The Computing Center Software
A Step Towards Metacomputing

Friedhelm Ramme, Thomas Römke

PC2

PADERBORN
CENTER FOR
PARALLEL
COMPUTING

Technical Report PC^2/TR-007-94

March 1994

PC^2 - Paderborn Center for Parallel Computing, Universität-GH Paderborn, D-33095 Paderborn, Germany

Phone: +49 5251 60 3342 • Fax: +49 5251 60 3436 • email: pc2-team@uni-paderborn.de

The Computing Center Software A Step Towards Metacomputing

F. Ramme, T. Römke

Paderborn Center for Parallel Computing

University of Paderborn
Warburger Str. 100
33095 Paderborn, Germany

e-mail : ram@uni-paderborn.de, tomtom@uni-paderborn.de

Abstract

Computing resources which are transparently available to the user via networked environments are often seen as a *metacomputer*. In this sense, a metacomputer is a network of heterogeneous, computational resources linked by software in such a way that they can be used as easily as a single computational unit. During the last years our work has concentrated on developing methods and tools to provide a transparent and vendor independent hardware management system to the users, giving *the same look and feel* when accessing different types of parallel and distributed computer systems. Solving the *hardware management problem* up to a high logical level will bring the idea of metacomputing a large step to practice. This paper describes the main components of the *Computing Center Software*, its central design goals and how they are matched. The object oriented approach of CCS, implemented as a message based distributed application by itself, has proved to be a flexible method to face the challenge of integrating new computer architectures into an existing management environment. To our knowledge, this is the first hardware management software package that serves as a metacomputing basis for supporting heterogeneous computing in a vendor independent manner. We will present on-going activities, give some prospects and show a sophisticated usage of CCS towards Wide Area Metacomputing.

Keywords: Distributed Parallel Computing, Resource and Access Management, Metacomputing, Heterogeneous System Integration

1 Introduction

Today, computing systems accessible to researchers with "Grand Challenge" problems consist of a hardware mixture ranging from clusters of workstations to parallel supercomputers. This hardware is available via geographically distributed networks with various communication capabilities. Due to local demands, the exact collection of hardware available at any one time will change dynamically. Outside laboratories, users are neither willing to face the technical system details nor to handle the vendor dependent system software. However, general interest is focused on a central user and accounting management, utilization of the machinery and on batch processing facilities on all systems which are not exclusively dedicated to interactive usage.

Following the discussion in [18] we emphasize that the first stage in developing a metacomputer is to create and harness the software to make the user's job of utilizing different computational resources easier. For any one project, a typical user might use a desktop workstation, a remote supercomputer, a mainframe supporting the mass storage archive, and a specialized graphics computer. Thereby conventional supercomputer are more and more replaced by remotely accessible massively parallel computer systems [7]. Nevertheless, we will probably see a long period of time where conventional supercomputer and MPP systems are co-existing, which will make the idea of metacomputing more and more important for modern research institutes.
Some users have worked in traditional heterogeneous environments for the past decade, using *adhoc* custom solutions, providing specific capabilities at best. In most cases users have to move data around and port applications from machine to machine by hand. The goal of building a metacomputer is elimination of the specific problems encountered in carrying out a project on such a diverse collection of computer systems. The *first stage* is largely a software and hardware integration effort. It involves interconnecting all of the local resources with high-performance networks, implementing a distributed file system, coordinating user accounts across the various computational elements, and making the environment seamless using existing technology. Today, in most representative locations this first stage is reached. Thus we can reasonably assume that common services like NFS, NIS, e-mail etc. exist.
The *second stage* in metacomputer development moves beyond the software integration of a heterogeneous network of computers. It involves spreading a single application across several computers, allowing a center's heterogeneous collection of computers to work in concert on a single problem. This enables users to attempt types of computing that are impossible without the virtual machinery. Software that allows this to be done in a general way (as opposed to one-time, *adhoc* solutions) is just now emerging.
The *third stage* in metacomputer evolution will be a transparent national network that will dramatically increase the computational and information resources available to an application. This stage involves more than having the local metacomputer use remote resources. Stage three involves putting into place both adequate wide-area network (WAN) infrastructure and developing standards at the administrative, file system, security, accounting, and other levels to allow multiple local-area network (LAN) metacomputers to cooperate. A first approach in preparing the third stage is seen in the United States, where a collaboration between the five National Science Foundation (NSF) supercomputer centers is established to create a "national virtual machine room" [18].

