Guidelines for the QMIPS-Workshop FORMALISMS, PRINCIPLES AND STATE–OF–THE–ART

University of Erlangen-Nürnberg, March 17 – 19, 1993

Planning the QMIPS-project we committed ourselves to survey, classify and compare the fundamental properties of all candidate methods for the performance modelling of parallel systems. The purpose of our workshop is therefore to present and discuss facts, expert opinions and different views about both established work as well as new formalisms, in particular

- □ queueing models
- □ stochastic graph models
- □ stochastic Petri Net models
- □ process algebras
- □ temporal logic.

The aim is to summarize, classify, and publish our knowledge and experience on

- □ the basic ideas of different modelling concepts,
- □ the various technical aspects,
- □ the methodologies for modelling parallel systems, and
- □ the relationship between the various formalisms

in a state-of-the-art report scheduled as the first deliverable of work-package one. Publication within some book series is planned.
In order to faciliate an attractive, structured presentation of our final report, authors are requested to organize their talks and transparencies along the following guideline:

1. **Basic Ideas and Concepts**
   - Definitions
   - Characterization of the basic modelling elements and assumptions
   - Underlying mathematical and logical concepts
   - Results, measures of performance
   - Specific features
   - Relevant literature

2. **Technical Aspects**
   - Model-subclasses with particular properties
   - Standard evaluation techniques
   - Analysis and composition of complex models
   - Customer classes and priorities
   - Relevant literature

3. **Methodical Steps in Modelling Parallel Systems**
   - Level of observation and modelling the parallel system's behavior: degree of abstraction, granularity, functional, temporal, topological, spatial aspects.
   - Phases of the modelling process and their interdependences.
     At Sophia Antipolis everybody agreed on the particular value of sketching this process by the same examples. Considering the different ideas, we present our "mandatory exercise" model in appendix 1 and "volontary exercise" models in appendix 2. Note that we do not expect complete solutions! Just give an idea how to apply the method (or methods) you are considering.
   - Experience with the modelling of large systems
   - Tool support
   - Relevant literature

4. **Relationship to Other Modelling Methods**
   - Analogies, correspondences and equivalences
   - Transformation techniques
   - Complementation and supplementation
   - Combination and interface procedures
   - Relevant literature

5. **Attempt of Summarizing Evaluation**
   Bearing in mind that we want to contribute to the foundation of a parallel systems modelling methodology, we should try to value the different formalisms, in order to discuss their advantages and disadvantages, their strengths and deficiencies. In appendix 3 we propose a list of the major characteristics, including
   - General characteristics
   - Expressiveness
   - Mathematical tractability
   - Hierarchical modelling and constructive design
   - Expenditure
   - Clearness and understandability.

As discussed in Sophia Antipolis, each group is expected to report at least on one formalism; joint presentations (and papers) are welcome. Additional presentations on special topics and ongoing research, as well as proposals for cooperative research activities are also welcome.

Appendix 1:
The Modularly Composed Multiprocessor Example (mandatory)

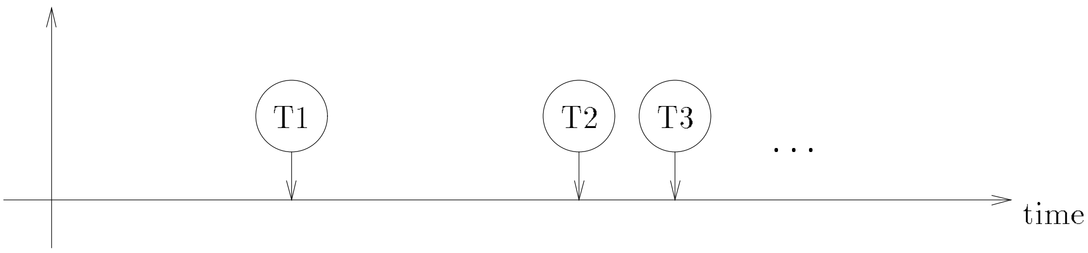
Remark: Remember that we agreed to support the presentation of each method by the same simple example(s). Studying carefully all proposals we got from you, we realized that they are by no means simple enough. All, however, contain interesting aspects.
In order to keep this mandatory example simple, demanding and flexible at the same time, we allow several variations for

- the workload model,
- the multiprocessor type,
- the operating mode, and
- the communication network.

