Summary of Case Studies in FOCUS - a Design Method for Distributed Systems[1]

Manfred Broy Max Fuchs Thomas F. Gritzner
Bernhard Schätz Katharina Spies Ketil Stølen

Institut für Informatik
Technische Universität München
Postfach 20 24 20, D-8000 München 2

Contents:

[1] This work is supported by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen"

1. FOCUS: A Design Methodology for Distributed Systems

FOCUS is a formal system development method based on descriptive and functional system models. Case studies are essential for research in development methods for system and software design. In the following a brief description of a selection of case studies is given that have been carried out applying FOCUS. FOCUS suggests to organize the development of systems in four phases

- requirement specification,
- design specification,
- abstract programming,
- concrete programming.

For all phases specific formalisms and methods are provided. These include verification techniques to prove refinement steps. In the following we shortly sketch the ideas of FOCUS.

A common concept of all phases is that the activities of the system are modelled by streams of actions or messages. During the requirement specification we concentrate on the behavioural description of the entire system, not talking about its internal structure. We use predicates on action streams (traces) to describe the history of a system. In a design specification we switch to input/output oriented component descriptions based on stream processing functions. In the next step an abstract program is derived. The main task is to achieve a constructive description based on algorithmic constructs; here functional programming techniques are used. A more efficient description is obtained on the concrete programming level where a procedural parallel program is given. Stepwise refinement is used for the transition from one phase to the next. In particular, program transformation is applied to derive a concrete (procedural) program from an abstract (functional) one. The essentials of FOCUS are explained in detail in the following publications:

M. Broy, Towards a Design Methodology for Distributed Systems, Working Material of the International Summer School on Constructive Methods in Computing Science, Marktoberdorf, Germany,1988

M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T.F. Gritzner, R. Weber: The Design of Distributed Systems - An Introduction to FOCUS. SFB-Bericht Nr. 342/2/92 A, January 1992.

In the following section we present a summary of the case studies on FOCUS.

2. The Case Studies

In the following summarizes for a couple of case studies are given. Each case study is shortly presented as follows:

(i) an informal problem description is given and it is indicated on which aspects of development specific emphasis is laid,

(ii) comments on the formal treatment of the example are given,

(iii) the respective paper(s) are cited.

2.1 Views of Queues

(i) This case study outlines a method to derive interactive and concurrent modules from algebraic specifications. FIFO queues are treated as an example. Several versions of interactive queue modules are derived from a data type specification of queues. These versions include distributed and non-distributed modules.

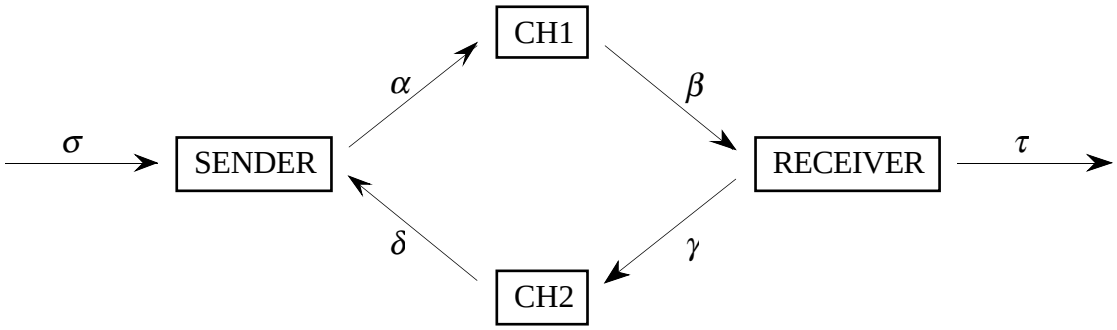
(ii) Technically the study starts by giving two data type specifications of queues. The first is a partial specification which does not consider exceptions (like reading from an empty queue), the second includes exception handling. It turns out that exception handling on the data type layer complicates the specification and moreover restricts the implementation freedom severely. Thus the partial specification is used to derive an interactive queue modul. Several versions are derived in a systematic way, ranging from quite simple to more robust variants. Technically speaking, an interactive queue is a stream processing function that accepts data from its environment, stores them in a FIFO order and releases them on request. The queue is realized as a sequence of autonomous cells, i.e. in a distributed way. The distributed version is proved correct w.r.t. the non-distributed version using fixpoint techniques.