This short perspective should be sufficient to see a main stream in the development of leading computing centers. Considering stage two and three mentioned above, we can extract three relatively independent subtasks.

- Establishing a LAN infrastructure between local computer systems is a central part in stage one. However, connecting various local metacomputing centers by high bandwidth WANs is a prerequisite for stage three, too.

A central goal of stage two involves distributing a single application across several computers within a local environment. This topic is extended by stage three in spreading a single application beyond the local environment. Furthermore, the development of standards at the administrative, security, accounting and other levels like batch processing, load balancing, access control or user management are mentioned.
We suggest to solve this problem by using two independent but well supplying subtasks.

- Before starting a distributed application, it must be assured, that the computational resources required by this application are available and configured in an appropriate way. And because we have to think in terms of a fully automated research center, the selection of suitable machines and the configuration should be concealed to the user. When the user's application is terminated all requested resources must be released. There is no doubt that a computing center needs a central user and accounting management or a vendor independent batch processing facility. We summarize those points as *transparent user and hardware resource management.*

Once the requested resources are configured it is possible to connect the hardware side to the user's terminal by using virtual terminals. Up to now, everything stated within the second subtask can be done independently of the (software) programming environment preferred by the user or required by an application. This key consideration leads to the third subtask.

- Providing the user with a programming environment running on a heterogeneous hardware platform and concealing the underlying hardware properties will supplement the second subtask. At that level there is a need for standard packages that help an application designer to parallelize code, decompose code into functional units, and spread that distributed application onto the metacomputer. Today, several encouraging packages, e.g. PVM[1], Express[2], MPI[3], Linda[4], DNQS[5], and Parix[6] are emerging.

[1]PVM [6] was developed by a team at Oak Ridge National Laboratory, Univ. of Tennessee and Emory Univ.

[2]Express [4] was developed at CalTech and is distributed by ParaSoft. It is a suite of tools similar to PVM.

[3]MPI [9] is a Message-Passing Interface. Its development was supported by ARPA, NSF and the CEC through Esprit project PPPE.

[4]Networked Linda [2, 13] is a Linda implementation which provides access to parallelism across a heterogeneous network of computers. It was developed at Yale Univ.

[5]DNQS [8, 12], a Distributed Network Queuing System and NQS, a Network Queuing System were first developed at Florida State Univ. DNQS was entirely rewritten by McGill Univ.

[6]Parix [17] is a runtime kernel, supporting virtual topologies on grid based transputer systems. It was developed by Parsytec Ltd, Germany.

Based on these considerations we developed a software package for the subtask of transparent user and hardware resource management. A first sample installation was tested with the MPP hardware environment at the Univ. of Technology, Aachen. Refined and enhanced versions were continuously evaluated at the Paderborn Center for Parallel Computing, PC^2 , serving various types of MPP systems. Further installations were made at the computing center at TU Hamburg-Harburg and the computing center at TU Chemnitz. A number of other sites state their interest.
The objective of the current implementation focus on an automated operating of various types of parallel and distributed systems (e.g. freely configurable, partitionable, fixed topology machines and even clusters of workstations) by a common interface. Its underlying concept includes *error recovery* and *high level load balancing* aspects. This software package is valuable for everyone, who offers parallel or distributed computational power, supports various developmental environments to a large and heterogeneous user community, or who administers different parallel systems or gives access via unstable connection lines (WANs, phone lines). We emphasis on strongly separating the user's view from that of the machines addressed. Due to CCS, MPP systems are viewed as manageable tools. Porting activities are considerably simplified by concealing all technical details and presenting the same look and feel to the user community. To our knowledge, it is the first time that a hardware management software package is presented that can serve as a metacomputing basis for supporting heterogeneous computing in a vendor independent manner.

This paper is structured as follows:
In Section 2 we discuss the general considerations leading to the distributed management approach and the development of a general purpose Resource Description Language (RDL). The main components of CCS are described in 3. We explain the central user- and hardware resource management, handling of connection break downs, integrating new system families and the modification of the machinery during runtime. In Section 4 we describe a sample interactive user session and present the machine independent batch processing facilities. A sophisticated usage of CCS, towards stage three in metacomputing evolution, concludes our paper.

