TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Graduiertenkolleg:
Kooperation und Ressourcenmanagement
in verteilten Systemen

Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

Christian Röder and Georg Stellner

TUM-I9727
SFB-Bericht Nr. 342/18/97 A
Mai 97

TUM–INFO–05-I9727-100/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Design of Load Management for Parallel Applications in Networks of Heterogeneous Workstations

Christian Röder and Georg Stellner

Abstract

We present the design of a system integrated load management facility for parallel applications which execute in systems of interconnected heterogeneous workstations. Cluster based parallel computing offers an attractive alternative to the usage of dedicated parallel machines due to advances in hardware and software technologies. One of the biggest issues in such systems is the development of effective techniques for the distribution of the processes of parallel applications to multiple processors. During the last years a large amount of load management techniques have been developed both for parallel and distributed systems. But, the adoption of load management techniques from one platform to another one is difficult due to different system implications. One of the main purposes of this design document is to uncover detailed constraints which have to be considered during the decision phase of the load management cycle. On the one hand side, heterogeneity of the system and its time-sharing usage model introduce additional implications which are generally outside the control of load management systems. Nevertheless, they have major impact on the behaviour and functionality of a load manager. We classify the term heterogeneity according its impact on the load management design. On the other hand side, management overhead might outweigh the expected performance benefits. Complex regulation operations like process migration or reconfiguration of the set of execution nodes introduce management costs which have to be cosidered during the decision phase. Therefore, we introduce the new concept of cost sensitive load management. The second part covers the design of our approach. A structured modeling technique is used which enables a functional decomposition of the extended management control loop.

Contents

List of Figures

1 Load Management in Heterogeneous Environments

This document presents the design of *LoMan* — a system integrated Load Management facility for parallel applications which execute in systems of interconnected heterogeneous workstations. The main purpose of the design is to uncover detailed constraints which have to be considered during the decision phase of LoMan. Two demands are compulsory for any load management mechanism. It should distribute the load producing components to the load consuming components[1] according to a predefined performance criteria. Additionally, the mechanism should minimize its own influence on the performance of the managed system by not wasting system resources. In other words, the overhead introduced by a load management mechanism should be kept as minimal as possible.

This introduction is organized as follows. In §1.1 we introduce and motivate the usage of networks of workstations as a parallel execution platform. In §1.2 we review the standard control loop of a general purpose load management facility and briefly discuss well known problems introduced by this control loop in §1.3. Finally, we will define the focus of the project and present an overview of the remainder of this document.

1.1 Motivation

Cluster based parallel computing on networks of coupled workstations offers an attractive alternative to the usage of dedicated parallel machines. The performance of this hardware platform compares well to parallel machines if the granularity of the parallel application fits to this environment. Generally, the ratio between computation and communication has to be higher than on parallel machines due to slower communication facilities with less bandwidth and higher latency between the processing components. The rise of such networks to be utilized as a parallel hardware platform was patronized by several factors. On the one hand side, the technological paradigm shift to prefer off the shelf rather than proprietary components for building new parallel machines assimilates the architectural characteristics between both execution environments [27, 34]. The great advantage of this paradigm shift from the application developers' point of view is that applications may be developed and tested on a less expensive and widely available platform. The same application often only needs recompilation and relinking for later production runs on a parallel machine. On the other hand side, sophisticated mechanisms for process initiation, interprocess communication and synchronization were provided by the introduction of PPEs (Parallel Programming Environments). Among the most popular PPEs are p4 [3, 4], NXLib [44, 45], PVM [11, 47] and various implementations of the message passing interface standard MPI [26, 50]. A detailed overview and comparision of PPEs is presented in [43].

[1] In literature, *load producing* components are also called *resource consuming* components (e.g., [5]).

If an optimal algorithm has been developed the remaining performance improvements during the execution of applications depend on a balanced utilization of available system resources. One possibility for increasing performance is to expand a simple resource management system by a more sophisticated load management system. Whereas resource management provides the basic functionality to access and use resources of an arbitrary computing system, load management aims at improving performace with respect to some previously defined performance measures. Popular examples for performance measures are minimizing the execution time of a single application, minimizing the mean response time for a set of applications or maximizing the systems throughput. Detailed analysis of application requirements and best fitting mappings of those requirements to available resources had been the most important steps previous to any load management. The adoption of a load management facility from parallel machines to heterogeneous time shared systems is difficult due to the different system implications (see §1.4).

