the postconditions that must hold after applying the function have to be defined.

To give a brief overview of this notation a very simple VDM specification defining the maximum number of a set is given thereafter (see [48]).

MaxOf ($s : setofN$)$r : N$
pre $s \neq \{\}$
post $r \in s \wedge \forall i \in s : i \leq r$

The parameters are specified just after the function name. The input definitions are enclosed in parenthesis followed by the output definitions. Thereafter the precondition and the postcondition are specified. The precondition defines considerations to be valid before the function MaxOf can be applied. The postconditions define the conditions that must hold after applying the function. Conditions are defined by the use of static variables which can be universally quantified.

In more advanced applications more sophisticated features of the VDM notation can be used. For further explanation see [48]. Invariants can be defined by using VDM and the specifications can be verified by the use of methods known from mathematical proofs.

4.3.2 Operational Representation

Since the mathematical notation mentioned above describes the properties of functions only, the operational specification describes explicitly the transformation by giving the computation of the intended behaviour. The operational specification at this level differs from the implementation by the means of a simple rather than an efficient description. Thus, setting up an operational specification it is not only possible to describe what should be realised, it also has to be described how it should be realised in a most abstract way. In most cases the operations being defined in the first formalised specification are not suitable for further decomposition. Often they have to be replaced by more efficient computations.

A most efficient operational specification is an executable specification. At present, the most widely used formalised specifications are programming languages and graphic-oriented notations of design tools providing a graphical user interface. The executable specification can be analysed and checked using a tool, which is chosen by the requirement engineer.

Operational specifications can be classified by:

- Graphic-Oriented Specifications

- Language-Oriented Specifications

4.3.2.1 Graphical Representation

A graphic-oriented specification can also be decided to be tool-oriented or method-oriented. There are several tools which support some popular design methodologies. The most practically used methodologies which are supported by tools are listed thereafter. All in common, the tools are based on a graphic representation of the underlying language.

The most important methodologies are:

- Petri-Nets
- Structured Analysis
- System Design
- Real-Time Modelling
- Conceptual Modelling

Petri-Nets. Design tools based on Petri-Nets always use some extensions of Petri-Nets. Petri-Nets are especially dedicated for the modelling of concurrent behaviour. High-Level synchronisation can be modelled easily.

Structured Analysis. Structured analysis is based on the development of Data Flow Diagrams. Some systems that handle structured analysis are able to handle SADT as well [21]. Using SADT facilities it is possible to model the control flow, too. No timing specifications are supported explicitly. For further explanation see [70].

System Design. The most famous representative is the Jackson System Development model (JSD) [47]. The JSD covers the specification of entities, actions performed/suffered by entities, and time-orderings in which these actions can occur. No exact timing specifications can be handled by this model.

Real-Time Modelling. Real-Time Modelling was first introduced by [66]. [45] gives a more sophisticated introduction to the methodology. Real-Time Modelling is based on the specification of Control/Data Flow Diagrams. Control Specification (CSPEC) can be defined by using a state transition diagram, decision table, state event matrix, and a process activation table. Timing can be modeled as well as high-level synchronisation.

Conceptual Modelling. Conceptual Modelling, sometimes called *Information Modelling*, is mainly based on some kind of Entity-Relationship Diagrams extented by rules/constraints. Some important notations are the ECIP Conceptual Model [24], ERAE (Entity Relation Attribute Event), Extented Entity-Relationship Diagrams, IDEF1X, and EXPRESS.

4.3.2.2 Language Representation

In industry there are some widely used high-level programming languages for specification, such as Prolog, Smalltalk, and recently VHDL.

All such widely used high-level specification languages lack some high-level mutual exclusion mechanisms, such as semaphors, critical sections, or monitors. This concepts have to be modelled using low-level synchronisation constructs at present.

Prolog. Using *Prolog* complex systems can be specified in a very efficient way. It is often decided to be an ideal language for high-level prototyping. To describe concurrent behaviour some extensions of Prolog, such as Guarded Horn Clauses (GHC) and Flat Concurrent Prolog (FCP), are available. These extensions seem to run more efficiently since they don't resolve their rules by backtracking. Recently a new version Prolog III was introduced. Prolog III has some advanced features, such as replacing the concept of unification by the concept of constraint solving, a complete treatment of two-valued Boolean algebra, a treatment and processing of relational operators, e.g. $<, \leq, >, \geq, \neq$ (see [19]).

No extensions have been made towards timing specification yet.

Smalltalk. *Smalltalk* might be such popular because it provides a complete development environment. Apart the most convenient handling of the Smalltalk system and the powerful library the Smalltalk language also provides some mighty concepts, such as data encapsulation and inheritance. Extensions of Smalltalk have been developed to specify concurrent features as well. Smalltalk IV was enhanced by timing specifications.

VHDL. *VHDL* is listed here as a representative of an existing hardware description language since it is been standised by the IEEE. VHDL seems to be the most complete language for system development. It first was intended to be a standard for a hardware description language, but it seems to be suitable for system specification as well. VHDL is developed for simulation use. Thus, concepts that support synthesis are missing. VHDL supports the specification of complex system by a multi/mixed level description. Hardware can be specified as well as software. At high-level specification algorithmic features can be used. Applying some steps of decomposition some low-level descriptions can be inserted using the same language. Thus, taking VHDL for the Internal Engineering Specification it can be used for the further design process. VHDL supports the specification of timing and concurrency as well as the handling of libraries. Recently, a lot of efforts are made by several major companies to build comfortable environments for system development. VHDL lacks the concept of inheritance, which is likely used by Smalltalk designers.