2 Frame-Work

2.1 Matching the Needs

The difficulties in operating MPP systems today are determined by the exclusive usage of resources for applications (SC, GCPP, GCel, iPSC/860, ...) and / or the system partition scheme (PPGC, GCel, Paragon, T3D, ...). Configuring small user partitions on large reconfigurable parallel machines can lead to a considerable waste of resources. Though some of them remain unused for certain periods of time, several applications might be waiting to be configured because they cannot be mapped onto free resources. On the other hand, a user cannot be expected to redesign his application to meet the resources in time. Therefore, one goal in the design of CCS was to minimize the waste resulting from unused resources by changing the configuration order of requests. Further aims were a fair treatment of requests and minimizing waiting times, leading to a high level load balancing. However, due to the limited space, we can not emphasize on the global resource optimization concept or discuss efficient scheduling algorithms. This is the subject of a separate work, based on the general concepts presented now [10].

To attain an optimized access to the machinery, users should be provided with a virtual view of the physical resources. This way, all technical details can be concealed by the hardware resource management. The operator should have the possibility to modify the computer environment during runtime (e.g. integrating new systems, changing interconnection structures, separating parts of the metacomputer for maintenance purposes, etc.). Even such massive interferences will remain transparent to the users by the virtual view provided. They will only perceive a change in the set of currently available resources with respect to the ordinary configuration. Merely the administrator of the computing center has to know the technical details of his metacomputer. On a high level he can specify the computer systems by their essential qualities and their interconnection structures. Now the user can formulate his resource requirements without preliminary knowledge of the physical world at different research centers by using a flexible Resource Description Language (see 2.2).
This standardizing and highly flexible approach implicitly considers fault tolerance aspects. The breakdown of a computer or parts within its domain can be handled similarly to changes within the environment by an operator. The difference is that a management process will notice such a breakdown and can act accordingly. Supporting unstable connection lines is only a special case of this very general method. Providing the user with the possibility to specify the structure and the properties of the requested resources at a pure logical level to the user gives the operator the flexibility needed. A pure hardware resource management can serve as a transparent link between these two spheres. As in modern computer technology, the management software should run in the front-end area of the systems to be administered and not on the systems itself. Thus, the front-ends consist of task dependent server stations transparently used within the LAN of a local metacomputer.

When deriving the software structure of CCS we found an object oriented approach suits it best. Each machine can be encapsulated by a software module, say a machine manager, hiding all the technical details of that machine. Thus, we can keep the term 'machine' very broad, e.g. even viewing a dedicated workstation cluster as a distributed machine. Furthermore, we have fixed other independent objects corresponding to a central user and hardware management, batch-job and queue management, user interface shell, application shell and a database object performing accounting and access control. Several less relevant objects are omitted in this list. Due to the geographically distributed locations of machines and services the objects have to be distributed as well.
Within CCS, this very general model is *implemented as a distributed process system* joined together by asynchronously exchanging messages. Programming the processes as daemons and using the interprocess communication facilities provided by Unix, keeps CCS portable and makes the software self-controlling. Therefore, an error recovery basis is provided to keep the management running even if parts of the metacomputer are malfunctioning.

2.2 Using RDL

The heterogeneity of today's complex network environments demands an adequate abstraction level. A representation of resources should, however, be independent of the respective application environment (*software abstraction*) and the machines running (*hardware abstraction*). Taking this into account, a basis well equipped for the development of compatible tools can be built. The resulting tools can easily be adapted and will work with the most diverse machines. In a broad sense, we have to specify *three abstraction levels* of resource structures.

> The *vendor-level* describes the internal structure of a specific machine. The *operator-level*, which is the metacomputer by itself, describes the connections of the various machines within the LAN and their general properties. Finally, at the *user-level*, a logical network topology is specified with its characteristics to be configured within a given hardware environment.

The Resource Description Language (RDL) [1, 11] exploits the similarities between the representations and descriptions above, thus forming a uniform methodology for handling resources. Using the hierarchical approach of RDL, it is possible to group logical structures, having their main properties in mind, as well as specifying physical structures reflecting the architectural conditions. Each hierarchical layer is composed of a definition part for defining attributes and macros, a declaration part for declaring the objects of the current layer, and a connection part specifying the connections between those objects and the surrounding unit. The current implementation is restricted to three layers. This has so far been sufficient for the representation of complex conglomerations of resources. Based on our experiences, we will extend RDL and its implementation to a complete version.