(iii) [1.a] M. Broy: Views of Queues. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8704. (Also in: Science of Computer Programming 11, 1988, 65-86)

[1.b] W. Schütz. Die Implementierung von Bäumen durch interaktive Module und verteilte Systeme. Universität Passau, Diplomarbeit, 1988.

2.2. The Alternating Bit Protocol

(i) The alternating bit protocol is explained best using the following diagram:



A stream of data is transmitted from the sender to the receiver via channel CH1 such that the input stream (σ) equals the output stream (τ). The receiver gives some response (with respect to the received data) to the sender via CH2. CH1 and CH2 are both unreliable in the sense that they may lose data, but only a finite number of consecutive data items may be lost. To obtain $\sigma = \tau$ the sender attaches a bit (flag) to each data. For the first transmission, the sender attaches an "1" to the data and sends it repeatedly until it receives an "1" via CH2. Then it sends the next data attached with the alternative bit "0" repeatedly until the respond via CH2 is "0", and so on. The receiver simply takes the attached bit and returns it to the sender via channel CH2. Two case studies on the alternative bit protocol have been done, in both the emphasis is put on modelling timing aspects. In [2.a] the modelling uses a special message tick, whereas in [2.b] the time modelling is done using an abstraction of time, the so-called "input choice specification".

(ii) In [2.a] the streams α, β, γ, δ are described as the solution of a system of four mutually recursive stream equations. The behaviour of the unreliable channels (CH1 and CH2) is specified by introducing a scheduler for each channel; it determines whether a message is lost or not. The loss of data is modelled by transmitting a “tick” instead of a bit.

In [2.b] the Alternating Bit Protocol is specified by so-called "input choice specifications". These are specifications that relate a stream processing function f and a particular stream s with the intention, that f guarantees certain minimal output when applied to s. Thus, the Alternating Bit Protocol can be specified without explicit modelling of time. This special, relatively complex form of a specification becomes necessary to avoid monotonicity problems. This problems usually occur in our formalism when one abstracts from relevant informations like the timing information in the Alternating Bit Protocol. Further it is shown that input choice specifications can be composed with the known composition operators (sequential composition, parallel composition and feedback). Finally the correctness of the specification is proven.

(iii) [2.a] Th. Streicher: A Verification Method for Finite Dataflow Networks with Constraints Applied to the Verification of the Alternating Bit Protocol. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8706.

[2.b] M. Broy: Functional Specification of Time Sensitive Communicating Systems. Working Material of the International Summer School on Programming and Mathematical Methods, Marktoberdorf, Germany,1990. (Also in: M. Broy: Functional Specification of Time Sensitive Communicating Systems. LNCS 430, Springer Verlag, 1990, 153-179.)

2.3. A Railway System

(i) A railway track is considered to consist of sections. Each entry point of a section is guarded by a traffic light. The lights have to be managed in such a way that two trains are never positioned on the same or in adjacent sections. This case study covers the first two development levels, namely the requirement and the design specification.

(ii) First a modelling of a railway track topology as a parametrized algebraic specification is given. Based on this formalisation, the physically possible states and state transitions of the system are specified algebraically by Horn clauses. This leads to a predicate defining the traces of actions the railway system may perform. Next a predicate is derived that describes the safe traces, i. e. those where two trains are never allowed to be on the same or in adjacent sections of the track. The result is a requirement specification of a safe railway system. In the design specification a control module is constructed which sets the lights and switches of the railway system. For this purpose control actions, like setting a light red or green, are introduced. Technically the control module is specified as a stream processing function that takes a stream of actions from the railway system and a stream of controller actions and checks whether the controller actions are safe. Only safe control actions are forwarded to the system, the others are rejected. Thus no unsafe states can be reached in the sense it is described above. This way safeness is ensured is proved in the final part of that case study.