4.4 Activities

The input of this process is the FRS that is delivered from the first process described in Chapter 3. There are three major different activities that have to be applied to the FRS. First the FRS has to be analysed by consistency and completeness. Thereafter design parameters have to be adjusted by dimensioning. During this activity software, hardware, and firmware components have to be identified. The resulting improved and enhanced requirement specification has to be transformed to an internal engineering representation (see Figure 4.1).

The reader should note that the entire specification phase as well as the sub-activities analysis, dimensioning, and transformation respectively form a most iterative and simultaneous process.

4.4.1 Analysis

The items the requirement engineer has to deal with depend on the requirements of the external customer prescribed in the contract.

There are two major approaches to the analysis of the FRS:

- Formal Proof
- Simulation

The most applicable notation for the use of a formal proof might be a mathematical notation as it is described in Section 4.3.1. The mathematical specification can be checked manually applying rules of mathematical deduction. Though manual checking is most time consuming, it seems to be the only way to check all kinds of specifications by a formal proof. A lot of efforts are undertaken to automise the procedure of a formal proof but there are still some fundamental problems to be solved so that a formal proof can be fully done by a machine.

The most widely used analysis methods using simulation deal with executable operational specification only. The executable specification is checked by tools. This kind of representation seems to be used more frequently in industrial projects.

Since some formal proof is prescribed by the contract, it seems to be the best solution to take a mixed approach of formal proof and simulation. Some parts can be proofed formally and the overall structure of the system, especially the concurrent interactions, can be checked by simulation.

4.4.1.1 Analysis by Formal Proof

A complete checking by formal proof seems to be applicable by the use of a mathematical notation only, since this kind of description fits best to the methodologies that are used, such as the denotational or the axiomatic approach.

To apply a formal proof it seems to be the most practical approach to take an Input/Output specification using VDM (see Section 4.3.1.2) since this notation supports the mathematical deduction. For further explanation see [48].

A formal proof is a way to deduce formulae. Instances of axioms are used and rules of interference are applied to generate true sequences from another. It is intended to keep all steps of deduction as mechanical as possible. A mixture of two major proof styles are recommended by [48]: Propositional Calculus and Predicate Calculus.

An operational specification, such as a language-oriented specification, is much harder to proof. Operational descriptions have to be proofed statement by statement. An equivalent algebraic expression similar to the algebraic specification for the semantic of each statement has to be defined.

4.4.1.2 Analysis by Simulation

Simulation is a tool-oriented analysis. The main disadvantage taking this approach is that there can never be an exhaustive test of the model. Checking by the use of a simulator is more applicable using operational notations so that the specification is an executable specification.

The simulation activities are based on the environment specification defined at the URS level. A suitable simulation methodology can be gained out of the environment specification. Furthermore, the environment specification helps to set up some stimuli for further simulation.

The formalised specification is checked by completeness, consistency, and reliability.

In current practice the simulation is done using tools only. Thus, consistency and completeness can be checked by a behavioural simulator.

Reliability can be checked by a functional/dysfunctional analysis.

Tools recently available support techniques, such as Behavioural Simulators, Fault Simulators, Failure Mode Effect & Critically Analysis (FMECA), Predictive Reliability Evaluation (PRE), Acquisition and Logistic Support Analysis (LSA), and Reliability & Maintainability Evaluation (R&M).

Tools can be classified according to the notation that is used (see Section 4.3.2): graphic-based, language-based analysis. Of course there can be a mixture of both using schematics as well as specification languages.

4.4.1.2.1 Graphic-based Analysis. Tools using a graphical notation can be decided to be also method-oriented. The most widely used notations are based on Petri-Nets, Structured Analysis, System Development, Real-Time Modelling, and Conceptual Modelling (see Section 4.3.2.1).

Most of the professional tools got documentation processors according to standards, such as the DoD Standard 2167.

Some analysis features according to the different notations are listed thereafter.

Petri-Nets. Some code generators for other compilable source codes are available, such as Ada, C, and VHDL.

Structured Analysis. Some tools support consistency checks, code generation, and project management.

System Design. Tools support project management, design rule checking, and consistency checking.

Conceptual Modelling. Relationship normalisers are available as well as schema generators for relational, network, and hierarchical database management systems. The conceptual models can be checked by consistency. The diagrams can not be used as a stand-alone method since they do not provide any behavioural specifications.

Real-Time Modelling. Tools support project management, design rule checks, and timing analysis.

4.4.1.2.2 Language-based Analysis. Analysis by the use of high-level specification languages, such as Prolog, Smalltalk, and VHDL can only be managed if design environments are used.

To provide an efficient design environment the tools have to make use of a graphical user interface based on a window system.

Furthermore, the design environment has to provide some features, such as reliability analysis, timing analysis, statistical analysis, documentation processors, etc.

To ease the design a powerful library management has to be included.

The methodology analysing the specification has to be adopted to the tools and the design environment that are used.

4.4.1.3 Other Analysis Techniques

For mission-critical system development some special techniques are used to ensure reliability (see [51]), such as dual development, N-version programming, and N-fold Inspection.

N-fold inspection is shown to be the most practical approach. This is, the same URS is given to N independent teams to identify faults that might not be detected by only one team. Each team performs a formal inspection using a checklist and analyses the URS. After the faults are collected from the teams, they can be used to rework a new version of the URS.

4.4.2 Dimensioning

4.4.2.1 General

There should be a permanent iteration between the subtasks analysis and dimensioning as well as among the subactivities of the dimensioning itself.

Dimensioning is in particular some kind of optimisation and adjustment over design parameters, such as allocation of hardware, firmware, and software.