2.3 Overcoming Technical Difficulties

Before starting programming we had to fix the communication model (see 3.1). Therefore, we were looking for a high level communication method based on a safe protocol like TCP (*Transmission Control Protocol*). RPCs (*Remote Procedure Calls* [3, 14]) met our requirements best. However, they imply the *client-server* model and therefore block the sending process (*client*) until the receiving process (*server*) returns. Since all CCS daemons should be able to run asynchronously we could not use RPCs in this way. Indeed a slightly modified version called *non-blocking* RPCs is used. The corresponding protocol remains the same, but the sender's timeout is set to zero. This has two important advantages. First, it is assured (by the RPC protocol, based on TCP) that any message will be delivered and secondly, the sender does not wait for an acknowledgment. In case a reply is expected by the initiator a separate message is generated by the receiver. Meanwhile, the sender can work on other tasks. CCS daemons may run on any machine supporting this protocol. To ensure that byte-ordering on different architectures does not matter, every message is converted by XDR-functions (*eXternal Data Representation* [15]) into a common format (RFC11014 used by Sun, Vax, IBM, Cray, ...), sent to the receiver and converted back into the respective local format. By finishing the first stage of metacomputing evolution at the PC^2 services like NFS, NIS, e-mail etc. are available on any machine. Thus, all preconditions required by the distributed object oriented approach of CCS are fulfilled.
Nevertheless, we have kept the interface to the communication transport layer small. In this way, exchanging our Unix RPCs by other mechanisms will require only a small effort.

3 Basic Components

3.1 The Communication Model and its Implementation

The general structure of any CCS daemon is depicted in Fig. 1. Boxes indicate the modules, arrows the control-flow. Each daemon has a *start_up* module initializing local data and establishing connections to other daemons. Afterwards, a dispatcher routine is called. The *dispatcher* is responsible for accepting incoming messages, handling interrupts and sending alive messages. The *worker* module is called on arrival of a message. This module extracts the message type and either activates corresponding *receiver functions* or rejects the message.

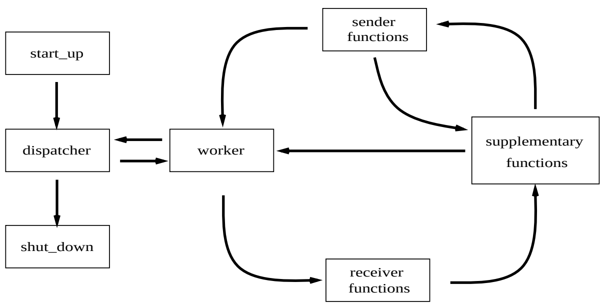

Figure 1: General daemon structure

In the former case the data are completely received and analyzed. *Supplementary functions* are used to process them. If, as a result, data must be sent to other daemons this is done by calling *sender functions*. At the end of processing an event, execution control is returned to the dispatcher.
The *shutdown module* is called just before terminating a daemon normally. It informs neighbored daemons and breaks off the connections.

The structure shown is completely independent of the communication model. The only requirement is that messages sent do not block the receiving daemons. Our implementation has the advantage that the daemons are distributed on various machines and thus watch each other. Failure of one machine running a part of the software can be detected and corresponding actions (e.g. booting the computer system or restarting the daemon) will be taken automatically.

3.2 Handling Disturbed Connections

To handle disturbed connections to the user's terminal a special mechanism has been integrated into CCS. This mechanism also permits a user to control several applications simultaneously by a single interface.

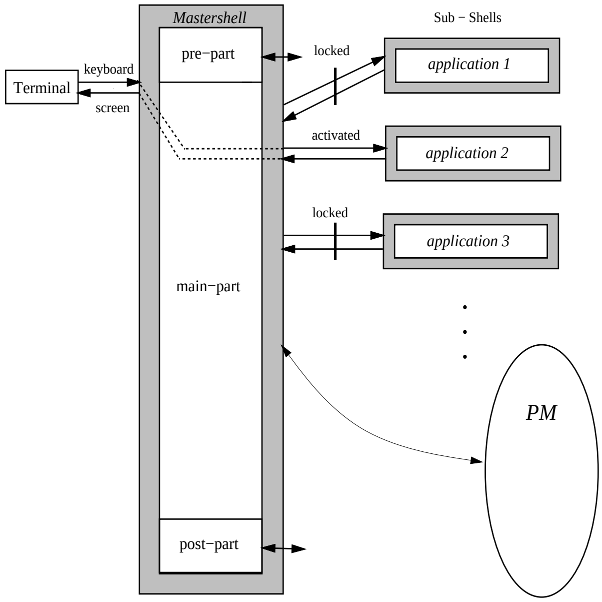

Figure 2: The Mastershell

Its interface, called *Mastershell* (MS), is *the only user-interface* to the metacomputer. It allows to start arbitrary cross- and runtime-environments, gives on-line help and supports full resource control. Because keyboard input can, at most, be assigned to a single application, each application is attached to a *Subshell* (Virtual Screen) and not directly to the user's physical screen (Fig. 2). Now a user can connect to and disconnect from the virtual screen of his application by typing MS commands. Furthermore he can leave the MS and resume working later by starting a new MS and reconnecting his old applications. In any case, the two physical terminals need not be of the same type.