(iii) [3] M. Broy: Specification of a Railway System. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8715.

2.4. The Abracadabra Protocol

(i) The Abracadabra protocol controls the two way message transfer between two stations via an unreliable transmission medium.

In principle the case study works as the alternating bit protocol (see 2.2), however here the transmission is discontinued after a certain finite number of unsuccessful attempts. This case study mainly addresses algebraic specification techniques on the design specification level.

(ii) The protocol is specified using techniques of algebraic specification. The message transmission is modelled by stream processing functions. The proof that an appropriate composition of these functions yields a correct message transmission between the two stations is done using algebraic (= equational) reasoning only. However this reasoning leads to a long and tedious deduction, hence computer support seems inevitable.

(iii) [4.a] M. Broy: Some Algebraic and Functional Hocus Pocus with Abracadabra. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8717. (Also in: M. Broy: Some Algebraic and Functional Hocus Pocus with Abracadabra. Technique et Sciences Informatiques, 1991)

[4.b] E. Grieger. Verifikation eines Protokolls durch das RAP-System. Universität Passau, Diplomarbeit, 1988.

2.5. A Serializable Database Interface

(i) A serializable database interface managed by the two-phase locking scheme is considered. Since an easy understandable informal description is rather long and complicated, we refer to [5.a] for this purpose. This very complex case study covers the first three design levels.

(ii) The database as well as its environment (the client world) are represented as abstract data types. Using these basic specifications the possible streams of the system consisting of client requests and database responses are described by a predicate. This is the requirement specification. For a design specification the implementation scheme of the interface is specified as a stream processing function which maps a stream of client requests to a stream of responses of the database. Finally it is shown, how the Horn clauses of the specification of the response function, which controls the database interface, can be simply transformed to obtain a recursive declaration of this function in an applicative programming language.

(iii) [5.a] M. Broy: Algebraic and Functional Specification of a Serializable Database Interface. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8718.

[5.b] R. Bauer. Implementierung einer seriellen Datenbankschnittstelle ausgehend von einer formalen Spezifikation. Universität Passau, Diplomarbeit, 1989.

2.6. Shared Resource Arbitration

(i) This case study covers a problem in the domain of operating systems. A finite number of devices equipped with priorities for using a shared resource are given. Each device may request, start or stop the use of the resource. The sequence of these actions performed by the system has to fulfil some obvious conditions such as exclusive access to the shared resource

and some more subtle constraints concerning the priorities of the devices. In this case study the emphasis is placed on time modelling.

(ii) The requirement specification is given as a predicate describing the possible traces of the system. For the design specification the system is split into two stream processing functions. One (control) corresponds to a module which controls the access to the shared resource. The other (device) corresponds to the devices that try to access the shared resource. A proof is given, which shows that the composition of the two components fulfils the requirement specification. It turns out that the first attempt to specify the function "control" models the timing aspects of the system in a very restrictive way, it may even contradict the (informal) intended behaviour. Here the problem arises that a less restrictive specification of "control" would yield inconsistency between prefix monotonicity and liveness conditions. In a second attempt this difficulty is overcome by explicitly splitting the specification into a safety and a liveness part. The safety predicate describes (as before) a class of prefix-continous functions and hence the partial correct behaviours of the system. In contrast, the liveness predicate now is given as a relation between a function and an input (input choice specification). A third possibility to treat timing in the design specification is given by an explicit modelling of time. For this an action “tick” is introduced allowing a function to produce the action tick whenever the time proceeds and no actual output action is available.