Up to this level of the specification process the behavioural requirement specifications are still independent of any technology. That is, no functionality has been allocated neither to hardware nor to software so far. A detailed Constraint Propagation as it is introduced in Section 3.4.1 has to take place considering non-technical parameters as well. Thus, not only technical parameters are to be reorganised, refined, and optimised.

Furthermore, the variable design parameters have to be fixed. That is, open decisions have to be fixed to complete the specification entirely. Related parameters can be improved and optimised according the new fixing of variable parameters.

In a last step the hardware and the software has to be checked by a performance analysis to improve the timing of the system that is to be defined.

Thus, this iteration cycle includes the following main activities:

- Preselection of Hardware and Software
- Software, Firmware, Hardware Partitioning
- Performance Analysis

4.4.2.2 Preselection of Hardware and Software

Before the partioning of the functionality into hardware or software can take place first some hardware and software has to be preselected according to some technical and/or commercial constraints. Considering all the *Qualitative* and *Quantitative* constraints mentioned in Section 3.4.1 a preselection of hardware and software has to be done.

According to a spreadsheet-like description (equations, inequalities, relations) a Constraint Propagation can be done (see also Section 2.6 and [20]). After checking the constraints in most cases the requirements gained during the previous specification process will lead to several possible valid hardware and software alternatives. The different alternative parts can be extracted out of a so-called *Acquisition Device*.

Optimisations can be done using linear or integer programming, for instance. This notation is also suitable for other techniques, such as dynamic programming.

By the use of a Knowledge Based Assistant considering the history of previous decisions and methodologies a particular software and hardware solution can be selected automatically.

Some of the hardware selections can not be decided on yet. They have to be postponed after the partitioning.

4.4.2.3 Software, Firmware, Hardware Partitioning

The partitioning has to be done regarding to the preselected hardware and software.

The partitioning can be classified by several subtasks (see [45]):

- Identify hardware functions
- Identify software functions
- Identify firmware functions
- Select hardware to host the software and firmware

The identification should be done considering already existing components as well as concerning the performance to be expected. Functions representing bottlenecks should be mapped to hardware or firmware, respectively if no constraints are hidden.

It should be carefully evaluated what would be more efficient, to use some already existing components or the new development of some ASICs or software/firmware components. The cost and the performance tradeoff is a paramount issue that the designer has to be aware of.

Identifying hardware functions they have to be mapped to physical components and their physical interconnections. A similar mapping can be adopted to firmware and software functions.

Identifying software function there first must be a decision on the implementation language. An approximate evaluation of size and the performance has to be done.

Though firmware is a different domain the identification of firmware functions are similar to the identification of software functions.

4.4.2.4 Performance Evaluation

The alternative that has been chosen in the previous step is usually checked by a performance evaluation. Furthermore, the decision between different alternatives is to be supported by a detailed performance analysis of the different alternatives.

A performance evaluation is usually made by a *stochastic discrete event simulation*. Complex systems and communications are typically modeled by the use of a queueing network. Several sources compete for different servers. Statistical parameters can be checked, such as arrival rate, service time, turnaround time, throughput, and utilisation.

Specification languages widely used in performance analysis are: SIMULA and Simscript.

4.4.3 Transformation

The procedure of decomposition fairly depends on the representation that was chosen to define a formalised specification. If the specification is already defined using an operational notation this kind of representation can be directly used as an input to the further design synthesis. If some parts of the system specification are still defined in a mathematical representation according to Section 4.3.1 it seems to be the best solution to transfer this representation into an operational notation first.

An axiomatic specification can hardly be decomposed. It has to be transformed to another representation.

[48] gives some guidelines to decompose a VDM specification by the means of a hierarchical decomposition.

However, it seems to be more practical to transform the I/O specification into an operational specification since it has to be transformed during the further design process anyway. Furthermore, the effort to verify the decomposed I/O specification might increase tremendously. Thus, each function first has to be transformed to its equivalent operational specification. The translation has still to be done manually by looking for a valid computation for each function.

4.5 Reports

The following reports can be delivered during this process (see also [46]).

- **Formalised Requirement Specification Documentation**
  This is a detailed report on the Formalised Requirement Specification.
  Status: Internal Engineering Report.

- **Internal Engineering Specification Documentation**
  This is a detailed report on the Internal Engineering Specification.
  Status: Internal Engineering Report.

- **Component Report**
  This is a report on the hardware, software, and firmware components that were identified during the preselection and partitioning.
  Status: Internal Engineering Report.

- **System Model Analysis Report**
  This is a report on the verification, validation, and analysis of the system specification .
  Status: Internal Engineering Report.

- **Project Plan Approval**
  This report covers refinement and improvements of the Project Plan.
  Status: External Management Report.

- **System Model Approval**
  This report covers refinement and improvements of the System Specification report.
  Status: Internal Engineering Report.

- **Environment Model Approval**
  This report covers refinement and improvements of the Environment Specification report delivered by the requirement to the system engineer.
  Status: Internal Engineering Report.

- **System Validation and Verification Plan Approval**
  This report covers refinement and improvements of the System Validation and Verification Plan.
  Status: Internal Engineering Report.

- **User Change Request**
  The changes which are required by the customer should be fixed in this report.
  Status: External Engineering Report.

- **Task Reports**
  This are intermediate documents containing the progress that is made during the second period.
  Status: Internal Engineering Reports.

- **Anomaly Reports**
  This are intermediate reports containing information about anomalies in specification, schedule, cost, procedures, and methods which are used.
  Status: Internal Engineering Reports.

Figure 5.1: From the FRS to the UES

The FRS has to be analysed, dimensioned, and transformed to a human-interpretable representation (see Figure 5.1). In some sense it is a reversal of the process mentioned in Chapter 3 but the transformation is done in a more formal way since some tools can be used, such as documentation processors.