The specific advantages of each method may be demonstrated by additional examples (see appendix 2).

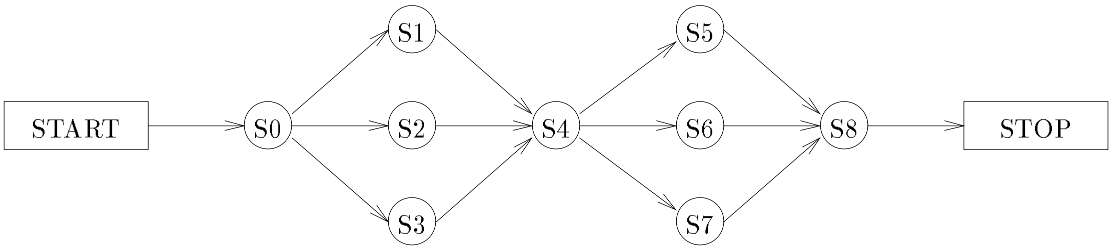
1.1 The Workload Model

a) The workload consists of a sequence of independant tasks arriving according to a Poisson process:



The required service time for the individual tasks may be either exponentially, constant or generally distributed.

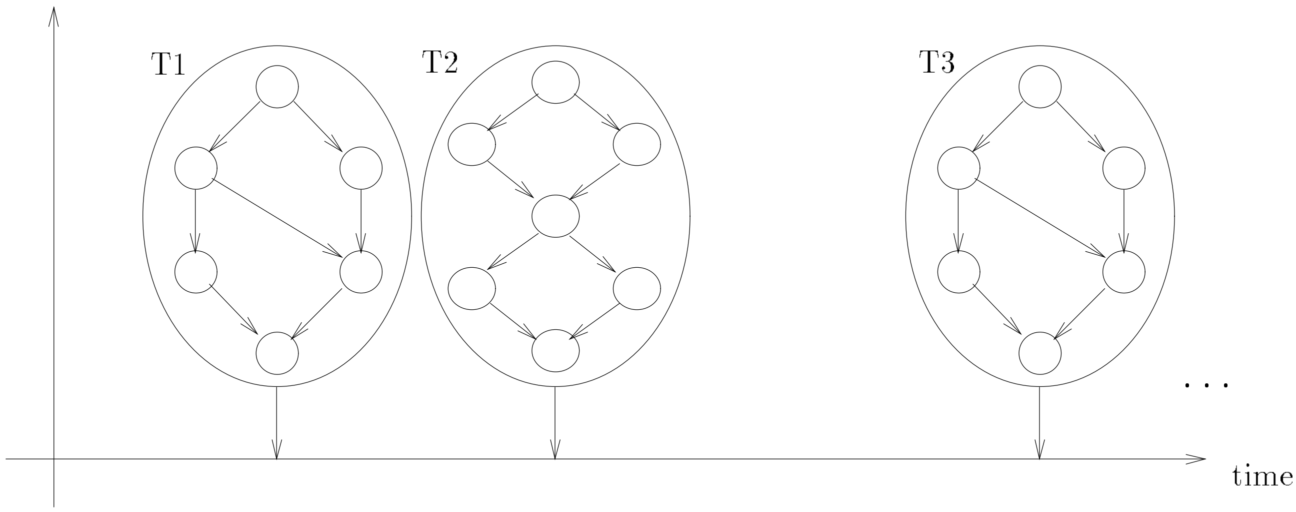
b) The workload consists of a single program (e.g. a multigrid algorithm) with sequential and parallel subtasks:



Again the required service times for each task may be exponentially, constant or generally distributed.

c) The workload consists of a sequence of programs of type b) with exponentially distributed interarrival times.

d) The structure of each program is represented by an acyclic graph, the type of which is randomly chosen in advance from some class of graphs, e.g.:



1.2 The Multiprocessor Type

a) There are two identical processors with local memories

b) There are N identical processors with local memories

c) Additionally there is a single global memory accessible by all processors

1.3 Operating Mode

a)
- Monoprogramming, i.e each processor completes a single program (or task) before processing the next one.
- Static mapping, i.e. programs and/or tasks are allocated in advance
- Scheduling: FIFO
- No breakdowns

b)
- Multiprogramming, i.e. tasks of different programs may be intermixed
- Static mapping
- Scheduling: FIFO, Processor sharing or priorities
- No breakdowns

c)
- Fault tolerance, i.e. breakdowns are allowed
- Fault removal, i.e. breakdowns and repair.

1.4 Communication Network

a) Communication costs are negligible

b) There is some fixed communication overhead

c) The communication subsystem is explicitely modelled and included into the analysis: A single bus, ring, grid, hypercube, crossbar switch or multistage network.

Start with the a)a)a)a) combination and try to advance as far as possible.
Note again: We do not expect complete solutions!
Just give an idea how to apply the method(s).

Performance values: Finally, please give an idea how to compute standard performance values (like throughput, program execution times, etc. ...) using your model(s).

Appendix 2: Advanced Examples

Just try to investigate one or several of these examples

- INR 1	Routing, Load Balancing
- INR 2	Distributed Signal Processing
- INR 3	Multiprogrammed Parallel Systems
- UER	Communication Network with Send-and-Wait
- UNE	Repair, Parallel Architecture
- UTO 1	MP with Cache Memories
- UTO 2	MIMD
- UTO 3	A Parallel Program Example

INR 1
A Problem of Routing and Load-Balancing
in Transputers-Style Networks

Meval team
I.N.R.I.A. Sophia Antipolis

1 Framework

We consider a parallel program made of T tasks, to be executed on a network of P identical processors.
Processors connections are fixed, and messages form one processor to another follow a fixed route.
On each processor are placed:

- a subset of the T tasks.
- a routing process taking care of:
  - accepting messages emitted by local tasks
  - local distribution of incoming messages
  - forwarding messages in transit

The tasks and the routing process share local resources (CPU), with or without a specific scheduling policy (fair processor sharing, priority mechanism, etc...)
Task M_i requires D_i units of processing time, where D_is are i.i.d. random variables.
Unit length messages are sent from M_i to M_j, following a Poisson process of rate $\lambda_{i,j} s_i(t)$, where $s_i(t)$ is the fraction of processor resources allocated to task M_i at time t.
All messages sent to a specific node P_j are put in a local queue and processed in FIFO order.

2 Performance Issues

We are essentially interested in program completion times.
Subproblems of interest include:

- queue lengths
- finite capacity queues
- alternative routes mechanism
- adaptive routing

INR 2
A Real-Time Problem in a Distributed System

Meval team
I.N.R.I.A. Sophia-Antipolis

1 Framework

This example is motivated by applications in distributed signal processing or distributed control of robots or plants.

The system is formed of a number of stations (captors or processing units) connected by a network. The stations send streams of messages to each other. It is assumed that arrival process of messages to the stations is the superposition of deterministic processes. It is further assumed that when a message of a certain stream is not completely transmitted when the next message of the same stream arrives for transmission, it is lost. When a message arrives at destination, it is removed from the system. Reception and transmission are performed independently.

For instance, station 1 may have to transmit n_1 different message streams. For $1 \leq j \leq n_1$, stream $S_{1,j}$ is composed of message of (fixed) length $L_{1,j}$ with destination $D_{1,j}$, arriving every $R_{i,j}$ units of time.