(iii) [6] M. Broy, Th. Streicher: Specification and Design of Shared Resource Arbitration. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1987, MIP 8721. (to appear in Journal of Parallel Programming)

2.7. A Lift Controller

(i) A simple lift is specified which may perform the actions “visiting level i” and “being required on level i”, where i runs trough a finite set of levels including “top” and “bottom”. The lift has a simple strategy to serve the requests of clients:

- it visits level i only if there has been a request for this level which is not yet satisfied,
- it changes its moving direction from up to down or vice versa only if there is an open request in the new and no open request in the old direction,
- every request will eventually be served,
- in the very beginning the lift is at level “bottom”.

This case study addresses all four development phases of our design method.

(ii) This informal description is straightforwardly translated into a predicate defining the possible traces of the lift, which is the formal requirement specification. As in the Shared Resource Arbitration example (see 2.6) the design specification consists of two (sets of) stream processing functions. One models the moving elevator, the other a control device. Passenger requests as well as position reports, i.e. the level the elevator is currently at, are sent to the

controller. The controller takes this information and determines the next action of the elevator, which is sent back to the elevator. Combining both modules leads to a feedback loop. It is shown that the moves of the elevator managed by the controller produce a trace that satisfies the requirement specification. The development then proceeds by giving an abstract applicative program which is then refined into a concrete program. These programs can be constructed by translating the specification axioms into programming language constructs.

(iii) [7] M. Broy: An Example for the Design of Distributed Systems in a Formal Setting: The Lift Problem. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1988, MIP 8802.

2.8 A Gas Station

(i) A gas station with two pumps and three queues is described. The cars arriving at the station have to be served each at one of the two pumps according to a particular service discipline. The emphasis in this case study is put on the splitting of global requirements into local requirements for the program to be designed. This splitting is done according to the rely/guarantee paradigm.

(ii) The specification is done on the requirement and design specification level. Methodolocical aspects are stressed whereas verification is skipped. On the requirement specification the behavior of the whole system is given by a trace specification. To specify the gas station on the design level the system is split into components according to the rely/guarantee paradigm. Two different designs, namely a two component (environment and controller) and a three component design (environment, access-controller and pump-controller), are presented. The paper presents ideas how to facilitate the transition from a requirement specification (technically speaking a trace specification) to a design specification (technically speaking a functional specification). Further techniques are outlined to specify nondeterministic components (sets of stream processing functions) and to integrate 'states' in stream processing functions.

(iii) [8] R. Weber: Where can I get gas round here? - An Application of a Design Methodology for Distributed Systems. LNCS 490, Springer Verlag, 1991, 143-166.

2.9 The Dining Philosophers Problem

(i) The well known example of the dining philosophers is treated. The emphasis is placed on particular methodological aspects in the design of reactive systems, namely the systematic derivation of component specifications from an initial requirement specification that does not distinguish between the system to be implemented and a given environment. The system behaviour fulfils the following informal constraints:

– a philosopher performs the actions "getting hungry", "sitting down and starting to eat" and "getting up and starting to think" accurately in this order,

– neighbours are prevented to eat at the same time,

– every philosopher sitting at the table will leave it eventually,

– every hungry philosopher will eventually be allowed to eat.

(ii) An initial requirement specification which does not distinguish between the system and the environment is given in a trace oriented and a corresponding state oriented view. Based on the state oriented requirement specification a systematic derivation, with regard to safety and liveness requirements, of a butler and a system component is outlined. It turns out that one component can fulfil its tasks satisfactorily only when the other component behaves according to certain restrictions (rely-guarantee-paradigma). It is proved that the conjunction of the butler and the system component implies the initial specification. In a next step the butler component is described on the design specification level using state oriented stream processing functions. Finally the correctness of the butler specification on the design level w.r.t. to the requirement level is presented. A notion of correctness based on causal interleaving is used.