Analysis and Dimensioning are the overlapping tasks with the process introduced in Chapter 4. The only difference to the process mentioned in that chapter is the transformation to a different notation.

There should be several iterations between the three different subtasks analysis, dimensioning, and transformation. Intensive feedbacks to former and simultaneous phases have to take place as well.

5.2 Actors

Customer External Customer
Supplier Requirement Engineer

The actors which are involved in this process are the Requirements Engineer and the External Customer, since documents as well as a contract approval is delivered to the External Customer.

All the considerations mentioned in Section 3.2 are still valid and can be applied to the above classification.

5.3 Data Representation

Machine interpretable data are handled during this process. Human-interpretable documents are to be produced.

The notations that are used for the Formalised Requirement Specification can be classified by three different notations: mathematical, graphic-oriented, and language-oriented notation.

For a detailed explanation see Section 4.3.

Some of the document representation already mentioned in Chapter 3 are handled to make some technical and contractual decisions visible to the external customer.

The documents that are delivered to the external are to be produced by some documentation processing systems, if possible.

The documents should be set up according a documentation standard. As already mention in Section 2.7 there are some standards defined by CALS covering textfiles, vector graphics, raster graphics, and the composition of technical manuals. These are SGML, CGM, IGES, GRP 4 Raster, and MIL-STD-1804.

5.4 Activities

Analysis and Dimensioning are the overlapping activities regarding to the process mentioned in Chapter 4.

5.4.1 Analysis

See Section 4.4.1.

5.4.2 Dimensioning

See Section 4.4.2.

5.4.3 Transformation

The improved and enhanced FRS is to be transformed to a human-interpretable representation applying more formal methods, so that the completed and checked specification can be approved by the External Customer. In some way it is a reversal of the transformation process mentioned in Chapter 3

No structural decomposition should take place during the transformation since the UES is set up to make the results gained from the analysis and dimensioning visible to the external customer only.

There are two approaches to the documentation production to set up the UES depending on the way how the documents are produced.

- Using a Documentation Production Systems
- Manually Documentation Production

Documentation Production Systems. It is most efficient if documents can be produced automatically. Unfortunately it is only possible if the preceding process is been managed by the use of tools that includes some documentation software as well. Thus, some adhoc notes and comments can be attached to the specification from the very first beginning. These notes are used by tools to provide any documentation processing. Documentation software is available providing Scripe[1] and PostScript using IEEE standards and military standards. Emerging documentation standards are defined by CALS (see Section 2.7). Recent tools provide technical documentation only. There is no tool known that supports any contractual aspects.

In most cases is not sufficient to pass the documentation that is produced by a system to the external customer without giving any further comments. Since tools produce documents referring to technical issues only they are to be enclosed in these documents that have to be set up manually. The way how it is managed in detail depends on the external customer and changes from project to project.

Manual Production. Setting up the UES manually the transformation process mentioned in Chapter 3 has to be logically reversed. The major difference between the URS - the input to the evaluation - and the UES - the output of the process - is that the UES is a complete, consistent, and partly optimised URS proposing hardware, software, and firmware to be used.

The reversal of the transformation mentioned in Chapter 3 can be done by two different transformations:
Formalised Specification → Structured Natural Language or

[1] Scripe is a trademark of UNILOGIC LTD..

Formalised Specification → Natural Language.
The graphical representation at higher hierarchical design levels can be directly used for the UES omitting some most detailed technical issues only.

5.5 Reports

The following reports can be delivered during this process (see also [46]).

- **System Contract Approval**
  This is a revision, improvement, and enhancement of the System Contract mentioned in Section 3.5.
  Status: External Report submitted to the external customer to agree with.

- **Formalised Requirement Specification Documentation**
  This is a detailed report on the Formalised Requirement Specification.
  Status: Internal Engineering Report.

- **System Model Analysis Report**
  This is a Report on the verification, validation, and analysis of the system specification .
  Status: Internal Engineering Report.

- **Project Plan Approval**
  This are refinements and improvements of the Project Plan.
  Status: External Management Report.

- **System Model Approval**
  This are refinements and improvements of the System Specification report.
  Status: Internal Engineering Report.

- **Environment Model Approval**
  This are refinements and improvements of the Environment Specification report.
  Status: Internal Engineering Report.

- **System Validation and Verification Plan Approval**
  This are refinements and improvements of the System Validation and Verification Plan.
  Status: Internal Engineering Report.

- **User Change Request**
  This are reports documenting changes that are required by the customer should be fixed in this report.
  Status: External Engineering Reports.

- **Task Reports**
  This are intermediate documents containing the progress that is made during the second period.
  Status: Internal Engineering Reports.

- **Anomaly Reports**
  This are intermediate reports containing information about anomalies in specification, schedule, cost, procedures, and methods which are used.
  Status: Internal Engineering Reports.

Chapter 6

Design Synthesis

6.1 Objectives

The main activity occurring in this task is to take the agreed Internal Engineering Specification (IES) and to derive from it a valid implementation that will fully satisfy the specification. This is the decomposition of the previously evaluated hardware and software specification at most. The design synthesis is a simultaneous process of several concurrent subactivities, such as design, implementation, testing, manufacturing, training, maintenance, etc. Software design and hardware design are inherently different processes. Software design and implementation can be seen as one activity. Whereas, hardware design and manufacturing is done successively.

It is necessary to check the compliance with the different constraints, such as reliability, safety, etc. Therefore, the testability and maintainability are to be designed into the system from the very first beginning (see also Chapter 2). This is, the inclusion of self tests, diagnostic tools, and other maintenance aids, for instance.

