Distributed and Parallel Systems and HOOD4

"Ada in Europe 1995"

Frankfurt, Germany

October 02 - 06, 1995

Rainer Gerlich, Mladen Kerep
Dornier Satellitensysteme GmbH
RST16
D-88039 Friedrichshafen
Tel: +49/7545/8-2124
Fax +49/7545/8-5626

Daimler-Benz Aerospace
Dornier

Distributed and Parallel Systems and HOOD4

Rainer Gerlich, Mladen Kerep

Dornier Satellitensysteme GmbH
D-88039 Friedrichshafen, Germany
Phone +49/7545/8-2124 Fax +49/7545/8-5626

Abstract: For distributed and parallel computing the new version of HOOD, HOOD4 [1], brings a significant advantage: it decouples the logical design from the partitioning required to map software onto a net of processors. The HOOD Run-Time Support System (HRTS), introduced for HOOD4, will support an engineer to distribute the software. With the HOOD4 approach a software engineer can concentrate on the required functionality and has not to care about a certain hardware configuration which may have to be changed when he has finished the implementation. HOOD4 tackles the problem of software distribution in threefold manner: Firstly, it introduces clearly defined planes in a design where a cut can easily be done without impacting the logic of the software system. Secondly, the HRTS provides the means needed to establish the communication channels between the physically separated partitions. Thirdly, it allows to provide timing information from which a performance prediction can be derived. This allows to evaluate the performance of a hardware and software configuration already during the design phase. Based on the results of the performance prediction the optimum hardware configuration can be evaluated in advance. The HOOD4 concept for support of distributed systems was defined during the SOFTPAR project [2] by customising approaches for migration of software [3,4] and for real-time processing [5]. During the SOFTPAR project an exercise with this concept will be done for a high performance parallel C++ application [6,7] using tools of the project and a PowerPC network [8] and a workstation cluster.

Keywords: HOOD, Distributed Systems, Parallel Systems, Software Partitioning, Software Design

1. INTRODUCTION

Partitioning of software over a network of processors is a difficult task. Potential cuts must already be considered during design and the engineer must foresee the communication links. This makes the software rather inflexible for repartitioning. Usually, poor performance is detected when design and implementation is finished, unfortunately not before. Then much effort is needed to change partitioning of software.

The problem one is faced with here is well known, but difficult to solve if the following cyclic dependency between design, software partitioning and performance exists:

- the design fixes partitioning,
- partitioning impacts performance,
- performance constraints impact design.

If design and partitioning are not decoupled, one has to spend a lot of effort for another system configuration. Again, whether a change in design will solve the performance

problem can only be analysed when one has done the code change. So one has to perform a maintenance cycle in addition, but still the risk not to meet the performance goals when starting the re-design.

A solution to break this cyclic dependency is (see Fig. 1)

1. to make the design invariant against partitioning by introducing a concept which allows to perform the cuts where and when needed,
2. to ensure that a cut has minimum impact on the overall software system,
3. to ensure automated insertion of communication interfaces at the cut planes.

This is the way HOOD4 supports partitioning of distributed and parallel systems. The chosen approach also satisfies (hard) real-time needs because the interface between operations is not changed when an operation is accessed remotely[1] [2].

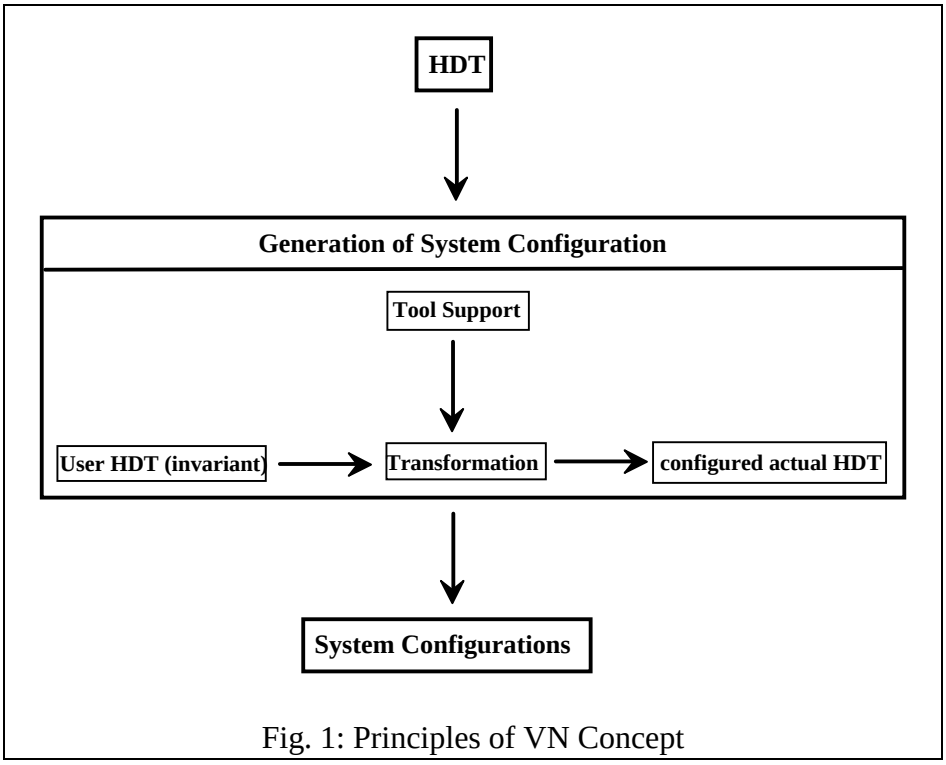

Fig. 1: Principles of VN Concept

Moreover, HOOD4 supports specification of real-time properties now. This information can be accessed via the HOOD SIF (Standard Interchange Format). In the SOFTPAR project a Performance Prediction Tool (PPT) will be established which allows early evaluation of performance of a distributed / parallel system already during the design phase based on this timing information.

The advantage of using the PPT is that one can get the performance figures in less time by simulation. Here the term "less time" is overloaded: firstly, one does not need to wait for completion of system development. Secondly, real runs may need more execution time. So one will get the performance figures faster. In consequence, more system configurations can be analysed for their performance.

---

[1] HOOD4 will also cover hard-real time processing by introduction of related attributes for operations. A final decision will be made in near future.

[2] Of course, one has to pay in terms of performance for communication through the network. But by parallel execution several reporting ER's may be executed in parallel so that deadlines can really be met whilst they may be lost when the ER's are executed on a single processor.

2. GETTING DESIGN STABILITY BY OBJECTS AND STUBS

Object-oriented software engineering introduces objects as entities of high stability: they encapsulate operations and data, provide a well-defined interface and hide everything else. Consequently, in an object-oriented approach objects are the ideal units of distribution.