1.2 The Control Loop of Load Management

The central question which has to be answered by every *load management* system is: *when* to map/migrate *which* load producing entity *to which* load consuming target [6, 21]. To refine this terminology we state that the load producing entities are the tasks of a parallel application and that the load consuming targets are the workstations (also called nodes) of the execution environment.

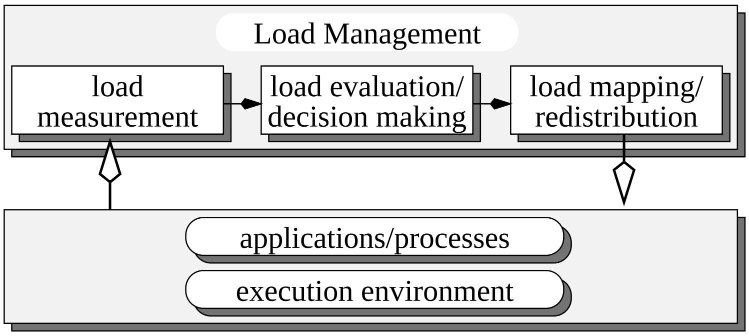


Figure 1: The Control Loop of a general purpose Load Management Facility

The general structure of a load management system may be seen as a *control loop* [21] (see Fig. 1). Parallel applications and sequential user processes execute concurrently on the execution environment. Three phases are typical for every control loop and thus, typical for a load management system. Each phase of the *standard* control loop is implemented by a corresponing *load management unit*. The *load measurement unit* determines the load at every node in the environment. The *load evaluation unit* compares the load situation of the nodes. In case of unbalanced load in the environment the *decision making unit* selects the tasks which have to be mapped on the selected target nodes. In general, two mechanisms exist to regulate an unbalanced system.

Tasks which have been submitted but are not yet executing are initially mapped to the selected target nodes during the *mapping phase*. During the *redistribution phase* already executing tasks will be migrated between two nodes (the source and the target nodes). A comprehensive collection of studies concerning load management was proposed by [38]. Detailed classification frameworks which discuss general characteristics and techniques of load management can be found in [5, 21, 35, 43].