First of all already existing components can be installed as far as possible. Non existing hardware has to be designed, usually covering machine, board, and ASIC design with respect to the firmware to be developed. Beside the software that has to be developed a prototypical virtual machine can be implemented if the target hardware is not available yet. Then it would be necessary to install an emulation of the non-existing hardware first so that there will be no major bottleneck in the hard & software activities. The final goal of the design synthesis is to ensure that all the design aims have been achieved.

6.2 Actors

The entire design, implementation, and test activities are managed by system engineers using appropriate tools. System engineers can be classified by hardware and software engineers, respectively. Software engineers are usually design, implementation and test experts. Hardware engineers are usually design and test experts.

6.3 Data Representation

Data Representation used in this phase can classified by:

- Representation of Software
- Representation of Hardware

Software is typically described by the use of a programming language. In industry widely used languages are: Cobol, Fortran, Lisp, C, and the related languages. C++ seems to be the emerging standard of the 90's. The ANSI is going to define a standard based on C++ Version 2.0.

Hardware is described either by the use of a Hardware Description Language or by a schematic representation or by a mixture of both. Considering Hardware Description Languages there is an European and US agreement on VHDL. Considering physical representation IGES was set up for the exchange of graphics. Most of the available tools provide standard interfaces supporting PDES, EDIF, VHDL for data exchange via plain ASCII files.

6.4 Activities

6.4.1 General

Two further major design cycles can be identified during the design synthesis starting from the already existing specification:

- Software Design
- Hardware Design

Since the specification has already been partitioned into software, hardware, and firmware within the former dimensioning of the system the corresponding parts of the specification can easily be passed to the related design activity.

Software design and hardware design are inherently different processes. However, software design, implementation, and testing can be seen as one unique

activity, hardware design and manufacturing is applied successively. Hardware design is usually done by decomposition according to different levels of abstraction applying tests at each level to confirm the correctness of each transformation. Particularly at lower levels well established algorithms are known for Automatic Test Pattern Generation (ATPG). Test patterns are to be proved by a fault simulation computing the fault coverage. The design phase is to be completed by a final fault simulation at the lowest abstraction level that is visible to the designer. This has to be done to improve test patterns that can be applied after the manufacturing.

After the implementation the developed soft & hardware has to be composed to a complete system. The entire development phase is to be completed by several tests: hardware/software integration test, system functional test, acceptance test, and the installation test. The system functional test is to be done to check whether the implemented system fully satisfies the requirements.

The firmware design belongs to the hardware design domain using methods of low-level software design. Thus, firmware design is implicitly described by both of the following sections and is not outlined in an additional paragraph.

6.4.2 Software Design

6.4.2.1 General

Typically, software design, implementation, and testing is a permanent iteration between an editor, compiler, and debugger applying the procedure 'design a little bit, implement a little bit, test a little bit'. A valuable method applicable for object-oriented software design is proposed by Booch. In [12] he proposes an incremental, iterative process called 'round-trip gestalt design' applying several phases: analysis, design, evolution, and modification.

Software design in the context of system design fits into the following scenarios:

- The hardware that the software should be implemented on is not available yet.
- The software can be developed on already existing hardware.

In the first case a prototype of a virtual machine has to be installed first. This is necessary since the hardware design should be done simultaneously to the software design. For the implementation of the virtual machine the entire hardware specification available from the IES can be used for. This specification has to be transformed to a valid emulation of the hardware. It should be considered that in many cases it is a most time consuming task to make use of preexisting software tools (compilers, ...) running on an emulated machine, since in most cases emulations got a very poor performance.

6.4.2.2 Decomposition

Software design at most is a process of decomposition starting from the specification. Valuable definitions and guidelines can be found in [12, 32]. In literature the entire design process is simplified by considering a top-down approach only applying a stepwise refinement adding details incrementally. In practice a mixture of a top-down and a bottom-up design is used. Booch [12] called this method 'round-trip' design. It is also well known under the name 'Yo-Yo' design.

This is, a process starting from the specified 'what' ending with the detailed design describing 'how', from the abstract consideration to a concrete representation. This process has to be applied iteratively to get small, manageable modules with well-defined relationships.

The process of stepwise refinement was first introduced by Wirth [69] covering the following subactivities:

1. Decomposition of entities into subentities
2. Isolating non-interdependent properties
3. Adding details, postponing low level details as long as possible
4. Verification (formally and informally) that the decomposed design is a valid expansion

During the incremental refinement in most cases functional descriptions have to be replaced by operational description. Considering the time-space tradeoff the most efficient algorithm has to be allocated to the regarding function.

6.4.2.3 Levels of Abstraction

Decomposing the design can be done according to several levels of abstraction. One model was introduced by Dijkstra [23].

Level 5: Operator Services
Level 4: User Programs
Level 3: I/O Buffering
Level 2: Console Message Interpreter
Level 1: Memory Segment Controller
Level 0: Processor Allocation, Clock Interrupt Handling

Each level performs services used by the next upper level. Functions at higher levels can not be accessed from lower levels. They can be invoked from the next higher level only. Using this concept a high degree on information hiding and encapsulation can be reached.

6.4.2.4 Testing

Testing activities have to be done during and after the design & implementation activities. Testing can be classified according the hierarchical structure and soft-

ware development: Module/Unit Test, Module Integration Test, and Functional Test.

Module/Unit Test. The task of the Module/Unit test is to check the software modules against their interface or functional specification.

Module/Unit Integration Test. The task of the Module/Unit Integration test is to check the interoperability of the different modules.

Functional Test. The task of the functional test is to check the entire software that has been developed against its specification included in the IES.