(iii) [9] F. Dederichs: System and Environment: The Philosophers revisited. Technische Berichte der Fakultät für Mathematik und Informatik, Universität München, 1990, TUM-I9040.

2.10 A Distributed Spanning Tree Algorithm

(i) The design of a distributed algorithm for computing a minimal distance spanning tree on a given fixed directed graph is carried out. The emphasis is placed on the systematic derivation of a distributed algorithm in a functional setting. The minimal distance spanning tree of a given graph with one of its nodes marked as root consists of paths of minimal length from the root to each node.

(ii) First the algorithm using the algebraic structure of sequences is specified. This allows to translate the minimal spanning tree problem into the problem of finding valid sets of messages yielding a verification criterion. The design specification is developed as a network of communicating entities that are divided into two groups: within the first group each node is assigned to an agent, whereas the second group consists of a single agent called *bus* provided for the delivery of messages. The verification, which is only sketched informally, is based on the notion of the valid set of messages, which is produced by the algorithm.

(iii) [10] M. Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm. Technische Berichte der Fakultät für Mathematik und Informatik, Universität München, TUM-I9046, SFB-Bericht Nr. 342/24/90 A, December 1990.

2.11 On Bounded Buffers

(i) In a way this study is similar to views of queues. However here the interest is focused on exception handling (robustness). Moreover compositionality plays a major role. A (bounded) buffer is presented as an interactive module. It consists of a finite number of cells. Besides the obvious behaviour caused by putting data into a non-full and taking data from a non-empty buffer the exceptional cases are specified as follows: If the buffer is empty the operation "take" will be passed to the output indicating that the buffer cannot fulfil the request. If the buffer is full the operation "put.data_in" produces the operation "put.data_out" at the output. Data_in is appended to the buffer whereas data_out (first element in the buffer) is taken from the buffer.

(ii) Several functional specifications of bounded buffers are given, i.e. the buffer is described as an interactive module. Starting with a simple version, finally a robust and composible bounded buffer specification is derived. Robustness covers the exceptional cases namely "putting something into a full buffer" and "taking anything from an empty buffer". Compositionality means that the (functional) composition of a n-bounded buffer and a m-bounded buffer yields a n+m-bounded buffer. Correctness proofs are mostly sketched in an informal manner (for the proof of compositionality the necessary induction technique is outlined). Based on the final buffer specification an algorithmic version of the buffer is eventually derived.

(iii) [11] M. Broy: On Bounded Buffers: Modularity, Robustness, and Reliability in Reactive Systems. In Beauty is our Business, Chapter 9, Pages 83-90, Edited by W. Feijen, Springer Verlag 1990 (Also in: M. Broy: On Bounded Buffers: Modularity, Robustness, and Reliability in Reactive Systems. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, 1989, MIP 8920).

2.12 The Connecting Switch

(i) In this paper the full development life-cycle of a design method for distributed systems is explained with the example of a connecting switch. A connecting switch is a system where stations may get connected, may then send actions to each other, and may finally get disconnected; it exhibits the behaviour of a very simple protocol.

(ii) The connection switch is considered under all the phases of the FOCUS methodology. First, in order to deal with definitions more uniformly, a notational framework similar to algebraic specification is proposed. Several techniques of building trace specifications are treated and applied to this example. Different forms of trace specifications of the example are verified. Two aspects of modifiability of trace specifications with respect to manipulations of the action set are dealt with: action refinement, i.e. actions are mapped into sets of sequences of actions of another action set in order to be "detailed", and action enrichment, i.e. further actions are added to the action set. Two forms of functional specifications are considered. The first one

is that of partial order processing functions, which is intended to express and exploit the causality of inputs and of outputs. The other form is that of stream processing functions. But the continuity requirement of stream processing functions is strong enough to establish the immediate transition from design specification to the first implementation level of abstract programs. Then, the two phases of implementation are treated: abstract programs written in functional style, concrete programs written in procedural style. The formal transition from abstract program to concrete program is sketched by giving examples for transformation rules.