The software architecture of a HOOD design is represented by a HOOD Design Tree (HDT) consisting of objects. Fig. 2 shows a sample HDT. It comprises six objects: two non-terminal objects 1 and 4 (including the root object) and four terminal objects 2, 3, 5 and 6.

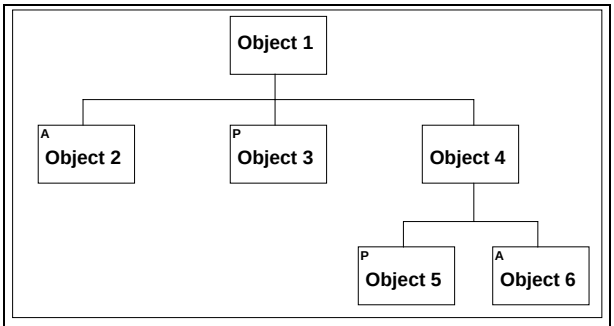

Fig.2: Sample Hood Design Tree

The letter in upper left corner indicates whether the object is an Active or Passive Object. However, in fact the nature of an object can be ignored when partitioning an HDT in HOOD4. So actually we do not care about these letters. In the HOOD4 approach a Passive Object has not to become an Active Object when it is migrated. This is an important step towards design stability: objects using a migrated object shall not recognise that an object is migrated. Design would become rather instable, if a Passive Object would have to become an Active Object when (re)-partitioning a HDT. The chosen VN approach is compatible with the CORBA concept.

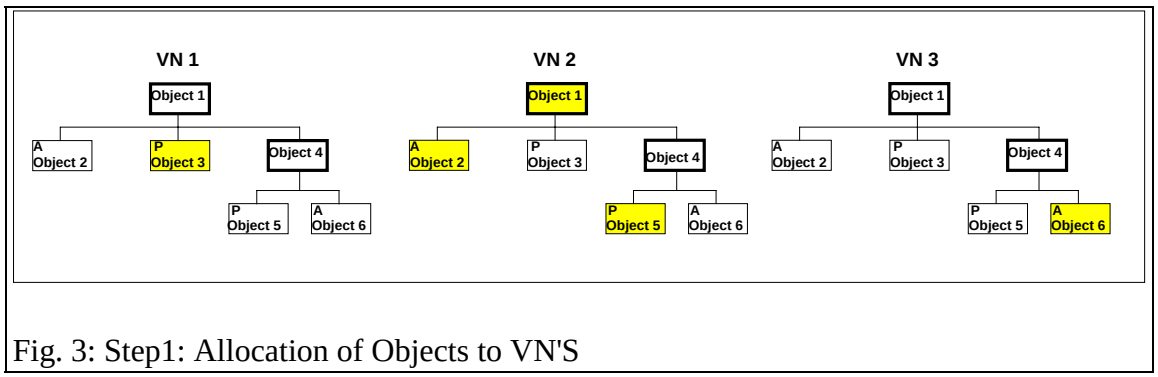

Fig. 3: Step1: Allocation of Objects to VN'S

Fig. 3 shows the partitioning of this sample HDT into three pieces: VN1, VN2 and VN3[3]. The shadowed (terminal) objects represent the real objects available on a certain node, the non-shadowed objects indicate that the corresponding software has been migrated to another node[4]. So we have the following situation for each Virtual Node:

VN1: objects 2, 5 and 6 must be accessed via the network[5],

---

[3] We will call these pieces "Virtual Nodes": these are the units of distribution. Seen from an HDT they look like software entities allocated to processor nodes. But they are independent of physical nodes. Therefore they are called "Virtual Nodes" to indicate partitioning of the HDT w.r.t. to units of distribution.

[4] The non-terminal objects have a special status: as empty shells they can exist on each Virtual Node.

[5] The term "network" implies direct communication on the same physical node, e.g. by message passing.

VN2: objects 3 and 6 must be accessed via the network,

VN3: objects 2, 3 and 5 must be accessed via the network.

As some objects are missing in each VN the resident objects cannot be executed. Therefore each VN needs a complementing functionality for handling of remote execution of missing operations. To build a specific environment for each of the VN's, this is one solution. However, this is a solution which makes the design heavily dependent on the allocation of objects to nodes. If a configuration is changed, then the complementing environment on each node has to be changed as well.

But one can easily see what is stable and identical for each of the nodes: this is the HDT which is still completely shown in Fig. 3. So the goal is to keep the full HDT on each node. The object-oriented design makes it easy: when keeping the interfaces of each of the missing objects, we will get identical HDT's on each node, although we do not have available the full functionality. Only by remote access we will get the needed functionality. But that is another problem, we will care about later.

The available objects in each VN cannot recognize the missing functionality of the migrated objects. So they can execute. In an object-oriented approach it is just sufficient to provide the interfaces in a VN. This ensures correct compilation and linking for each such HDT which is incomplete from a functional point of view. The missing functionality can be provided by remote execution, but this mechanism is hidden to the client objects.

The stability we aimed to get we have now achieved by means of stubs, which are (nearly) empty shells. They just keep the interface and a little bit more: the capability for communication and remote access of operations.

The task of a HOOD tool is now, to replace a real object (which shall be migrated to another VN) by such a stub (placeholder) which forwards messages to the remote object and returns its messages. This task is usually not too difficult and can be automated in most cases. What the tool has to do is:

1. to provide remote access, and
2. to convert pointers to values and vice versa and to pack/unpack values of parameters,

   There may be some exceptions, when too complex types occur in the parameter list of operations. Then a user has to provide operations which do the transformation.

The HRTS specifies which support a HOOD4 toolset must provide for "Virtual Nodes". So a user just establishes a HOOD design, defines the allocation of objects to Virtual Nodes and then asks the tool to generate the code for the Virtual Nodes and adds - if needed - operations for parameter transformation. Finally, a user has to specify which VN's shall be loaded on which Physical Node.

3. VIRTUAL AND PHYSICAL NODES

Having introduced the principal mechansim which ensures design stability we can now look on "Virtual Nodes" (VN) in more detail. Virtual Nodes consist of a subset of real objects of a HDT plus a set of stubs complementing this set towards the full HDT. VN's are heavy-weight processes which can be migrated. They include

- the full functionality of their real objects,
- the information visible via the interfaces of the migrated objects, and
- the capability for communication so that remote objects can be accessed.

A VN is executable under an operating system and the smallest unit of migration. By the size of a VN a user defines the granularity of load balancing.