Even if the virtual screen of an application is not connected, the application continues running without losing any output. In such a case, terminal output is stored in a temporary buffer. If the user reconnects to the virtual screen, this buffer is printed first. If it is completely filled or the application ends while being disconnected, the buffer content is sent to the user by e-mail.

Due to its distributed communication model, CCS is able to recognize a connection breakdown to the user's terminal automatically. Such a situation is treated in the same way as a disconnection from a virtual screen. In this sense, we are able to support unstable connection lines, which is an indispensable feature when giving access by WANs or phone lines to a metacomputer.

3.3 Integrating New Machine-Families

To enable a hardware management software to work adequately with various computer systems in a complex environment, it must be provided with a relatively high level of abstraction of existing physical resources. This is suitably attained by encapsulating each computer system by a software module, called *Machine-Manager* (MM), hiding all technical details of that machine. Thus, each machine corresponds to an object responsible for accessing and configuring the concealed computer system. Furthermore, that object is responsible for starting and terminating arbitrary scripts (jobs) on that system or at appropriate system entries. Because we have never made any assumptions about the contents of those scripts, we are able to **activate and control arbitrary user applications and programming environments.**

Within CCS the MMs are programmed as dedicated Unix daemons. Thus, including a computer system into an existing conglomeration is done by starting the corresponding

MM. Adapting a new machine family requires re-implementing the MM daemon. To be more precise, only those parts of an MM have to be re-implemented which depend on the corresponding hardware or system software. The uniform protocol between the PM (see 3.4) and different MMs makes further modifications within the CCS sources superfluous. We aim at standardizing this interface to efficiently integrate various machines into an existing configuration. By maintaining the strategy of strictly differentiating between the abstraction levels provided to the hardware management (operator-level in 2.2) and those used by the system (vendor-level in 2.2), we strive for providing a good foundation to exploit modern parallel computers today and in the future.

3.4 The Central Resource Management

The central resource management consists of two separate daemons, called *Queue-Manager* (QM) and *Port-Manager* (PM). The QM is responsible for maintaining several queues and computing efficient request schedules. Depending on priority, time, resource requirements, charging level and the application mode (batch or interactive), the queue for enqueuing a request is chosen. Once the QM decides that certain *Virtual Hardware Environments* (VHEs)[7] should be created, the resource requests are transferred to the PM.

The PM is the central control and administration unit within CCS. It provides a complete, but abstract view of the computing center environment (operator-level in 2.2) and of its utilization. Therefore the PM can provide an operator with a *unique operator-shell* of the whole metacomputer. According to some optimization heuristics, the PM selects the best suited machines for configuring a VHE. Further tasks are a central user management, the limitation of resource requests, generation of accounting data and the supervision of time limitations. To keep the implementation flexible, we have integrated an open interface calling appropriate procedures. A simple text-file based mechanism serves as a 'fall back' medium for more sophisticated implementations, e.g. the installation at the PC^2 is using an integrated SQL-Interface with remote access to a fast commercial database for gathering accounting data and controlling user-, project- and request limitations. Independently, it is possible to build automated charging tools accessing the database system. Due to the limited space, we cannot discuss the *three level optimization* policy. Within this policy the first level is implemented by an efficient request scheduling (QM) [10], the second by selecting suitable machines (PM) and the third by the system dependent configuration namely an optimized request placement onto the corresponding system architecture (MM) [5]. This policy leads to a *high level load balancing* within the metacomputer.

[7] Basically a VHE corresponds to the user-level (2.2) together with options from the user's MS commands.

4 Working with CCS

4.1 How Things Act Together

Now, we are going to present the essential daemons of CCS as depicted in Fig. 3and explain their interactions. The *Mastershell* (MS) is *the only user interface* to CCS (see 4.2). It offers a limited environment for creating VHEs and running applications in interactive or batch mode.