(iii) [12] T.F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch. In: D. Hogrefe (ed.): Formale Beschreibungstechniken für verteilte Systeme. series: Informatik Aktuell, Springer-Verlag (1992) 144–176.

2.13 A Mail System

(i) The *Multiprocessor Multitasking Kernel* (MMK) developed at the Technische Universität München uses mailboxes as a means of communication between concurrent processes. In the paper [13], a functional state-oriented specification of the MMK's mailbox system is given. The overall aim is to present a formal description that can be used by a programmer as a reference to the MMK mailbox system. Therefore, special emphasis was put on “pragmatic” aspects like modularity, readability and notation. Some possible time models and state-oriented specification techniques are discussed.

(ii) The MMK's mailbox system is described as a functional agent, i.e., as a predicate on stream processing functions. In order to facilitate the specification, first a state space is introduced by axiomatic (algebraic) specification techniques. Based on this state space, some predicates on stream processing functions are defined, which specify the mail system's possible behaviours. These predicates are written in a structured way that resembles the description of a state-automaton. A special modelling of time is used, where time ticks serve as separators between blocks of simultaneous actions. Surprisingly, the explicit description of time also helps to simplify the specification by reducing nondeterminism.

(iii) [13] C. Dendorfer: Funktionale Modellierung eines Postsystems. SFB-Bericht Nr. 342/28/91 A, Technische Universität München, Oktober 1991.

2.14 The Stenning Protocol

(i) A simple *communication protocol*, the *Stenning-protocol*, is developed using the technique of stepwise refinement. The protocol deals with the same problem as the alternating bit protocol (cf. section 2.2), i.e., providing reliable communication over an unreliable medium. However, here a different mechanism based on integer valued sequence numbers is

used instead of just bits. The case study covers all development phases of FOCUS and includes detailed proofs of all design steps.

(ii) The protocol development starts from a trace specification of the service to be provided by an upper layer and of the service of the lower layer, on which we rely. This is considered the requirements specification of the protocol entities. In several design steps on the trace level, interface specifications of the protocol entities are derived. These are then schematically transformed into predicates on stream processing functions. Several further design decisions are incorporated into this specification until finally an AL program and a PL-program are achieved. This case study reflects the current state of FOCUS as documented in the introductory paper mentioned in section 1.

(iii) [14 a] C. Dendorfer, R. Weber: From Service Specification to Protocol Entity Implementation - An Exercise in FOCUS. SFB-Bericht Nr. 342/4/92 A, Technische Universität München, January 1992.

[14 b] C. Dendorfer, R. Weber: From Service Specification to Protocol Entity Implementation - An Exercise in Formal Protocol Development, In Protocol, Specification, Testing and Verification XII, R. Linn and M. Uyar: editors, IFIP Transactions C-8, 1992.

2.15 Three Communication Processors

(i) Three communication processors are specified using a relation based formalism. A refinement calculus is formulated and employed to reason about these specification. In particular, it is shown how this calculus can be used to decompose specifications into networks of specifications.

(ii) First, a refinement calculus based on a relational apporach of is introduced. Within this formal framework three communication processors are formulated. The presented calculus is used to reason about these three communication processors. The case study deals with processors that have send- and receive-links. The first processor performs purely asynchronous send- and receive-communications, whereas the second processor employs a synchronization protocol. Contrary to the first two processors, the third processor is constrained to have only a bounded memory. For every processor the following question is investigated: can two communication processors of this kind be combined to obtain another communication processor of the same kind? The proposed refinement rules are used to answer this question.

(iii) [15.a] K. Stølen, T. F. Gritzner: Using a Relation Based Formalism to Specify and Reason about Three Communication Processors. Technische Universität München, unpublished.