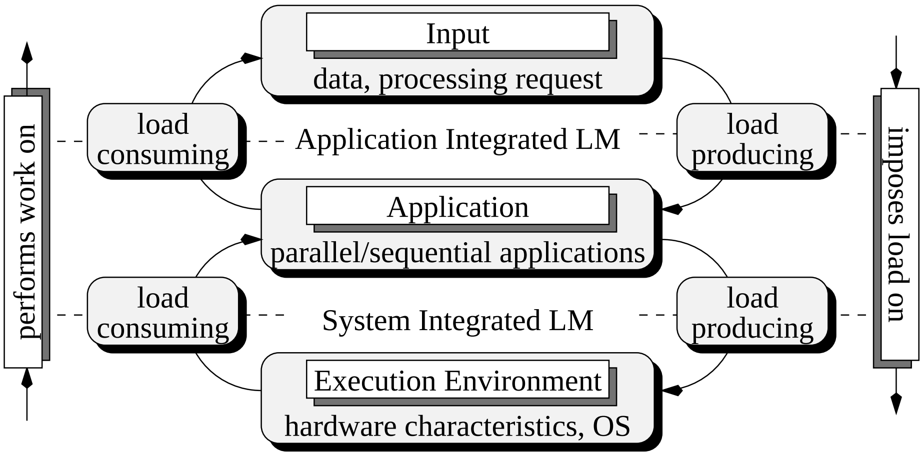


Figure 2: Application and System Integrated Load Management

There exist different approaches to integrate a load management system (see Fig. 2) into the controlled environment: *application integrated* and *system integrated* load management. We refer to an *hybrid integration* approach if different load management units are located in both the application and the system. In the former case load management is part of the application. Load is produced by the input data and processing requests of the parallel application and sequential user processes. From the viewpoint of applications, load is consumed by using and transforming those input data. Load management is performed by distributing the data and processing requests to the tasks of the applications. In the latter case the load management system is integrated into the runtime environments for the applications. Load is consumed by the nodes of the execution environment and is imposed by the executing application tasks. The regulation is performed by distributing the tasks and processes to the execution nodes in the environment. The major advantage of application integrated load management is that the application programmer has best knowledge about the structure of his application. The load producing application parts are supposed to be easier identified and the mechanisms to regulate the load on the tasks are easier to implement compared to the mechanisms in system integrated load management. On the other side, those load management techniques are generally dedicated to one application and have to be implemented again and again for every new application. This major disadvantage is extinguished when using a system integrated load management technique. Once developed and implemented, the system integrated load management mechanism can manage different kind of applications. On the other side, the best application knowledge is generally lost and not considered to conclude the management decisions. We may state as a concluding remark, that the advantage of one integration approach is the disadvantage of the other.

A hybrid integration approach seems viable at this point. Nevertheless, we choose the system integrated approach since the location of any load management unit within the applications causes loss of generality. Generality is a major design issue for our approoach. As in application integrated approaches, the application programmer has to implement a missing load management unit for every new application. Instead, we will provide mechanisms to identify the resource requirements of applications in more detail (see §2.1) and hence, will overcome this drawback.

1.3 Known Problems and Research Areas

Several well known problems arise by using control loop model of load management. The first problem is the determination of the system load. Activities which are observed by the load management system cover the submission of jobs and their execution behavior. Different jobs have different resource requirements and the utilization of resources is measured in the system or the application. In its simplest form, *load* is defined as the amount of work to be done by the processing elements. This definition of load does not consider detailed application requirements on resources of the processing elements. Although, a more detailed definition and representation of the term load is needed it is not part of this document. We just state that not only the current system utilization but also the application resource requirements are used to model the load and to determine the loading situation. For a detailed discussion we refer to [33, 32].

The second problem is to stabilize the control loop to avoid that a load producing entity which has been mapped/migrated is not touched right afterwards. Otherwise, it may happen that the entity never executes to produce application results but only moves between nodes. A history of management operations, i.e., mapping and redistribution operations, could be used during the decision phase to prevent this kind of load or *processor thrashing* [39]. In non-dedicated environments (see §2.2) such a history might also be used to prevent that a dynamically changing machine configuration will be reconfigured at any possible time. In this case a comparable problem to processor thrashing might occur. Once a node is selected and configured to execute application tasks it should not be replaced by a new node right afterwards. Otherwise, it may happen that nodes are only exchanged but never execute application tasks. We will call this situation *node thrashing*.

Finally, the activities of a load management system need system resources and hence, impose load to the execution environment. One of the major demands to any load management technique is to reduce the management overhead. Management overhead occurs within every management phase but have only partially be considered in the literature. For example, [18] considers the complexity of different workload descriptions on the performance of heuristic load management techniques. It has been shown that simple load indices outperform more complex workload descriptions. These are easier to measure and introduce less measurement overhead on the system. But management overhead also occurs especially for complex regulation operations, like migration of tasks between nodes. The costs of such operations might outweigh the expected performance benefits. Therefore, we will consider the costs of the regulation activities (see §2.3).

1.4 Organization and Contents

The reminder of this document is organized as follows. The design of LoMan is preceded by an analysis of the implications imposed by the system and the application (see §1.4). These implications will be called *external constraints* since they are not part of the load management mechanism itself but rather have to be faced and considered. The goal of this analysis is to define a model for the execution environment — a system of interconnected heterogeneous workstations. §2.2 describes details of the execution environment. §2.1 presents a short overview of the structure and dynamic execution behavior of parallel applications. We will derive the objects to be managed by LoMan and the necessary functionalities to do so. Additionally, we have to analyze the *internal constraints*, i.e., the load management overhead imposed to the execution environment. From those two types we will define a cost sensitive control loop in §2.3.

We will argue that a comprehensive modeling approach is required to cover the complex aspects of load management. In §2.4 we will review the basic concepts of the chosen modeling technique which is based on functional decomposition. The design of LoMan will focus on its decision component. §3.4 presents a general view on the functionality of LoMan and serves as starting point for its the top-down design. §4.4 further refines the framework by uncovering more details to be considered.