6.4.3 Hardware Design

6.4.3.1 General

Hardware design is inherently different from the software design using a top-down approach. It is mainly based on the idea of an iterative decomposition transforming behavioural description into structure.

The domain of hardware design can approximately classified by Network, Machine (Mainframes, Workstations, etc.), PCB, and IC design.

Methods and tools that can be applied do not differ too much in network, machine, and PCB design. These three different domains are in most cases covered by the term 'system design'. Beside the system design domain IC design seems to be its own domain considering design environments and tools.

In current design practice tools or environments that handle the entire hardware design starting from a most abstract high-level specification delivering a tape or a disc that contains layout information for the manufacturing are not frequently used, if they are available anyway. In many cases a lot of transformations are still done without using any tools.

Typically, hardware design at higher levels is a permanent iteration between an editor and a simulator. There are many other tools that are applied during this process at lower levels.

6.4.3.2 Decomposition

In general the hardware specification delivered by the IES has to be transferred to physical implementation on the layout level. The output of this process usually is a tape and/or a floppy disc which can be passed to the fabrication using some layout exchange format, such as CIF (Caltech Intermediate Format).

The decomposition is an highly iterative process starting from a behavioural description, decomposing into some structural subparts, and assigning some geometry to the subparts. The first kind of decomposition is often called synthesis. During the second mapping several activities, such as partitioning, placement, and routing have to be applied.

6.4.3.3 Levels of Abstraction

In practice and in literature many models of abstractions are used. Gajsky's Y-chart is well accepted covering 3 different domains: behaviour, structure, and geometry. Rammig extended the model by the testing domain (see [54]). The following table gives an assignment of the specific entities to be handled to the structural and behavioural domain.

	Level	**Behavioural Entities**	**Structural Entities**
5	*System*	protocols, tasks, information flow	processing elements & their relations
4	*Algorithmic*	concurrent algorithms	modules, data structures
3	*Register-Transfer*	operations, bistrings	ROMs, PLAs, ALUs, registers
2	*Gate*	Boolean equations	netlists
1	*Switch*	discrete equations multiple valued logic	transistors & their connections
0	*Electrical*	differential equations	electrical parts & their connections

6.4.3.4 Testing

Two kinds of tests can be distinguished:

- verification to ensure a valid decomposition
- product test

Applying the first one the decomposition from level i to level i-1 has to be checked. This is necessary to ensure that the adding of details does not change the intended behaviour of the system. At higher levels there are functional tests applied using similar strategies known from the testing of software. At the lower levels test patterns are oriented towards structural properties.

The task of the product test is to sort out any faulty parts after the manufacturing process before the parts are delivered to the customer. The test patterns that have to be applied are delivered by the designer at the end of the decomposition. This test patterns have to guarantee an as high as possible fault coverage. The fault coverage in most cases is measured by using the single stuck-at fault model. The test patterns are improved by the use of a fault simulator at the end of the design process.

Chapter 7

Conclusion

This report tries to give an overview of the entire activities applied during the early phases of system development. It tries to cover all important languages, notations, and methods that are in practical use. Though this research has been conducted as complete as possible it has considered only information as far as it has been available. In most cases the internal dataflow inside companies is handled confidentially.

Considering the entire system specification and the high-level design process there is neither a commonly used method nor a commonly used language or representation yet. Companies use different languages and they apply their own inhouse methodology. It seems that the entire system development is not well automised yet and that a lot of transformations are still done manually.

This gap between theory and current practice can only be bridged by the use of complete design environments based on a common database such as it is provided by a CAD framework. Considering the internal representation as well as the data exchange interfaces there has to be a common decision on the standards to be used. Frameworks are based on the concept of complete data integration. Therefore there is a need for a standard conceptual modelling language. There seems to be a wide acceptance of the ISO/STEP standard EXPRESS since several modelling groups, such as CFI, IMG, and recently ECIP2 WP 1 have decided to use this language for their modelling activities.

In high-level design and specification there are two major flows that have been identified:

engineering flow

documentation flow.

Considering the documentation flow there seems to be an acceptance of the standards set up by the CALS groups (see Section 2.7).

Considering the engineering flow several subdomains can be identified according to this report:

- specification of structure
- specification of behaviour
- specification of constraints

Constraints can be subdivided in:

- behavioural constraints
- physical constraints
- commercial constraints

Considering the specification of structure, digital behaviour, and behavioural constraints (assertions) VHDL seems to be a standard that fulfills all this requirements. Though VHDL is on the way to be extended towards analog description it still has to be proven whether the extented VHDL allows the efficient specification of physical constraints.

VHDL is not suitable for the specification of commercial constraints and relationships such as it would be needed for a spreedsheet-like representation that can be used for *Constraint Propagation* in particular and for the internal representation of *Decision Support Systems* in general (see Section 2.6).

Since VHDL is designed to be a simulation language only VHDL misses some features for software engineering and rapid prototyping. But VHDL provides an interface to other languages:

> "A subprogram written in another language can be made available by defining its interface via a subprogram declaration within a package declaration that has no corresponding package body. The body of such a subprogram must be associated with the declaration of its interface in some implementation-dependent fashion." (IEEE Std 1076-1987 Page 2-9).

Concerning programming languages there is not only a trend but an acceptance of object-oriented languages. The emerging standard in the application domain of frameworks and operating systems seems to be C++ Version 2.0. There will be an ANSI standard available soon.

Evaluating all requirements of a specification language one should carefully distinguish between language and tool properties. Considering the requirements a language should be kept as simple as possible, so that the language is still human interpretable and supportable. A specification language should not provide too many syntactical constructs for the same concept to be expressed.