[15.b] M. Broy, K. Stølen: Specification and Design of Finite Dataflow Networks – A Relational Approach. In the Proceedings of Formal Techniques in Real Time and Fault Tolerant Systems, Working Group ProCoS, Germany, 1994.

2.16 Functional Specification of a Tempomat

(i) In this case study the user interface as well as the speeding switch-off of a tempomat are specified on the design specification level. In general a tempomat is used to keep the speed of a car constant. The speeding switch-off turns off the tempomat whenever the speed exceeds the required level -- this could be the case if the car goes downhill. This work is carried out as a feasibility study in the area of car electronics. The main emphasis is put on time handling.

(ii) A functional design specification is formulated. To handle real time requirements, timed streams are used. A time slot without any proper action is represented by a special symbol called time-out. To improve the readability of functional specifications a tabular form is introduced.

(iii) [1] M. Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung. SFB-Bericht 342/1/93 B, Technische Universität München, 1993.

2.17 A Min/Max Component

(i) In this case-study a so-called Min/Max component is specified and formally developed. The Min/Max component has two input channels and two output channels. The values on the two output channels correspond to the minimum and the maximum values received along the input channels so far. There are no constraints on the order in which the component switches from processing inputs received on the two input channels. However, it is required that all input messages eventually are read and processed. The main emphasis of this work is to investigate the suitability of refinement rules given for a relational specification style in Focus.

(ii) The specification of the Min/Max component is refined in a stepwise fashion into a large network of subcomponents communicating via channels of type Bit. The specifications are given in a relational style [2] and appropriate refinement rules are given. Finally, in the style of [3], it is outlined how this network can be translated into SDL.

(iii) [1] Max Fuchs, Ketil Stoelen: Development of a Distributed Min/Max Component. SFB-Bericht 324/18/93 A, Technische Universität München, 1993.

[2] Manfred Broy, Ketil Stoelen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach. SFB-Bericht 342/7/94 A, Technische Universität München, 1994.

[3] Eckhardt Holz, Ketil Stoelen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus. SFB-Bericht 342/11/94 A, Technische Universität München, 1994.

2.18 Specification of a Production Cell with FOCUS

(i) A functional specification of a production cell in an assumption/commitment style has been developed in this case study.The production cell roughly works as follows: A feed belt transports a metal blank to an elevatory rotary table, which moves up the blank in an upper slightly rotated position. Being in this position, the blank is taken by a robot and put into a press. After pressing, the robot puts the forged blank on a deposit belt. The main emphasis of this work is to investigate the suitability of refinement rules for the stepwise refinement of distributed systems in an assumption/commitment style.

(ii) The formal treatment of the production cell is carried out at the design specification level. Based on a so-called master/slave rule a first abstract specification of the entire production cell is refined in a stepwise manner into a net of executable specifications. All important design steps are proven correct. Finally two implementations of the executable specification are given, namely a program written in concurrent ML and a SDL specification based on SDL processes, which correspond to the individual component specifications in Focus.

(iii) [1] J. Philipps: Spezifikation einer Fertigungszelle - eine Fallstudie in Focus. Diplomarbeit. Technische Universität München, 1993.

[2] J. Phillipps: Specification of a Production Cell with FOCUS. In GRONICS-94, Proceedings of the 1994 Groningen Student Conference on Computer Science, CS-N 9401. H. Groenboom and H. Klijn Hesselink and M. Lankhorst: editors. University of Groningen, 1994.

[3] M. Fuchs and J. Philipps: Formal Development of a Production Cell in Focus - A Case Study. In Case Study Production Cell, ISSN 0944-3037. C.Lewerenz and T. Lindner: editiors. Universität Karlsruhe, Forschungszentrum Informatik, 1994.

2.19 Functional Specification of a Communication Protocol

(i) The aim of this case study is to develop the specification of a simple communication protocol on the design level.

The communication protocol describes the data transfer in a system consisting of receiver, consumer and medium. The permission to transfer data from the consumer to the receiver is given to the medium by the receiver. Equally, the receiver can forbid this data transfer. The medium is specified as a network of two agents which are composed sequentially and by feedback.