2 Analysis of Constraints

The development of LoMan will be preceded by a detailed analysis of the system implications. Firstly, we will discuss the application environment in §2.1. We focus our discussion on the dynamic execution behavior of a single parallel application. In §2.2 we discuss the design considerations which are guided by the implications of the execution environment. From these discussions we derive the costs introduced by the management operations in §2.3. Additionally, we argue that the structure of LoMan and the complexity of the managed objects requires a comprehensive design approach.

2.1 The Application Space

Talking about job parallelism is multifaceted. It depends on the space of the observed execution environment and the viewpoint from which the observation is taken. The jobs share system resources for which they compete. Hence from the viewpoint of a single job the other jobs introduce a *background load* [2, 41]. In the context of load management this background load is not under the control of the management system. In Fig. 3 we refine our observation space.

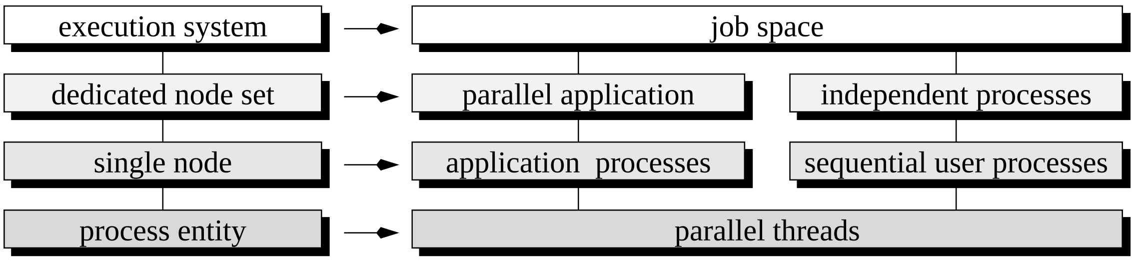


Figure 3: Levels of parallelism of executing jobs

At the top level we observe the execution system (for a detailed discussion see §2.2) and see a vast collection of jobs executing in parallel. For our purposes, we subdivide this *job space* into two major classes: a global set of parallel applications and a global set of independent processes. At the next level, a parallel application executes on a selected and dedicated set of nodes. From the viewpoint of a single parallel application a subset of independent processes (and possibly parts of other parallel applications) execute on the same set of nodes. These other execution entities are viewed as the background load. We end up with parallelism and concurrency on a single node. On each single node, application tasks and sequential user processes execute quasi parallel[2]. For a detailed discussion of the time-shared system usage see also §2.2.3. Finally, each such process, either an application task or a regular user process, may contain several execution threads.

[2]On workstations with more than one CPU they even may execute in parallel.

LoMan manages the tasks of a single parallel application but also considers background load. In the following, we focus on the possibly dynamic structure and relationship of a parallel application and its constituting tasks. A *parallel application* may be seen as a set of cooperating *tasks*. In contrast to serial processes they communicate to exchange intermediate results in order to compute the final application results. We assume that explicit *message passing* is used for communication and do not consider shared memory communication at this point. Internally, we do not consider fine grain parallelism, i.e., the tasks are not inherently parallel by exploiting light weight processes (threads).

Therefore, we formally define a parallel application $\mathcal{A}$ to be a set of tasks t_i:

$$\mathcal{A} = \{t_0, t_1, t_2, \cdots, t_a\}$$

Several characteristics of the parallel application and its constituting tasks have to be considered during the design of LoMan:

1. Initially, we assume that during the startup phase of the application its tasks are created by a single parent task t_0. A task t_i which has been created and is ready to run will be annoted with t_i^r, whereas a task which is already mapped and executes on a node is annoted with t_i^e. The number of tasks may change during the execution time of the application. Each task may create children tasks due to algorithm demands or due to the availability of additional resources. Branch and bound algorithms are popular examples for this behavior. The classical load management cycle distinguishes between two phases: the initial task mapping of T^r (i.e., the set of ready to run tasks) and the dynamically task redistribution of T^e (i.e., the set of already executing tasks). In our case, the two phases are merging since the number of tasks might dynamically change. Hence, the management decisions might additionally have to consider the mapping of the pair (T^r, T^e) to the nodes.