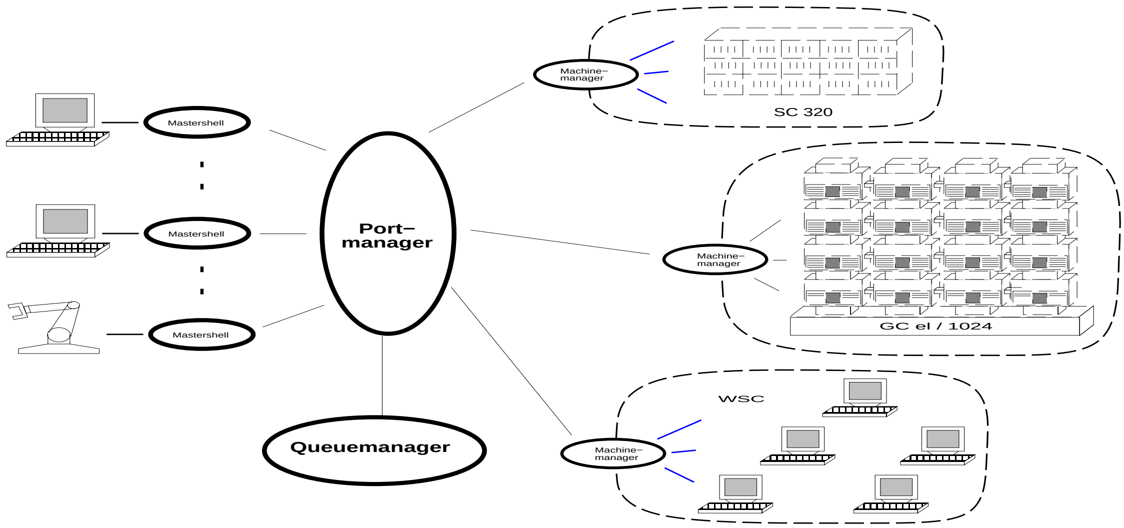

Figure 3: The essential CCS daemons

The MS can be started manually by the user or automatically by the surrounding Unix system as the user's default login shell. In either case a connection to the *Port-Manager* (PM) is established and data identifying the user (e.g. uid, hostname) are transferred (see 3.4). The PM uses this information to initiate a first authorization. If this authorization fails, the user session is aborted immediately.

Otherwise, the user has the whole command language of the MS at his disposal. Let us assume that the user requests a VHE consisting of a number of processors in a certain configuration and exclusive usage for one hour. The number of VHEs a user can handle simultaneously is bound merely by the limitations of the metacomputer and the restrictions set by the administrator or given by the operating system.
When a VHE is ordered from the MS site the PM checks the user's limitations first, i.e. the maximum number and the kind of resources allowed for the requesting user or project. If the request validation was successful, the VHE is sent to the *Queue-Manager* (QM).
The QM administers several queues. Depending on priority, time, resource requirements and the application mode (batch or interactive) a queue for enqueuing the request is chosen. If the scheduler of the QM decides that a certain VHE should be created, it is sent to the PM. The PM configures the request in cooperation with appropriate *Machine-Managers* (MMs) and supervises the time limits. In addition, the PM generates operating

and accounting data.
Now, the user is allowed to start applications or arbitrary development environments within his VHE [2, 6, 17].

4.2 Running Interactive Applications

A typical user session is depicted in Fig. 3. Our sample user created two VHEs by typing in the MS command *cappl*. The first one, named *net1*, is a request for 32 processors (1 hour). The second one, named *net2*, is a request for 64 processors (2 hours). Each VHE gets a unique ID (3875 for *net1*, 3876 for *net2*) and is sent to the PM.