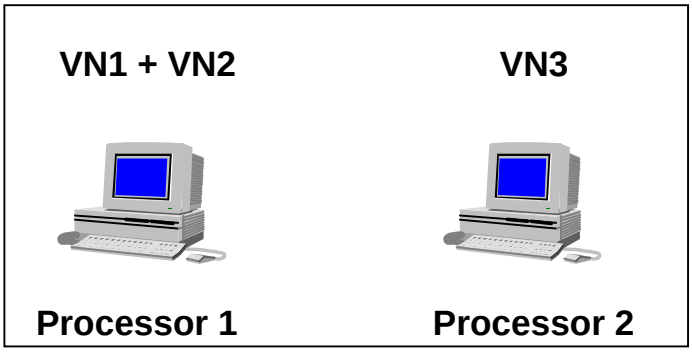

Fig. 4: Step2: Allocation of VN's to Processors)

The fact that VN's are heavy-weight processes is rather trivial. As they represent the full HDT (from a logical point of view), they represent the full system. As the full system is a heavy-weight process by definition, VN's are also heavy-weight processes. They may include Active Objects as light-weight processes if the operating system supports it[6].

VN's are allocated to "Physical Nodes" (PN) for execution. The mapping is not necessarily one to one[7]. More than one VN may be allocated to a PN (Fig. 4). Even all VN's may be allocated to one PN, only. This case is equivalent to a single processor system, but bears some communication overhead, of course. Also, a VN may contain only one object or it may contain all objects: these are the two extreme cases of partitioning a HDT into VN's.

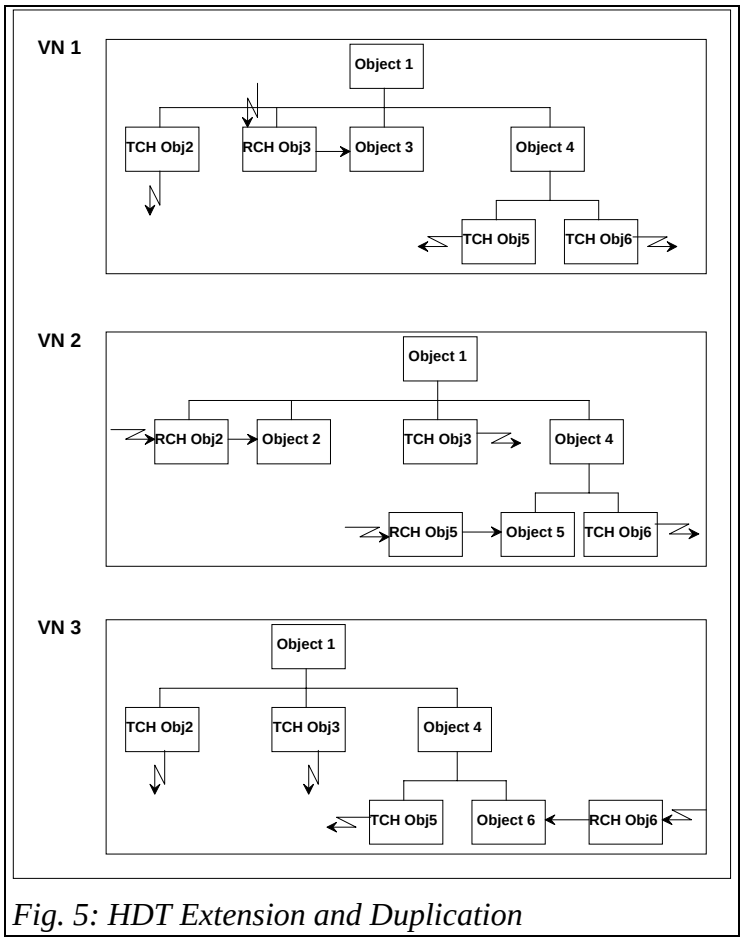

Fig. 5: HDT Extension and Duplication

Hence, distribution of a HDT across a network of processors is a *two-step* process:

1. a HDT is partitioned into VN's,

   i.e. objects are allocated to VN's
2. VN's are allocated to physical processors.

---

[6] In the SOFTPAR project the operating systems MPI and PVM allow only one heavy-weight process per physical processor. Therefore only one Active Object is possible in a VN. This is the worst case.

[7] This is the reason why we distinguish between "Physical Nodes" and "Virtual Nodes".

In HOOD4 the first step is performed at pre-run-time, the second step at pre-run-time or at run-time.

There is a high flexibility for selection of objects. Objects included in a VN may belong to different branches and different levels of the HDT hierarchy. There is no rule limiting the selection of objects. Passive Objects remain Passive Objects when they are migrated. Migration of other objects of an HDT is hidden for the objects inside a VN.

Fig. 5 shows what must be included in a VN: the remaining objects and communication handlers. "Transmitting Communication Handlers" (TCH) take the role of the migrated objects. For VN1 a TCH is needed for Object2, Object5 and Object6. They forward all execution requests to the real object and return output data. "Receiving Communication Handlers" (RCH) take the task to forward execution requests to the objects which shall be remotely accessed. In case of VN1 a RCH is needed for Object3. For VN2 and VN3 it is similar.

A TCH has to provide the following functionality (the numbers refer to Figs. 6-1 and 6-2 where the principal steps are shown in detail):

T1. ER service request (source: box 1, OP1 ER)

a. parameter transformation (box 2, OPCS_ER)

Parameters have to be transformed to a single data stream attached to an execution request. Only "values" can be included in such a data stream. Therefore addresses (pointers) have to be replaced by their values. In case of data structures the full data structure has to be included in the data stream. Also for nested data structures all addresses have to be replaced. In case of linked lists or cyclic references this transformation becomes very complex. Therefore a user has to provide the operations for parameter transformation. In simpler cases the transformation process may be automated.

b. putting the ER on the network (box 3, Client_OBCS)

The ER has to be transmitted to the VN and PN where the missing object has migrated to.

T2. Reception of results (destination: box 12, ER Termination)

a. getting results from network (box 10, Server_OBCS)

Results must be returned to the calling operation when they are received from the network including identification of the object and operation which shall receive the results.

b. inverse parameter transformation (box 11, OPCS_SER)

Now the received data stream has to be converted into a format which is equivalent for the normal direct access of an operation. Values which correspond to addresses or data structures have to be stored at the appropriate memory places. This step is inverse to step T1a.

A RCH has to include the following functionality:

R1. Reception of an ER (destination: box 6, OP1 OSTM)

a. getting parameters from network (box 4, Server_OBCS)

The object and operation to be executed must be identified. The data stream must be forwarded.

b. inverse parameter transformation (box 5, OPCS_SER)

The data stream must be converted to a format which corresponds to the direct call (see step T2b above).

R2. Transmission of results (source: box 7, OP1 Body)

a. parameter transformation (box 8, OPCS_ER)

The return parameters must be converted to a data stream similat to step T1a.

b. putting results onto the network (box 9, Client_OBCS)

The data stream including the results has to be sent to the requesting VN and PN .

Steps T1a, T2b, R1b, R2a, the parameter transformations, are depending on the actually called operation. Steps T1b, T2a, R1a and R2b are independent (or can be made independent) from the actually called operation.