2. The tasks may have different resource requirements. For example, the tasks of a parallel application might be grouped according to the roles they play within the application. A subset of the tasks might be responsible to compute the results while the another subset is responsible to read initial data from disk and write final results to disk. Hence, the second group of tasks should be located on nodes which have access to local disks. Performing I/O to a remote disk would cost additional communication resources. Parallel client-server applications are typical examples for this behavior. From the viewpoint of LoMan the resource requirements of the tasks have to be observed and considered in more detail.

3. Different versions of the tasks executable code may exist (see §2.2.1). The load manager has to evaluate the node-task architecture fitness. At application startup time, LoMan depicts the nodes of a fitting architecture class. This choice remains fixed during the load balancing phase since task migration is currently only possible between architectural homogeneous nodes. In contrast to this, several load management systems exploit load redistribution by moving data items between the tasks.

2.2 The Execution Environment

The *execution environment* comprises a set of interconnected heterogeneous workstations (also called *nodes*). Several users may exploit the processing power of the execution environment concurrently by time-sharing its resources. We will now discuss the system implications *heterogeneity* (see §2.2.1) and the *time-sharing* usage model (see §2.2.3). From this discussion we will derive the requirements and constraints which have to be considered by LoMan.

Formally, we define the distributed execution environment $\mathcal{E}$ to be a pair

$$\mathcal{E} = (\mathcal{N}, \mathcal{C}),$$

with $\mathcal{N}$ being the set of nodes ($\mathcal{N} = \{n_1, n_2, \cdots, n_i\}$) and $\mathcal{C}$ being the set of different communication links ($\mathcal{C} = \{c_1, c_2, \cdots, c_j\}$). The next section §2.2.1 refines this rough definition.

2.2.1 Heterogeneity

One of the major difficulties to provide transparency in distributed systems is the *heterogeneity* of its constituting elements [12, 34]. Heterogeneity is expressed on several levels of the hardware and software components which constitute the system. Talking about heterogeneity is often misleading in the context of distributed systems since the term covers several ambiguous meanings. In the context of system integrated load management, heterogeneity has two different obvious impacts. On the one hand side, if two nodes have different architecture the mapping and migration of incompatible code and data is prevented. On the other side, the hardware configuration of the nodes (e.g., CPU clock rate, size of main memory, local disk) determine their performance indices. Therefore, we introduce a coarse grain hierarchical classification of heterogeneity to refine and precise our discussion (see Fig. 4).

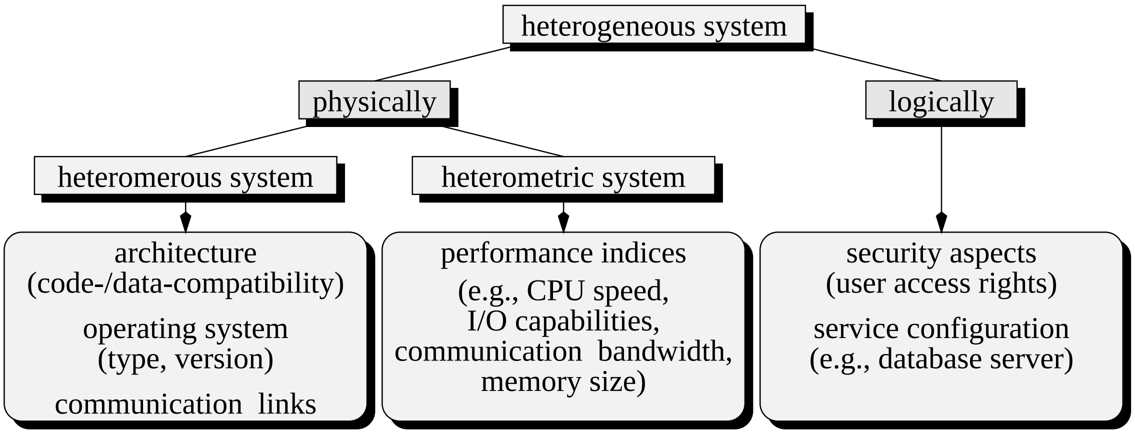


Figure 4: Heterogeneity of the execution environment

Dictionaries [13, 19, 40, 28] define the term *heterogeneous* as *consisting of parts or members that are very different from one another.* For our purposes, this definition is only useful to talk about the nodes in the distributed environment more generally and to state that we are concerned with different aspects of heterogeneity. We distinguish two facets of heterogeneity: *physical* heterogeneity derives from the hardware and software components used, whereas the term *logical* heterogeneity derives from the usage policy and the roles each node plays in the system.