Adapting different notations to a language can better be done by the use of different front-ends rather than by the extension of the language. Furthermore, several

features can be provided by the use of back-ends. For instance, the result of a behavioural simulation can be used for further evaluation, such as performance analysis, result comparison, etc. Managing complex specifications tools have to provide an excellent user interface. Since the design of complex systems makes frequently use of already defined parts (Reusability!) a design tool should include a complete hard & software library as well.

At last there is a need detected for recommendations on design methods, such as the design flow to be followed, what tools are to be used, and how to apply the languages. Considering recommendations on the application of VHDL first research has been conducted within ECIP2 WP 2 covering System level and ASIC level design as well as design of synchronous systems.

This report considers high-level design and specification only. It does not give a complete overview of further design activities. This report stops where the other work within ECIP2 Workpackage 2 starts its enquiry, such as hardware design covering different levels of abstraction.

Bibliography

[1] R.J. Abbott, "Report on Teaching Ada," Technical Report SAI-81-313-WA, Science Applications, Inc. , McClean, Virginia, 1980.

[2] R.J. Abbott, "Program Design by Informal English Descriptions," Communications of the ACM, Vol. 26, No. 11, November 1983, pp. 882 - 894.

[3] R.J. Abbott, Unit and Integration Test Manual, NCC Publications, Manchester, England, 1988.

[4] W.W. Agresti, New Paradigms for Software Development, IEEE Computer Society Press/North Holland, Washington, DC, 1986.

[5] M.W. Alford, Software Requirements Engineering Methodology (SREM) at the Age of Four, Proc. COMSAC 80, IEEE Catalog No. 80 CH 1607-1.

[6] J.A. Anderson, J. McDonald, L. Holland, and E. Scranage, "Automated Object-Oriented Requirements Analysis and Design," Proceedings of the Sixth Washington Ada Symposium, June 26-29, 1989, pp. 265 - 272.

[7] S.J. Andriole, Software Validation, Verification, Testing and Documentation, Petrocelli Books, Inc., Princeton, New Jersey, 1986.

[8] S.C. Bailin, "An Object-Oriented Requirements Specification Method," Communications of the ACM, Vol. 32, No. 5, May 1989, pp. 608 - 623.

[9] N.D. Birrell and M.A. Ould, A Practical Handbook for Software Development, Cambridge University Press, Melbourne, Australia, 1985.

[10] B.W. Boehm, "A Spiral Model of Development and Enhancement," Software Engineering Notes, Vol. 11, No. 4, August, 1986.

[11] G. Booch, "Describing Software Design in Ada," SIGPLAN Notices, Vol. 16, No. 9, September 1981, pp. 42 - 47.

[12] G. Booch, "Object Oriented Design with Applications," The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.

[13] G. Booch, "Object Oriented Development," IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 211 - 221.

[14] P. Coad and E. Yourdon, OOA – Object-Oriented Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[15] P. Coad, "Object-Oriented Requirements Analysis (OORA): A Practitioner's Crib Sheet," Proceedings of Ada Expo 1988, Galaxy Productions, Frederick, Maryland, 1988, 9 pages.

[16] CALS, An Overview of the CALS Standards, CALS Policy Office, DASD(S)CALS, The Pentagon, Room 2B322, Washington, DC 20301-8000, April 1989.

[17] CALS, Military Standard - Automated Interchange of Technical Information, CALS Policy Office, DASD(S)CALS, The Pentagon, Room 2B322, Washington, DC 20301-8000, December 1988.

[18] B. Cohen, W.T. Harwood, M.I. Jackson, The Specification of Complex Systems, Addison-Wesley Publ., Massachusetts, Massachusetts, 1986.

[19] A. Colmerauer, "An Introduction to Prolog III", Communications of the ACM, Vol. 33, No. 7, July 1990, pp. 69 - 90.

[20] A.J. Czuchry, D.R. Harris, "KBRA: A New Paradigm for Requirements Engineering", IEEE Expert, Winter 1988, pp. 21 - 35.

[21] T. DeMarco, Structured Analysis and System Specification, Yourdon Press, New York, New York, 1979.

[22] V. Dhar, A. Croker, "Knowledge-Based Decision Support in Buisness: Issues and a Solution", IEEE Expert, Spring 1988, pp. 21 - 35.

[23] E.W. Dijkstra, "Structure of the 'THE'-Multiprogramming System," Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 341 - 346.

[24] ECIP Conceptual Model of Electronic Products, ECIP.PH.085.

[25] ECIP Dictionary of Electronic Product Terms, ECIP.PH.021.

[26] ECIP Conceptual Model of Product Design Process, ECIP.PH.086.

[27] A Guide to the ECIP Conceptual Model of Electronic Products, ECIP.PH.104.

[28] An Introduction to the ECIP Conceptual Modelling, ECIP.PH.022

[29] ECIP Conceptual Modelling Reference Manual, ECIP.PH.032.

[30] Specification of System Requirement Specification, ECIP2/SCTF/89.1179.

[31] S. Evanczuk, "Concurrent Engineering - The New Look of Design", HIGH PERFORMANCE SYSTEMS, April 1990, pp.16-27.

[32] R.E. Fairley, Software Engineering Concepts, McGraw-Hill, New York, New York, 1985.

[33] H. Falk, "CASE tools emerge to handle real-time systems," COMPUTER DESIGN, January 1988, pp.53-74.

[34] P. Freeman and A.I. Wasserman, Editors, Tutorial on Software Design Techniques, Third Edition, Catalog No.EHO161-0, Institute of Electrical and Electronic Engineers, New York, New York, 1980.

[35] P. Freeman and A.I. Wasserman, Editors, Tutorial on Software Design Techniques, Forth Edition, Catalog No. EHO205-5, IEEE Computer Society Press, Silver Spring, Maryland, 1983.