2 Performance Issues

The performance measure of interest in such systems is the frequency of loss of messages. The problem is to compute, estimate or bound this frequency for various network configurations:

- the network is a shared bus; different priority levels are assigned to the stations or to the message streams;
- the network is a slotted ring: slots of small size pass periodically before the stations. If a slot is empty, the station may put a piece of message in it. The slot is freed when the information it contains arrives at destination.

The choice of the queueing discipline is essential in such a system. To simplify, one may limit oneself to the FIFO discipline and non-preemptive priorities.

INR 3
Performance Characterization of Multiprogrammed Parallel Systems

Meval team
I.N.R.I.A. Sophia-Antipolis

1 Framework

This example is motivated by applications in multiprogrammed parallel systems.

The system consists of several heterogeneous processors with local memories (we are thus considering distributed memory parallel systems). These processors are connected by a communication network (which can be a ring, a grid, a hypercube, or simply a bus), and communicate with each other using messages.
The applications running in the system are parallel programs represented by acyclic task graphs. These task graphs can be statically generated with random structure or dynamically generated with a given stochastic distribution. A task is enabled when it receives all the messages from its predecessors. When a task finishes, it sends messages to all its successors.
The system is multiprogrammed and multitasked so that all enabled tasks may run simultaneously. The task assignments may be defined either statically or dynamically. On the same processor, the enabled tasks can either be executed in the time sharing way, or according to some specific scheduling policy which defines a dynamic priority list of the enabled tasks (for instance, first come first serve).

2 Performance Issues

The performance measures of interest in such systems are the response times of the parallel programs (the user view) and the throughput in term of programs (the system view). It will also be useful (from both qualitative and quantitative view) to analyze the communication delays of the messages, the synchronization overhead of the tasks, the utilization of the processors and the communications channels.

more precisions

As for our third example, here are some precisions:

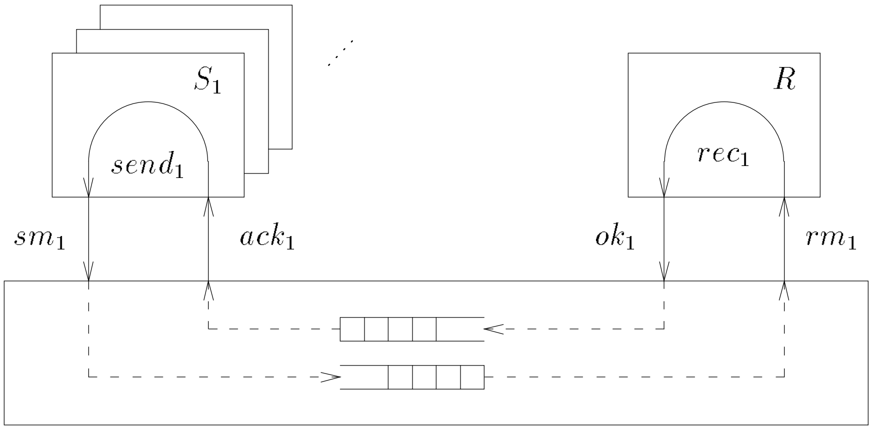
(1) The workload is a sequence of task graphs. Each task graph is randomly generated before its execution starts (hence "statically") according to some predefined random distribution on some class of graphs. Usual models generate first a random number of nodes (say, uniform, Poisson, Binomial or -even!- deterministic), then generate the dependencies (i.e. the edges) with a Bernoulli distribution: a dependency exist between $i < j$ with some probability p (or p_i).

(2) Here, the structure generates itself during its execution (hence "dynamic"). When a task is completed, it generates some subgraph with a predefined distribution (as above) which replaces it. Each task of this subgraph may in turn generate other sub-structures. Of course, the subgraph generated may be empty so that the process can terminate.