Logical heterogeneity is of importance for the load management design if security aspects have to be considered. Additionally, it is one source of background load which has to be considered. For example, the load management system should not select a node which is configured as a database server or which is reserved to a certain other service. In this case, we assume that the services are frequently used and need large amount of node resources. Instead, other workstations should be selected even if their performance or their current loading situation favor the specialized nodes. Additionally, system security aspects might restrict the usage of some nodes for different users. From the perspective of a system integrated load management mechanism it might even be necessary to exclude some nodes for one group of users while they have to be granted to other users.

Physical heterogeneity is subdivided into two further classes: we distinguish *heteromerous* systems which *have or consist of parts that differ in quality or the like* [40] and *heterometric* systems which *have or consist of parts that are marked with different quantities* [28]. It is crucial for our purposes that quality and quantity of system resources influence each other with respect to some of the properties of the resources. Nevertheless, we introduce these terms to focus on different aspects of heterogeneity. The first class determines the migration space of the entities to be managed, whereas the second definition covers the different performance characteristics of the nodes. The following list will detail those aspects with respect to their influence on the load management design.

1. The heteromerous property of the system influences the selection of target nodes according the tasks executable architecture characteristics. The *architecture of the nodes* determines the data representation, i.e., instruction sets, word sizes and byte ordering of the computed data. The execution environment may comprise a number of different architecture classes. Additionally, the types and versions of the *operating systems* which may be found on the nodes in the execution environment might differ. The operating systems' local scheduling and memory management policies determine when the mapped tasks will have access to the resources of the nodes. Different local policies will have different side effects on the runtime of the application tasks. Furthermore, different versions and releases might prevent process migration even within a single type of operating system. We do not consider process migration facilities which depend on costly preparations [48] or at which residual dependencies [30] remain. We assume that task migration completely hides the task occurrence on the source node [43].

We group the nodes between which processes are migratable into a *homomerous subsystem*. Hence, several disjoint homomerous subsystems N^i might exist in the heterogeneous execution environment:

$$\mathcal{N} = \{N^1, N^2, \cdots, N^n\},$$

with $N^i \cap N^j = \emptyset$, for $i \neq j$ and $1 \leq i, j \leq n$.

2. The hardware performance of the nodes in the system is determined by the *performance characteristics* of their constituting hardware resources (i.e., processing speed, memory bandwidth, I/O bandwidth). The performance indices of the nodes influence the computation of the load in the system. For example, a popular load index is the number of processes in the node's ready queue [10, 18]. This load index has to be scaled to a comparable value if two nodes have different processing speeds [1, 51]. Recently, the memory bandwidth became more and more important to determine the load of a node load since parallel applications typically utilize huge amount of memory [35, 36]. The memory management unit of the local operating system will swap the pages as soon as the local memory is fully utilized and more memory pages are needed. The swapping of memory pages will slow down the performance of the node. As already stated in §2.1, the resource usage of I/O operations significantly depends on the availability of a local disk.

   The heterometric nodes of a single homomerous subsets N^i may be further subdivided into *homometric* classes, although it is difficult to group the nodes according to their relative performance indices. The most popular property to group the nodes is their processing speed as determined by some artificial benchmarks. In the literature this property is also called *architecture factor* [51, 37] (in our sense this terminology is quite misleading). Therefore, we state that N^i is further subdivided into:

$$N^i = \{N^i_1, N^i_2, \cdots, N^i_\nu\},$$

   with N^i_j being the set of homometric nodes.

   Different types of *communication links* are possible within the same workstation cluster and hence, the performance (e.g., latency and bandwidth) of the communication links has to be considered. It is important for communication intensive parallel applications to select target nodes which are interconnected by fast communication links. Thus, the communication performance has to be considered even if the interconnection topology of the nodes is not exploitable (as it is done by many load management system on dedicated parallel machines).

   Similarly to the above notion, we refine the set of communication links $\mathcal{C}$ to be subdivided into performance classes:

$$\mathcal{C} = \{C_1, C_2, \cdots, C_c\}$$

> We state, that the same node might have access to different communication subsystems. The advantage of this approach is that for later enhancements of LoMan the communication via virtual shared memory is easily adoptable.