In fact, box 3 and box 9 have to transmit data through the network. Their functionality is similar. So the same object Client_OBCS can be used. Also, boxes 4 and 10 receive information from the network and distribute it to dedicated objects. They can share the same object Server_OBCS, too. Similarly, parameter and result transformation can be allocated to one object OPCS_ER and the inverse transformation to object OPCS_SER.

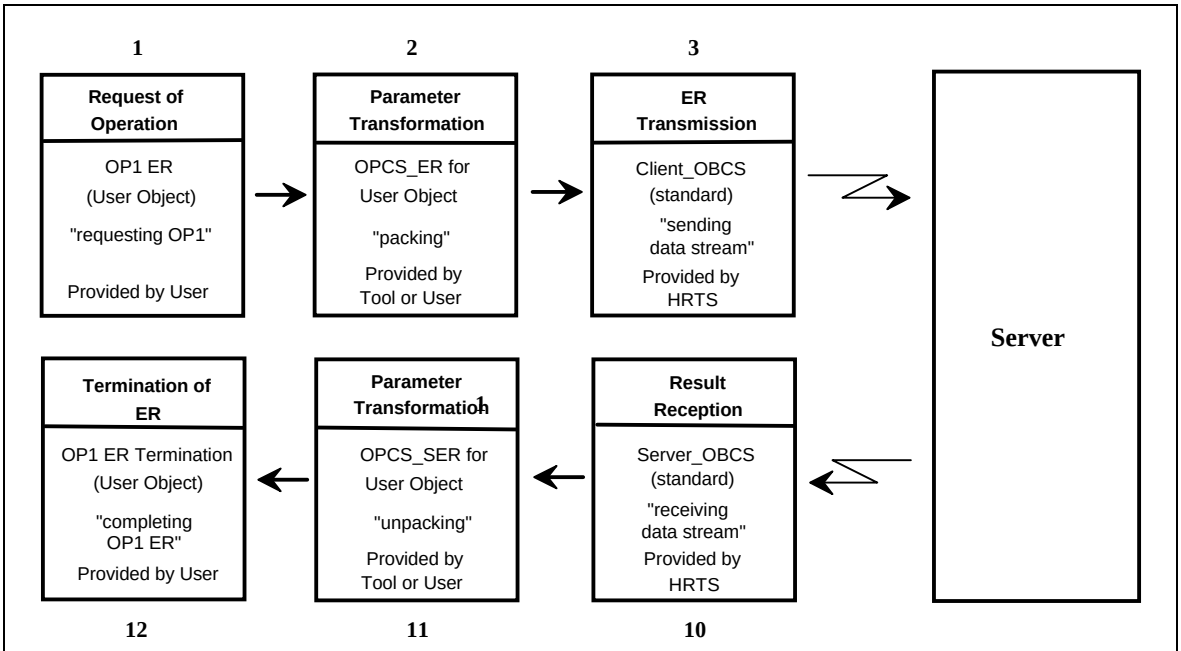

Fig. 6-1: Communication between Operations / Client Part

Fig. 6 which is divided into a client part (6-1) and a server part (6-2) shows in detail the communication and transformation steps.

Of course, network performance may impact real-time performance of a distributed system. However, this is unavoidable when one needs to use a set of processors. But the

performance of a distributed / parallel system is surely better than the performance of a single processor system bearing all the load of the complete software system[8].

By introducing VN's as independent units of distribution one decouples partitioning of software from allocation of software to hardware. First comes design stability, then partitioning to VN's and then allocation of VN's to PN's. This leaves a high degree of freedom to the engineer to optimise the hardware and software configuration of distributed and parallel systems.

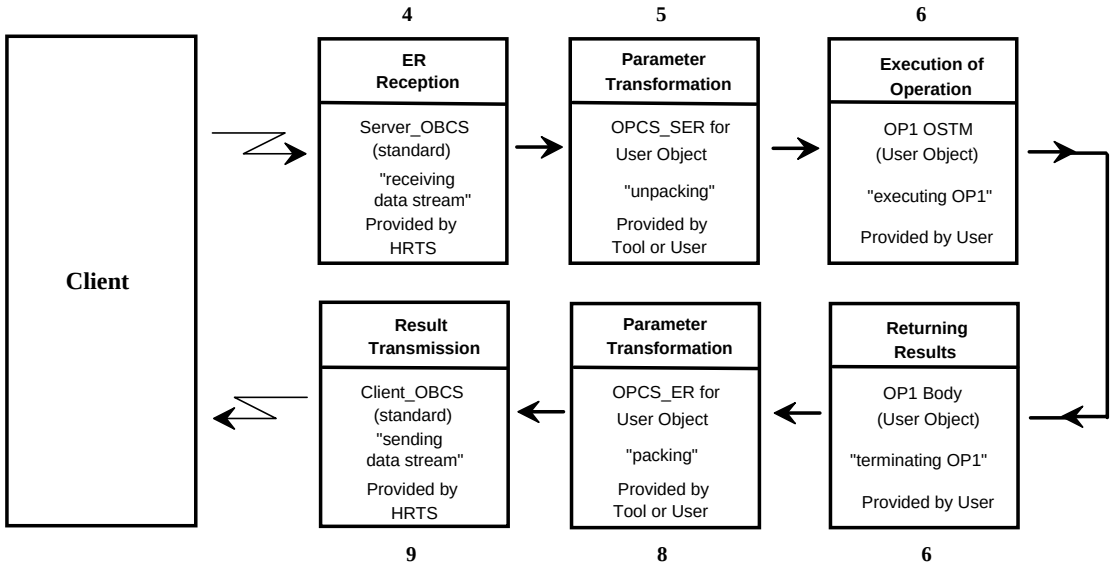

Fig. 6-2: Communication between Operations / Server Part

4. PERFORMANCE PREDICTION

Performance of a distributed / parallel system depends on the overall processor and channel utilisation to which each processor contributes when it executes operations. High processor utilisation can only be achieved when distributed operations can execute in parallel. Due to the usually highly complex dependencies between operations one does not know if decoupling is good or bad. Therefore one needs to get sufficient knowledge on sequential dependencies.

Objects which use each other by unconstrained operations or by HSER's (Highly Synchronous Execution Requests) can share the same VN. There is no advantage if they are allocated to different processors, but there might be a disadvantage by communication overhead. Vice versa, higher utilisation can be achieved, when objects using each other by loosely coupled execution requests (e.g. ASER, LSER, RASER, RLSER)[9] are allocated to different VN's and different PN's.

---

[8] Communication overhead (at least) must be compensated by parallelisation: this requires sufficient degree of parallelisation and sufficient parallel computing resources.

[9] HOOD4 introduces new types of execution requests (RASER, RLSER) to allow parallelisation with return of results. Therefore we have: ASER = Asynchronous Execution Request, LSER = Loosely Synchronisation Request, RASER = Asynchronous Execution Request Returning Results, RLSER = Loosely Execution Request Returning Results

The problem is how shall we know which objects depend on each other and which do not depend on each other. Usually, the problem is so complex that it cannot be analysed theoretically. Therefore simulation is needed to provide the right information.