UER
A Basic Communication Network with Send-and-Wait Protocol

UER-group

We consider a network with generalized (compute or file) service: Several client stations (sender) require service from a single central server (receiver) via the communication network (medium):



Each sender S_i generates internally service requests by the $send_i$ action. If the medium is available, a message is transmitted (action sm_i — *send message* i). After a successful transmission and acceptance the file server performs some internal activities (action rec_i — *receive*). Thereafter, a return message (action ok_i) is sent back to the sending station. Having received the return message (action ack_i) a new service request can be initiated. Of course, requests of several senders may interleave and messages may wait for the medium or the receiver to become available.
Instead of a network with separate forward and backward channels, we may also use as medium Ethernet, Token Ring or Token Bus.

UNE
Parallel Architecture with Breakdowns

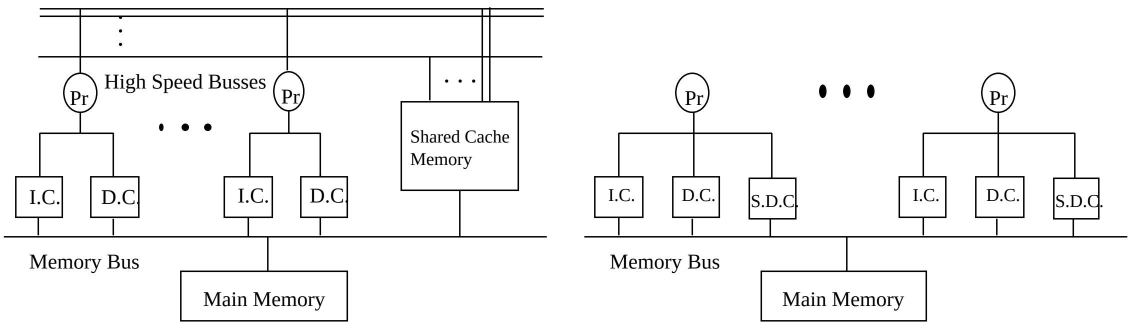
I. Mitrani

Here is an example of a parallel architecture for the purposes of the testbed:
A single unbounded queue is served by N parallel servers, each of which may be operative or inoperative at any given time. Breakdowns and/or repairs may occurr simultaneously in several, or all servers, and may also be triggered by job arrivals and/or departures. The problem is to determine the joint distribution of the number of operative processors and the number of jobs in the system.

Modelling and Performance Analysis of a Multiprocessor Architecture with Cache Memories

UTO-group

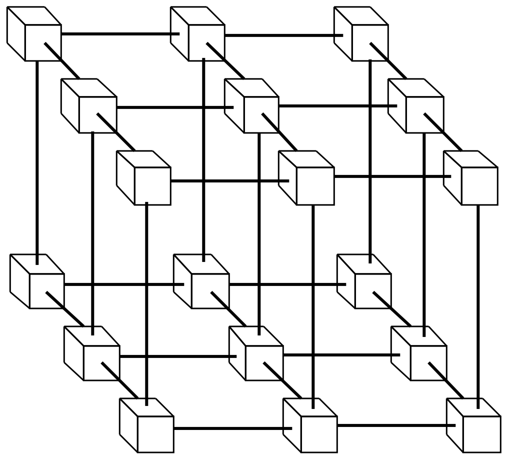
The parallel architecture contains P processors. Each processor of the multiprocessor system is provided with two cache memories for Instructions and Data that are not shared with the other processors. Shared data are stored in a single global cache memory, connected to the processors with a multiple-bus high-speed interconnection networks (B is the number of such busses). In order to support the many requests for shared data, the shared cache memory is assumed to be a multiport, interleaved memory.
A main memory is connected to the caches via a single main memory bus. Accesses to the main memory take place only when requests for instructions, data, and shared data are not satisfied within the caches. The locality properties of data and programs insure that such memory misses occur only in less than 10% of the references.
Processors execute for random amounts of time of average lenghts Z before issueing memory requests. Π represents the probability of accessing instructions, while $(1 - \Pi)$ is the probability of accessing data. The degree of data sharing is assumed to be S and R is the probability that a memory access is a read (obviously, $1 - R$ represents the probability of a write).
Finally, H is the probability of a cache hit. Cache access times last in the average 1 time unit. When a cache miss occurrs, the cache controller selects one cache block to be replaced to make room for the incoming block. The policy for this replacement could be random. A block (private or shared) must be written back in main memory when it has been modified in any cache. Assume that R represent also the probability of replacing non-modified blocks: with probability R a memory access (that keeps busy both the cache and the main memory) lasts K time units; with probability $(1 - R)$ $2K$ time units. The