[36] P. Freeman, "A Perspective on Requirements Analysis and Specification," IBM Design '79 Symposium, 1979. Reprinted in [Freeman and Wasserman, 1980], pp. 86 - 96.

[37] C. Gane and T. Sarson, Structured Systems Analysis: Tools and Techniques, Improved System Technologies, New York, 1977.

[38] C. Gane and T. Sarson, Structured Systems Analysis: Tools and Techniques, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[39] H. Gomaa, "A Software Design Method for Real-Time Systems," Communications of the ACM, Vol. 27, No. 9, September 1984, pp. 938 - 949.

[40] L. Gray, "Transitioning from Structured Analysis to Object-Oriented Design," Proceedings of the Fifth Washington Ada Symposium, June 27 - 30, 1988, Association for Computing Machinery, New York, 1988, pp. 151 - 162.

[41] C. Green, et.al., "REPORT ON A KBSA," in Artificial Intelligence-Data Processing, Morgan Kaufman Publ., Inc., Los Altos, 1986.

[42] I. Greif, "Semantics of Communicating Parallel Processes," Ph.D. Thesis, Technical Report MAC TR-154, Laboratory for CS, MIT, Cambridge, 1975.

[43] D. Hall, "Automation - Road Maps for Concurrent Engineering", HIGH PERFORMANCE SYSTEMS, April 1990, pp.28-37.

[44] H. Hart, "Ada for Design: An Approach for Transitioning Industry Software Developers," Ada Letters, Vol. II, No. 1, July/August 1982, pp. 50 - 57.

[45] D.J. Hatley and I.A. Pirbhai, Strategies for Real-Time System Specification, Dorset House Publishing, New York, 1988.

[46] IEEE, Software Engineering Standards, Library of Congress Catalog Number 87-080801, IEEE, New York, 1987.

[47] M. Jackson, System Design, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[48] C.B. Jones, Systematic Software Development Using VDM, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[49] J.S. Kerner, "Should PDL/Ada Be Compilable?," Ada Letters, Vol. II, No. 2, September/October 1982, pp. 49 - 50.

[50] G.K. Khalsa, "Using Object Modeling to Transform Structured Analysis Into Object-Oriented Design," Proceedings of the Sixth Washington Ada Symposium, June 26-29, 1989, pp. 201 - 212.

[51] J. Martin and W.T. Tsai, "N-Fold Inspection: A Requirement Analysis Technique," Communications of the ACM, Vol. 33, No. 2, February 1990, pp. 225 - 232.

[52] M.W. Masters and M.J. Kuchinski, "Software Design Prototyping Using Ada," Ada Letters, Vol. II, No. 4, January/February 1983, pp. 68 - 75.

[53] A. Ould and C. Urwin, Testing in Software Development, Cambridge University Press, Melbourne, Australia, 1986.

[54] Franz J. Rammig, Systematischer Entwurf digitaler Systeme, B.G. Teubner, Stuttgart, 1989.

[55] D.T. Ross and K.E. Schoman, "Structured Analysis for Requirements Definition," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 6 - 15. Reprinted in [Freeman and Wasserman, 1983], pp. 86 - 95.

[56] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 16 - 34. Reprinted in [Freeman and Wasserman, 1983], pp. 96 - 114.

[57] B. Scherf, "Knowlege-Based Assistance for the Description of Computer Architecture", in Proceedings of the Ninth IFIP Symposium, ELSEVIER SCIENCE PUBL. B.V., Amsterdam, 1989.

Yourdon Press: Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[58] S. Shlaer and S.J. Mellor, Object-Oriented Systems Analysis: Modeling the World In Data, Yourdon Press: Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[59] M.K. Smith and S.R. Tockey, "An Integrated Approach to Software Requirements Definition Using Objects," Proceedings of Ada Expo 1988, Galaxy Productions, Frederick, Maryland, 1988, 21 pages.

[60] An Introduction to SADT: Structured Analysis and Design Technique, SofTech, Inc. , Waltham, Massachusetts, 1976.

[61] J.M. Spivey, The Z Notation: A Reference Manual, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[62] S.E. Stoecklin, E.J. Adams, and S. Smith, "Object-Oriented Analysis," Proceedings of the Fifth Washington Ada Symposium, June 27 - 30, 1988, Association for Computing Machinery, New York, New York, 1988, pp. 133 - 138.

[63] W. Suydam, "CASE makes strides toward automated software development," COMPUTER DESIGN, January 1987, pp.49-70.

[64] D. Teichroew and E.A. Hershey, PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems, ToSE, Vol. SE-3, No.1, Jan. 77, pp.41-48.

[65] University of Dortmund, HIT: An Introduction, Informatik IV, University of Dortmund, Dortmund, 1987.

[66] P.T. Ward and S.J. Mellor, Structured Development for Real-Time Systems, Volumes 1, 2 and 3, Yourdon Press, New York, New York, 1985.

[67] P.T. Ward, "How to Integrate Object Orientation with Structured Analysis and Design," IEEE Software, Vol. 6, No. 2, March 1989, pp. 74 - 82.

[68] P.H. Winston, Artificial Intelligence, Addison-Wesley Publ., Massachusetts, Massachusetts, 1984.

[69] N. Wirth, "Program Development by Stepwise Refinement," Communications of the ACM, Vol. 14, No. 4, April 1971. (Reprinted in Communications of the ACM, Vol. 26, No. 1, January 1983, pp. 70 - 73).

[70] E.N. Yourdon, Modern Structured Analysis, Yourdon Press: Prentice-Hall, Englewood Cliffs, New Jersey, 1989.