The degree of coupling between objects depends on the type the operations couple themselves via execution requests (ER). Several ER types are defined in HOOD4. Some new ER types have been added (compared with HOOD3) in order to meet the objectives of distributed and parallel systems. For derivation of processor utilisation we need knowledge on these dependencies. Also, we need information on the execution time.

The type of execution request, the timing information (e.g. time out, worst case execution time) can now be expressed in HOOD4[10]. Together with additional assumptions on the execution time profile of operations and performance of the network, a simulation can be executed which provides representative information of processor utilisation of the actual hardware and software configuration. This is the task of the Performance Prediction Tool (PPT) of the SOFTPAR project. The PPT shall provide information on the performance of a certain system configuration by simulation and modelling of operations, processors and network topology.

4.1 The User Interface

A user has to provide information on software design and configuration. This information comes from different sources and may have to be defined

- only once,
  - -- e.g. the objects and operations during development,
- from application to application,
  - -- e.g. the scheduling strategy, or
  - -- network properties
    which depend on the chosen processor type
    (transputer, workstation)
- frequently for optimisation of object distribution over the processors.

In addition to above information which is precisely available by design or configuration a user has also to provide some estimations on the timing profile on the operations. This is necessary because the operations may not be implemented when the prediction is needed. If the operations are already implemented and there timing profile is available a user has also to characterise the timing profile by parameters.

The timing profile may impact the prediction significantly. This can be investigated by running several tests with different profiles in order to see what the performance is and if it depends critically on the profile or not.

The approach used for the PPT is that the body of an operation consists of a sequence of local processing in the operation itself (processor time consumption by the operation) and ER's. For better user support standard sequences can be defined and embedded in other sequences.

Fig. 7 shows the PPT interfaces and its principal components.

---

[10] The final decision on how the attributes look like is pending.

1. By HOOD information is provided on objects and their operations and the ER's issued inside an operation. This allows to provide information on scheduling, call of operations and operations' timing profile.
2. A user has to selcect a certain scheduling strategy, e.g. for hard real-time processing. This impacts the scheduler included in the PPT.
3. The properties of the actual hardware configuration especially of the network, influence transmission time of data. Data may be routed by a certain strategy through a network, e.g. by dedicated channels between the physical processors, by a bus to which all processors are connected or by another more complex connection like in a crystal frame work.
4. Allocation of objects and operations to physical procesors impact degree of parallel execution as already mentionned above. This information is delivered from the HPC++[11] configuration files.

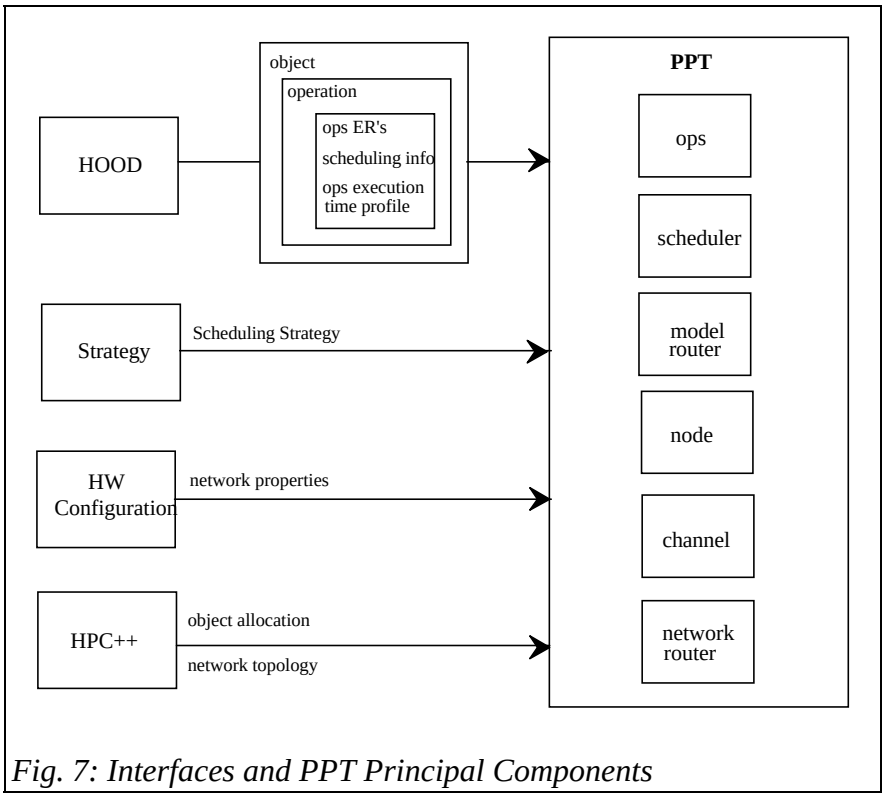

Fig. 7: Interfaces and PPT Principal Components

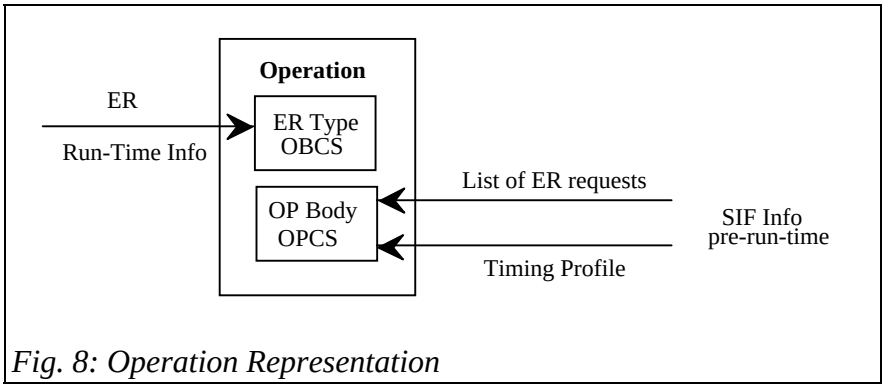

Fig. 8: Operation Representation

The components of the PPT are: the operations, the scheduler, the model router which forwards the ER's between the operations and returns the results, the nodes (physical processors) providing CPU power as shared resource for the operations, the channels of the network, and the network router which has the knowledge by which channels the data shall be transmitted between the processors.

For all the components model types are provided which can be instantiated for the desired number of elements. The

[11] High Performance C++

behaviour of the instances can be customized by parameters.

Fig. 8 shows how an operation is modeled for the simulation. In object-oriented manner it is divided into an interface and a body. The interface handles the protocol of the ER type and receives and returns data. For simulation the length of the input and output data stream is relevant, only.