performance of this architecture can be compared with that of a distributed shared cache multiprocessor in which each node is equipped with three cache memories.
The cache coherence protocol, that is needed in this last case, assumes a write-through organization (the main memory is considered to be 4 times slower thatn a cache) that, by broadcasting the new value over the bus, obtains the updating of the individual caches at the cost of blocking all of them during the write operation.

UTO 2
Modelling and Evaluation of Task Interaction MIMD in Distributed Memory Architectures

UTO-group



```
Computation task:
  while TRUE do
      local_computation;
      neighbour:= select(random);
      send_request(neighbour);
      receive(neighbour);
 od

Server task:
  while TRUE do
      receive(neighbour);
      compute;
      send_answer(neighbour);
   od;
```

(a) (b)

Figure 1: (a) 3x3x2 mesh (b) workload description

Let us consider MIMD distributed-memory architectures with n-dimensional mesh interconnection structure and possibly having a different number of processors in each dimension. Let us assume that the workload is evenly distributed across all the processing elements and the computations assigned to each processor entail interaction with the local neighbours only. The computation assigned to each processor is a set of $N \geq 1$ concurrent tasks performing a cyclic activity and one server task. The computation activity includes a local operation phase (whose duration is a stochastic variable exponentially distributed with parameter λ), followed by a request for service addressed with equal probability to one of the neighbours. Service time is a stochastic variable exponentially distributed with parameter μ. Furthermore let us consider two interaction policies:

- *preemptive interaction*: requests coming from the neighbours are served with higher priority than the local activity, preempting any computation task;
- *non-preemptive interaction*: requests are not preemptive, but are served at the end of the local operation with higher priority with respect to computation tasks.

As an example of performance measure we can consider the node utilization for different values of N.

UTO 3
A Parallel Program Example

UTO-group

We consider a parallel reactive program that monitors a system by controlling the outcome of three sensors. In Figure 2 the basic structure of the program is depicted. Processes

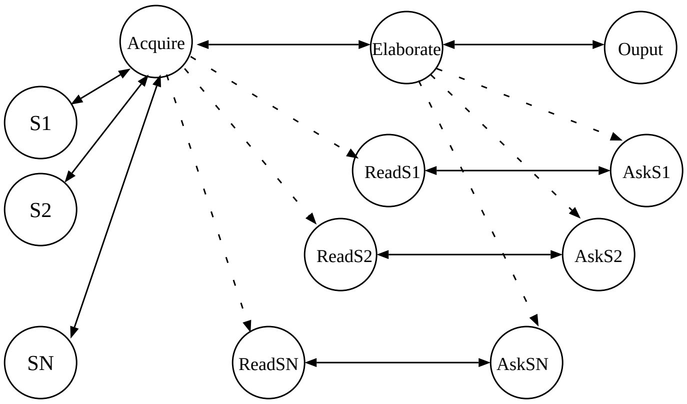


Figure 2: Structure of the program.