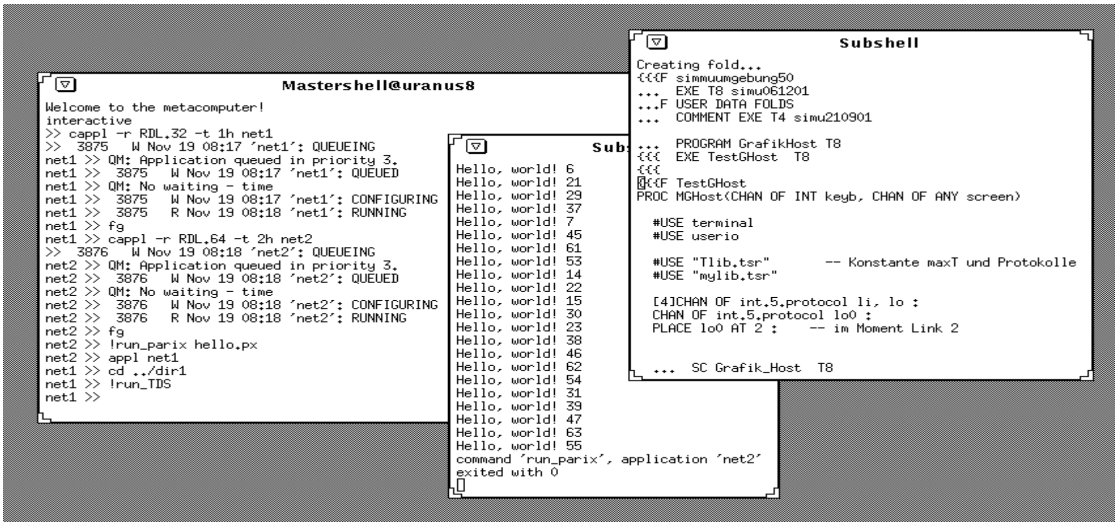

Figure 4: A snapshot of a user session

As described in Section 4.1 several states (QUEUEING, QUEUED, CONFIGURING, RUNNING) are taken. Each time the user is informed about the actual state. Our example user requested a VHE *net1*, awaited the state RUNNING and opened the virtual terminal (subshell) dedicated to this VHE by typing the *fg* (foreground) command. Then he took the same procedure for *net2*. Later on he decided to execute a Parix[8] application *hello.px* on *net2* by typing "! run_parix hello.px". The output is shown in the corresponding (left) subshell window. Concurrently he switched to *net1* and started TDS[9]. The output is depicted in the rightmost window.
As one can see, the metacomputer hardware is completely concealed. The VHEs *net1* and *net2* might be configured on the same, on different or distributed over several machines. Note that each VHE can be used for arbitrary applications or programming environments.

[8]Parix^{TM} is a trademark of the Parsytec GmbH, Germany
[9]TDS^{TM} is a trademark of the INMOS Group of Companies, U.K.

4.3 Batch-Processing

Using a VHE for batch processing is similar to the way it is created for interactive usage. In either case the standard MS commands can be used. To enable batch processing for an application one must specify a batch script-file by the *cappl*-option '-s <script>'. In general the output is sent by e-mail. If the option '-o <output-file>' is specified, however, the output is redirected to the named file. The script-file itself consists of standard mastershell commands and dialog input lines. By the following command:

```
MS:> cappl -t 2h -s my_batch.script -o my_prog.out  VHE_name
```

resources for a VHE, called *VHE_name*, are requested for a duration of at most two hours. After the VHE has been configured, *my_batch.script* is executed in batch mode and its output is redirected to *my_prog.out*. You may use our standard tools like "run_parix, run_ctools, run_otools, ..." within that script. Therefore, an example script-file would look like this:

```
! run_ctools  pre.btl
! run_parix  my_prog.px
4711
yes
```

The exclamation mark directs the following command to the location where the requested hardware is accessible, whereas the last two lines are input (stdin) for the application-program "my_prog.px". The commands to be executed are not limited at all, i.e. you may also use your own scripts and programs.

4.4 On-going Work and Sophisticated Applications

The first installation of CCS was carried out in Nov. 92 at the PC^2, integrating three types of parallel computers (plugged-in transputer boards, a freely configurable SuperCluster (SC, 320 processors) and a partitionable GigaCube system (GCel, 1024 processors)) in a 'virtual machine room'. The CCS installation at the Computing Center of the Technical University of Aachen administered several plugged-in transputer boards and a freely configurable SuperCluster (SC, 256 processors) with various special devices. Other installations are already made and further ones will follow.
We were very pleased with most of the user's reactions. However, some people found CCS slightly boring because the 'feeling' of the machines is lost.

To give an idea what might be possible in the future, we should recapitulate that every CCS daemon is able to run on a separate computer system. Setting some (local) environment variables is sufficient for the different daemons to get in contact. In current installations we are using the NFS and NIS services. To keep the network traffic as low as possible when exchanging messages between the daemons, we are sending references to the file system instead of file contents. However, there are some exceptions. Sending a resource request from a MS to the PM, the contents of the corresponding RDL- and script files are included. Thus, changing the content of these files afterwards will not interfere with running CCS activities.