An operation body executes normal statements and issues ER's according to information received from a data file which may be derived from SIF information.

For each ER type a template is provided for protocol handling in an operation's interface and in an operation's body for execution of an ER.

Execution of an operation is performed by acceptance of an ER or its rejection by the OSTM[12] and the execution of the operation body. For the PPT an operation body is represented by a sequence of "Basic Operations" which can be reused by a number of operations. This allows to provide templates for certain timing profiles (Fig. 9).

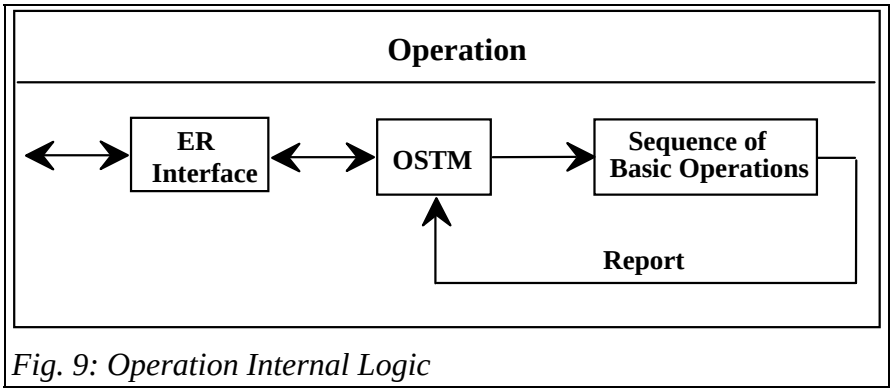

Fig. 9: Operation Internal Logic

A "Basic Operation Sequence" is shown in Fig. 10. The length of the input and output data is relevant only. Such a sequence is represented by "local" execution of code and ER's. The local code may be empty. For a block of local code or an ER block a probability for its execution and a repetition factor can be specified which provide the capability to simulate IF's and LOOPs. For each block a type is provided: for representationn of pure code and of an ER type.

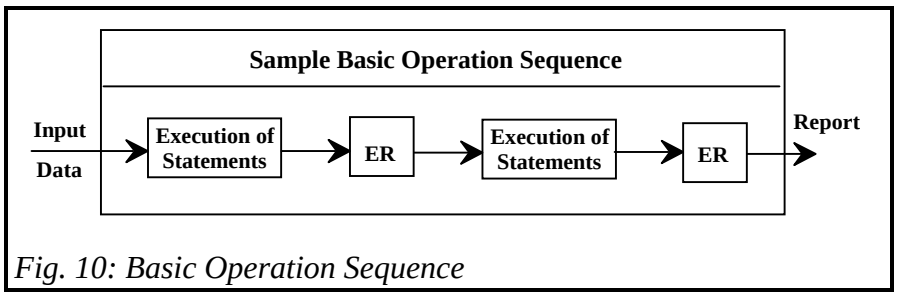

Fig. 10: Basic Operation Sequence

The PPT is based on the commerical tool SES/workbench [9] which is a tool for performance simulation.

4.2 Results

The sample configuration consists of eight processors and a set of objects and operations which are allocated to the processors. The results evaluate dependencies on network configuration and on ER type. Also, the capabilities for evaluation of communication between processors and channel length are shown. The graphical figures are derived with

[12] Operation State Transtion Machine which is a Finite State Machine and decides according to the actual state of an operation wheter it accepts the request or not.

SES/graph. When the interfaces to other tools are implemented which specifically support graphical presentation of properties of parallel and distributed systems, the visualisation capabilities will become much better.

The system configuration consists of eight physical nodes (processors) connected by three different network configurations:

1. all processors are connected to a bus
   e.g. workstation cluster (Ethernet)
2. the processors are connected through a network described by a matrix
   e.g. transputer network

   We consider in the examples "virtual channels". A "virtual channel" between processors may consist of several physical channels which connect two physical processors directly with each other, i.e. another processor may be involved in data transmission between two processors.
3. all processors are connected directly by physical channels.

Operations were allocated to four of the eight processors only as a starting point.

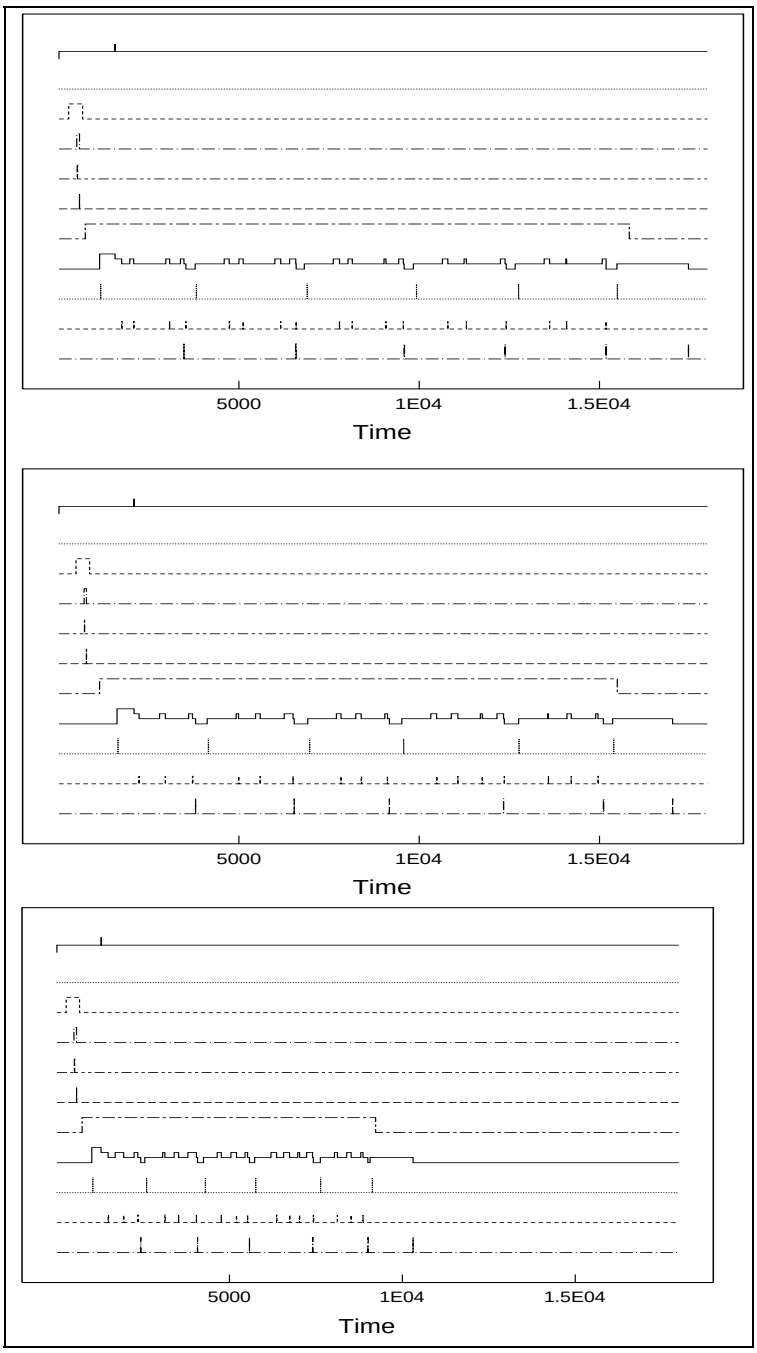

Fig. 11 shows the execution time of operations for the three network configurations and use of RLSER's. As far as an operation is active, the trace is at "high" level, when it is inactive, it is at "low" level. The top figure corresponds to the bus configuration, the middle figure to the matrix network and the bottom figure corresponds to the direct physical connection of processors. In the example the operations have a high amount of communication.