are represented by circles, communications are represented by arcs between processes and process activations are represented by dotted arcs. **S1**, **S2**, and **S3** represent the software architecture of the system to be monitored: an anomaly in the i-th device corresponds to a message being sent from **Si** to process **Acquire**, through channel Sig_i. We assume that signals are issued by processes **S1**, **S2** and **S3** according to Poisson processes of parameters λ_1, λ_2 and λ_3. The main function of process **Acquire** is to listen to the three channels Sig_i and to forward any message received on Sig_i to process **Elaborate**. The processing triggered by the reception of a signal consists of the acquirement of the current value of the other two signals, and of some computation on the measured values. It thus follows that, upon reception of a message from process **Acquire**, process **Elaborate** activates two of the three processes $\mathbf{Ask}_{S_i}$ with the special task of getting the values of the other two signals. In a similar way process **Acquire**, upon sending a signal to **Elaborate**, activates two processes $\mathbf{Read}_{S_i}$ with the specific task of reading the values and forwarding them to processes $\mathbf{Ask}_{S_i}$. There is also an **Output** process that prints the result of the computation. After a given number of signals is received, some statistics are computed and the monitoring system starts again.
The performance measures of interest for this system could be the maximum anomaly signal rate that the system can process and the probability of anomaly signal losses. The code of the parallel program is enclosed.

```
P0 :
while true
    PAR
        Acquire
        Elaborate
        Output
EndP0
```

```
Acquire : <acquiring process>
i:= 20
while (i> 0)
    seq
    ALT
        (Sig_1 ? val_S1)
            AcEl_1!val_S1
            PAR_11
                Read_S_2
                Read_S_3
        (Sig_2 ? val_S2)
            AcEl_2!val_S2
            PAR_12
                Read_S_1
                Read_S_3
        (Sig_3 ? val_S3)
            AcEl_3!val_S3
            PAR_13
                Read_S_1
                Read_S_2
    i:=i-1
EndAcquire
```

```
Output : < output process >
k := 20
while (i > 0)
    ElOut?datum
    output(datum)
    k := k -1
EndOutput
```

```
Elaborate : < signal processing >
j:= 20
while (j > 0)
    seq
    ALT
        (AcEl_1 ? val_S1)
            PAR_21
                Ask_S_2
                Ask_S_3
        (AcEl_2 ? val_S2)
            PAR_22
                Ask_S_1
                Ask_S_3
        (AcEl_3 ? val_S3)
            PAR_23
                Ask_S_1
                Ask_S_2
    datum := process(val_S1, val_S2, val_S3)
    ElOut! datum
    j := j-1
EndElaborate
```

```
Read_S_i :
    < acquire val_Si >
    (RdAs_i ! val_Si)
EndRead_S_i
```

```
Ask_S_i :
    (RdAs_i ? val_Si)
EndAsk_S_i
```

Figure 3: Code of the monitoring system

Appendix 3: Summarizing Evaluation

Please value one or more formalisms roughly by typically (++ very good, + good, ∘ satisfactory, – deficient, –– poor, additional comments and supporting literature should be included):

1) **General Characteristics**
   - Universality
   - Completeness
   - Maturity
   - Width of system study
   - Depth of system study
   - Accuracy
   - Acceptance and confidence
2) **Expressiveness**, i.e. the ability to model hardware and software in adequate detail
   - functional behaviour
   - temporal properties
   - topological, spatial relations
3) **Mathematical tractability**, i.e the ease to investigate, evaluate and predicate the system behaviour.

   Evaluation techniques:
   - Analytical
   - Numerical
   - Simulative

   Standard performance measures:
   - mean values
   - higher moments, distribution functions
4) **Hierarchical modelling and constructive design**, i.e. there exists a methodology which allows to structure complex systems and to build them systematically from smaller ones:
   - Modularity
   - Interface-concept
   - Composition
5) **Expenditure**, i.e. aggregate amount of time, energy and cost for the modelling process distinguishing:
   - modelling of system behaviour with tool support
   - modelling of system behaviour without tool support

- execution of experiments
- evaluation and interpretation of results
- design of large parallel systems

6) **Clearness and understandability**:

- without tool support
- with tool support
- graphical presentation