This case study was carried out as a feasibility study to add various specification formalisms like tables and graphical representations to the normally used timed and state-based stream processing functions. In this way it may be possible to raise the readability and understanding of the functional specifications in Focus.

(ii) The specification of the medium as a network of two agents is given as a functional specification on the design level using an explicit state-space. Each of this agents is completely specified with special tables which characterize the relevant behavior of each special state. The actual readable inputs, the outputs as the reactions of these inputs and the resulted states are the entries of the columns in the tables. A set of equations of stream processing functions forms the semantics of each table.

Graphs of finite automata form the graphical representation of the behavior of each agent. These graphs can be derived easily from the tables. This representation of the agents is used to specify the states as predicates over communication histories.

(iii) [1] K. Spies: Funktionale Spezifikation eines Kommunikationsprotokolls. SFB-Bericht 342/8/94 A, Technische Universität München, 1994.

2.20 The Sieve of Eratosthenes Case Study

(i) First a graphical notation and the corresponding denotational semantics for objects, classes and associations are presented. The denotational semantics is based on streams and stream processing functions. The textual representation uses the language Spectrum [1]. Then the feasibility of the proposed notation is studied based on an example: the sieve of Eratosthenes. Its requirement specification and two concurrent implementations are given.

(ii) A formalism is presented in which objects are modeled as stream processing functions. The input streams contain request messages and the output streams contain the answers. Classes are modeled as parameterized object definitions. Each instantiation of a class on actual parameters generates a new object. Object configurations are modeled as Kahn networks. Based on this modelling a requirement specification for the sieve of Eratosthenes is given. Then this specification is refined into two implementations. The first one uses a list data structure and avoids object sharing. The second one uses an array-like data structure with sharing. Both implementations are then shown to be correct with respect to the requirement specification.

[1] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, K. Stoelen: The Requirement and Design Specification Language Spectrum - An Informal Introduction. TUM-I19140, Technische Universität München, 1991.

[2] R. Grosu: A Formal Foundation of the OO-Methodology Based on Stream Processing Functions - The Sieve of Eratosthenes Case Study. Technische Universität München, unpublished.

2.21 A Functional Specification of the Alpha AXP Shared Memory Model

(i) The case study is concerned with the modeling and analysis of the Alpha AXP architecture - a RISC architecture with multiple instruction and multiple data streams - and the Alpha Shared Memory in particular.

Special emphasis is put on a modular specification by providing separate descriptions of the processors and the memory. The model is used to analyze several properties of the architecture, to define different notions of processor optimization and to give a precise definition of the correctness of such an optimized behavior.

(ii) First, a modular description of the system architecture is given by relating a set of continuous stream processing functions to each component. The behavior of each component is described by a predicate on these functions, thus leaving room for non-determinism in form of underspecification. The behavior of the system follows from the behavior of the components.

In the second part the model is used to analyze some of the architecture's properties. At first, causal loops, which are admitted by the informal specifications, are shown to be excluded in the given description. Then a notion of observational equivalence of processor behaviors is introduced. This allows a formal definition of the concepts of correct look-ahead and short-cut optimization. Based on those concepts the robustness of the Alpha AXP architecture concerning different scheduling strategies of memory requests can be shown. Finally, an extension of the model to a more sophisticated memory locking scheme is given and applied to prove properties of a program synchronization protocol running on the Alpha AXP architecture.

(iii) [1] M. Broy: A Functional Specification of the Alpha AXP Shared Memory Model, Technische Universität München, unpublished.

[2] R.L. Sites (ed.): Alpha Architecture Reference Manual. DIGITAL Press 1992.

Acknowledgement

We are grateful to Thomas Belzner, Frank Dederichs, Claus Dendorfer and Rainer Weber who have written preliminary versions of this report.