Between the bus and the matrix network there is nearly no difference in total execution time. The matrix network has a small advantage which is hardly to see. In case of direct processor connection the execution is significantly lower (10 000 compared to 18000). This means that the bus traffic is a performance bottleneck.

Fig. 11: Execution of Operations at high bus traffic for three different network configurations

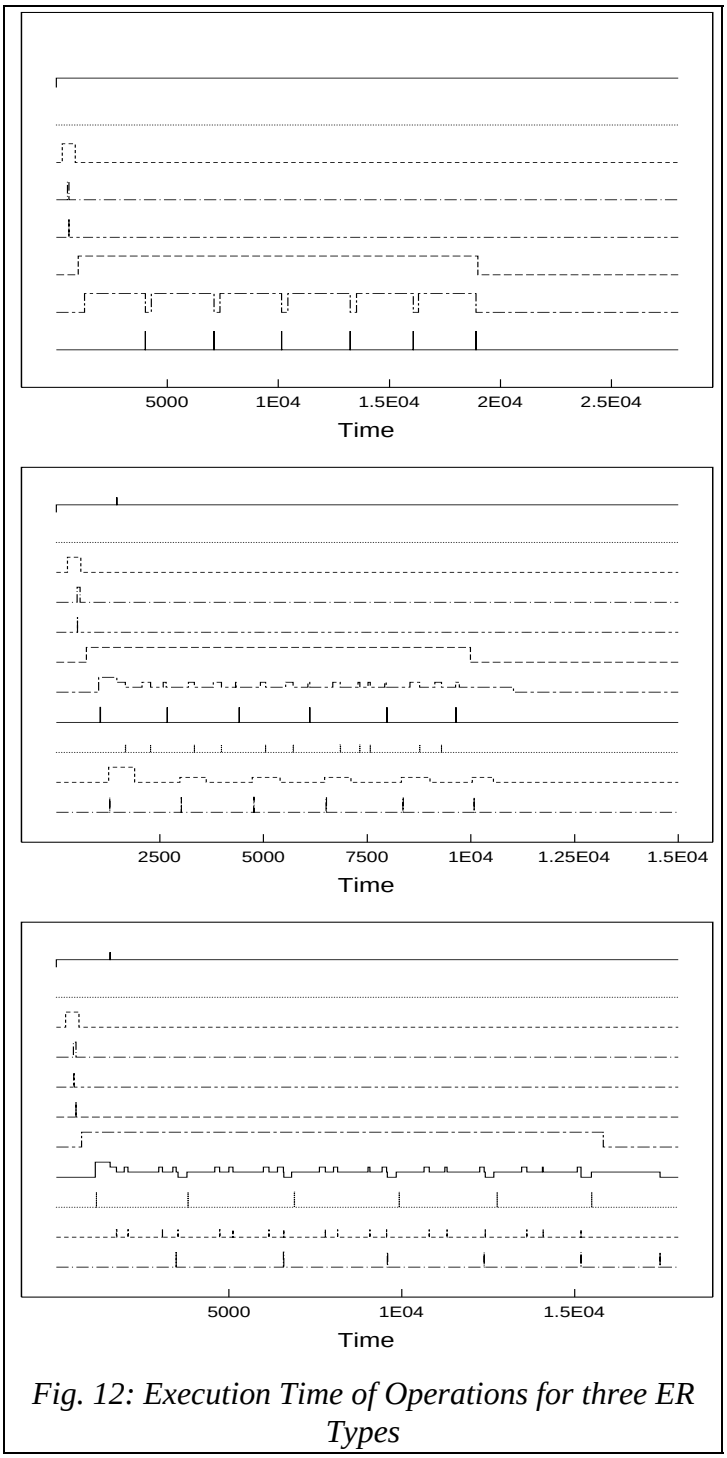

Fig. 12: Execution Time of Operations for three ER Types

Fig. 12 evaluates the dependency on the ER types. In the example a sequence of ER calls for the same type of ER's (HSER, LSER, RLSER) is executed. The execution time for a HSER is shown by the top figure. The repetition of ER calls is clearly to see.

For LSER and RLSER the execution requests are issued one after the other in the predefined sequence when executing operation bodies in parallel. In case of an RLSER no synchronisation point for a report is inserted. The report is just received when it arrives.

The middle figure shows the results for a LSER. Now we have more traces due to the fact that acknowledges (ACK) and negative acknowledges (NAK) can occur. As LSER's can be executed in parallel an operation can be called more than once at a certain time. The y-axis shows not only that a certain operation is called, but it gives also information on the number of additional calls which are rejected in our case. The steps on the y-axis indicate how many accesses to an operation are performed at a certain time.

The bottom figure gives the results for the RLSER. Of interest are traces 7, 8 and 9 (counted from top of each figure) and 9 and 10. Trace 7 shows the number of calls and operation execution (this operation shall be called "OPS1"), trace 8 shows the ACK'S and trace 9 the NAK's. Similarly it is for traces 9 and 10. Trace 9 represents number of accesses for another operation ("OPS2") and trace 10 shows the ACK's from that operation. One can see that the second operation does not reject any request because it has already finished when a new request arrives. For the first operation it is different. A lot of NAK's occur as the operation is frequently called and more CPU time is consumed.

The worst case in view of total execution time is obtained for a RLSER, of course, because more data and more CPU time are consumed for preparation and transmitting of the reports.

The execution state and protocol handling for OPS1 is given by traces 8 - 11. Trace 8 shows the execution state, trace 9 the ACK's, trace 10 the NAK's and trace 11 the reports. The report is extending the execution time sigificantly so that more NAK's occur. In case of reception of a NAK the request is repeated in a loop for a predefined number (user-defined parameter) of events. Between two following requests a delay (which is also a user defined parameter) is executed. As more NAK's occured the time until a succesful reply is higher. Therefore an operation has to wait a longer time and the total execution time is significantly increased.