Now, let us assume that whenever global services are used to access user or system data the message sent contains the data addressed, that we have two computing centers C_1,C_2 and different user locations $U_1 \ldots U_n$. The distance between all these locations might be some hundred kilometers and the only computing center known to the user side is C_1. Finally, let us assume that when an operator of C_1 specified the (virtual) metacomputer to CCS, he included the machines $C_M(1)$ of C_1 as well as the machines $C_M(2)$ of C_2 together with its locations.
A user at location U_k starting a MS and typing non CCS-specific commands will then use the computational power at location U_k. When a resource request command is typed in, it will be sent to C_1. If the required resources are available at C_1 we are done. Otherwise, if they are available at location C_2 the PM will fulfill the request together with MMs at location C_2. Afterwards, the user will get a direct connection between his MS at location U_k and his resources (at location C_2) via a high bandwidth WAN.

Notice that the user request (batch or interactive) was satisfied by a location totally unknown to the user. Both computing centers C_1 and C_2 were acting as one virtual metacomputer. In this sense, building a 'virtual machine room' and reaching stage three in metacomputer evolution is no longer an unreachable dream. However, it will probably take some more years until we can see a national metacomputer working.

References

[1] B. Bauer, F. Ramme : *A general purpose Resource Description Language,* Reihe Informatik aktuell, Hrsg: R. Grebe, M. Baumann. Parallel Datenverarbeitung mit dem Transputer, Springer-Verlag, (Berlin), 1991, pp. 68-75.

[2] N. Carriero, D. Gelernter, T. G. Mattson : *Linda in Heterogeneous Computing Environments,* Proc. Workshop on Heterogeneous Processing, March 1992, pp. 43-46

[3] J. R. Corbin : *The Art of Distributed Applications: Programming Techniques for Remote Procedure Calls,* Springer-Verlag, (New York), 1991.

[4] J. Flower, A. Kolawa, S. Bharadwaj : *The Express way to distributed processing,* Supercomputing Review, May 1991, pp. 54-55

[5] R. Funke, R. Lüling, B. Monien, F. Lücking, H. Blanke-Bohne : *An optimized reconfigurable architecture for Transputer networks,* Proc. of 25th Hawaii Int. Conf. on System Sciences (HICSS 92), Vol. 1, pp. 237-245

[6] G. A. Geist, V. S. Sunderam : *Network Based Concurrent Computing on the PVM System,* Concurrency: Practice & Experience, Vol. 4, No. 4, pp. 293-311, June 1992

[7] A. Khokhar, V. K. Prasanna, M. Shaaban, C. L. Wang : *Heterogeneous Supercomputing: Problems and Issues,* Proc. Workshop on Heterogeneous Processing, March 1992, pp. 3-12

[8] M. D. Lacasse : *DNQS: A dynamic network queuing system - user manual,* Department of Physics, McGill University, Montreal, Canada, April 1992

[9] *MPI: A Message-Passing Interface Standard,* Argonne National Laboratory, USA, Final Report V1.0, 12th April 1994

[10] F. Ramme, K. Kremer : *Scheduling a Metacomputer by an Implicit Voting System,* 3rd IEEE Int. Symp. on High-Performance Distributed Computing, San Francisco, 1994, pp. 106-113

[11] F. Ramme, T. Römke : *Resource Description Language – Language Definition,* Paderborn Center for Parallel Computing, internal report, 1992

[12] L. S. Revor : *NQS users guide,* Computing and Telecommunications Devision, Argonne National Laboratory, September 1992

[13] S. Scearce : *Network Linda builds a "virtual supercomputer" on LANs,* RiSc World, 1(3), August 1991

[14] B. Lyon : *Sun Remote Procedure Call Specification,* Sun Microsystems Inc., Mountain View (Calif.), 1984

[15] B. Lyon : *Sun External Data Representation Specification,* Sun Microsystems Inc., Mountain View (Calif.), 1984

[16] L. Mui, S. Talbott : *Guide to OSF/1 – A Technical Synopsis,* O'Reilly & Associates, Inc. 1991.

[17] Parsytec Computer Ltd. : *Parix 1.2: Software Documentation,* Aachen, March 1993

[18] L. Smarr Ch. E. Catlett : *Metacomputing,* Communications of the ACM, Vol. 35, No. 6, June 1992, pp. 45-52