These results are interesting for two reasons. Firstly, one sees that the report may extend the execution time at high network traffic and high data rates. Secondly, this is an example for a case where a small increase in an operation's execution time can dramatically impact the total execution time due to the number of additional NAK's and related wait delays.

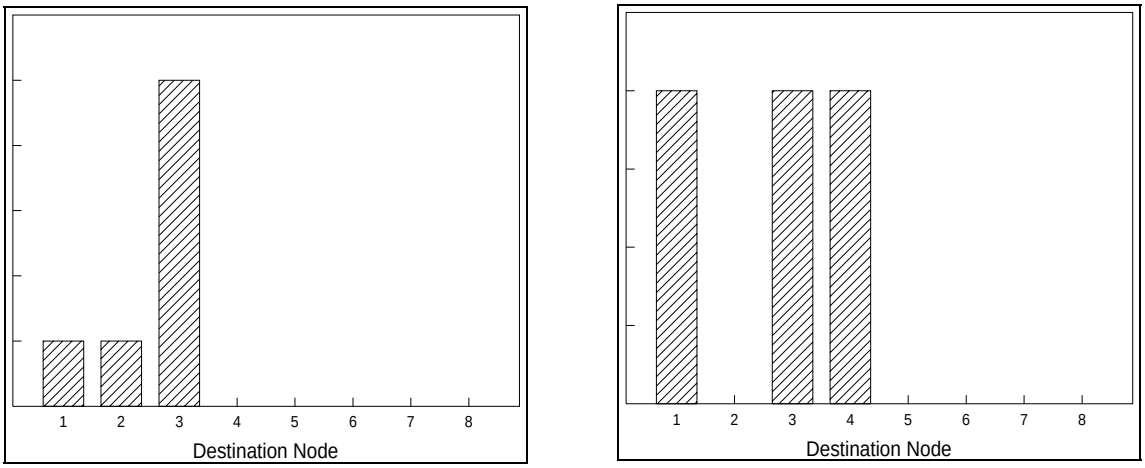

Fig. 13: Communication between Processors

Fig. 13 shows the number of communications from one processor to all other ones for two physical nodes. Such figures indicate how much the distributed operations depend on each other. If processor pairs create a high amount of messages compared to the other ones one should include the related objects into the same VN or allocate the VN's on the same processor at least in order to reduce the overhead for message exchange.

Fig. 14 gives the amount of input and output data exchanged between one processor and the other ones. This is another criterion for allocation of VN's to processors. The amount of data create loads on the data channels. If this load is too high it can be reduced by allocating objects with an high amount of data exchange on the same processor or in the same VN.

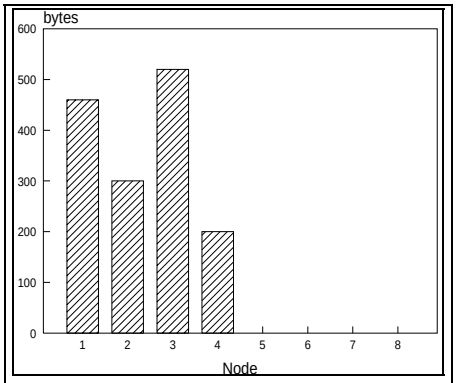

Fig. 14: Input - Output Data Transfer between

Fig. 15: Number of HOPS

Two processors may not be directly connected to each other by physical channels, but by virtual channels consisting of a number of physical channels which are used one after the other for transmission of data through the network. The transmission time of a virtual channel is the sum of the transmission time of its physical channels. As can be seen by Fig. 11 such a time delay can extend the execution significantly if transmission times are in the order of the normal processing times. Therfore it is important to know how long paths are really and how frequently they are used. Fig. 15 gives this information for the matrix network. The length of a virtual channel is also expressed in HOPS[13]. The number of HOPS is the number of physical channels a virtual channel consists of minus one, i.e. the number of transitions from one physical channel to the next one. For the bus configuration and the direct connection there is only one physical channel to be used, hence the maximum number of HOPS is one.

Above examples represent only a small percentage of information which can be derived on system performance by the PPT. E.g. response times, number of time-outs, number of ACK's and NAK's for each operation, each VN or each processor can also be derived and presented by histograms, statistics and graphical capabilities of other evaluation tools.

Each such performance information can be provided on an average level, e.g. per processor, per VN or per object, or more detailed for each operation, for each ER in an operation etc. An engineer gets a lot of possibilities to identify which component in the system consumes which amount of time. He can identify on a high level a global source of time consumption, e.g. a VN as a performance bottleneck, and can then look into this component to identify which of the subcomponents creates the load.

5. CONCLUSIONS

HOOD4 significantly improves the support for distributed and parallel (real-time) systems. It shows the way how to achieve stability of design without limiting the repartitioning capabilities and supports it by toolsets and HRTS. It further allows to define and to extract the information needed to predict performance from the design.

This information is taken as input for the PPT. The first results obtained from the Performance Prediction Tool prove that the essential performance bottelnecks can be identified.

The SOFTPAR project has introduced significant improvements to HOOD4: new types of execution requests, the new VN concept, and a proposal for harmonisation of HOOD and hard real-time needs. It has applied it to a C++ pilot application and provides a tool for optimisation of VN configuration.

---

[13] A "HOP" is a transition from one physical channel to the next one.

In summary, much experience was gained in applying HOOD4 for parallel and distributed systems and for optimisation of system configuration. The VN concept of HOOD4 was validated for C++ in a parallel environment. Performance prediction will be validated by the pilot application by end of this year. This makes it easier for future users to apply the VN concept and to optimise their system configuration.

REFERENCES

[1] HOOD4 Reference Manual, Draft,

[2] CEC ESPRIT Project 8451 "SOFTPAR" , *A Software Factory for the Development of Parallel Applications, November 1994*

[3] R.Gerlich: , 1st EUROSPACE Symposium "Ada in Aerospace", December 1990, Barcelona, Spain, pp. 254-272

[4] R.Gerlich: *Dynamic Configuration with Ada*, 10th National Conference on Ada Technology, February 1992, Washington D.C., pp.276-284

[5] Hard Real-Time System Kernal Operating System, ESTEC contract no. 9198/90/NL/SF, Final Report 1993, Noordwjk, The Netherlands

[6] J.M.Letteron, J.Bancroft, K.Wolf, A.Holtz, M.Lang, R.Gerlich, V.Debus: *HOOD and Parallelism in the SOFTPAR Project, HPCN95, Milano, Italy*

[7] K.Wolf, A.Holtz, M.Lang: *High Performance C++*, HPCN95, Milano, Italy

[8] PowerExplorer, Parsytec Computer GmbH, Juelicher Strasse 338, D-52070 Aachen

[9] SES/workbench, Scientific and Engineering Software Inc., Building A, 4301 Westbank Drive, Austin, Texas, 78746-6564, USA