In summary, this leads to a refinement of our definition of the execution environment proposed at the beginning of this section. We group the nodes in the heterogeneous execution environment into heteromerous and heterometric subsets. With those definitions in mind, we state that current load management systems for parallel applications in heterogeneous systems are limited to heterometric systems [53, 20, 52, 7].

2.2.2 Example Configuration

Fig. 5 shows a cluster of 9 heterogeneous nodes $\mathcal{N} = \{n_1, n_2, \cdots, n_9\}$. We subdivide this set of nodes according to their physical heterogeneity (i.e., their heteromerous and heterometric characteristics).

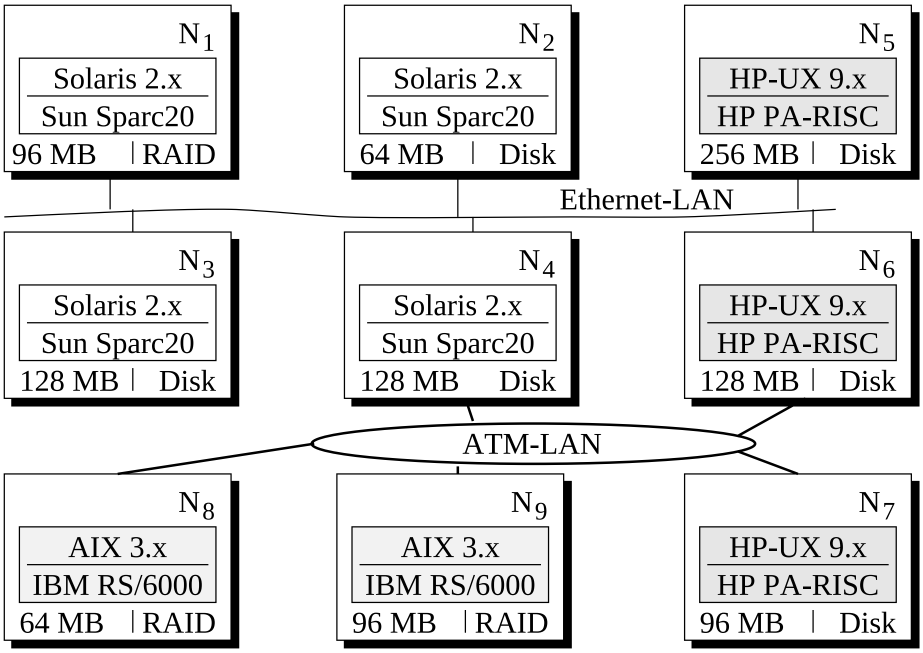


Figure 5: Example configuration of an execution environment

The nodes are subdivided into three architecture classes and two communication classes. Obviously, the architecture classes form non-overlapping homomerous sets of nodes: $N^1 = \{n_1, \cdots, n_4\}$ forms the SunSparc20 homomerous subset, $N^2 = \{n_5, \cdots, n_7\}$ forms the HP PA-RISC homomerous subset and $N^3 = \{n_8, n_9\}$ is the IBM RS/6000 homomerous subset. The communication subsets are overlapping, since there exist nodes which are able to communicate over several communication links: $\mathcal{E}_1 = (\{n_1, \cdots, n_6\}, C_1)$ communicate via the Ethernet link, while $\mathcal{E}_2 = (\{n_4, n_6, \cdots, n_9\}, C_2)$ communicate via the ATM link. In this case, $\mathcal{C}_{overlap} = \{n_4, n_6\}$, i.e., the nodes N_4 and N_6 may exchange messages via both links.

2.2.3 Multiuser Mode

Fig. 6 shows an additional obvious difference between dedicated parallel machines and the cluster of heterogeneous workstations [17, 49]. This difference is mainly concerned with the facets of the *multiuser mode*. The former machines are mainly used in space-sharing mode, i.e., parallel applications executes on non-overlapping compute partitions [8]. At any time only one application executes within a single partition which was granted before its startup time. Each partition provides an exclusive single system image to the application. Recently, new scheduling approaches for parallel machines overcome those fixed partition sizes [31].

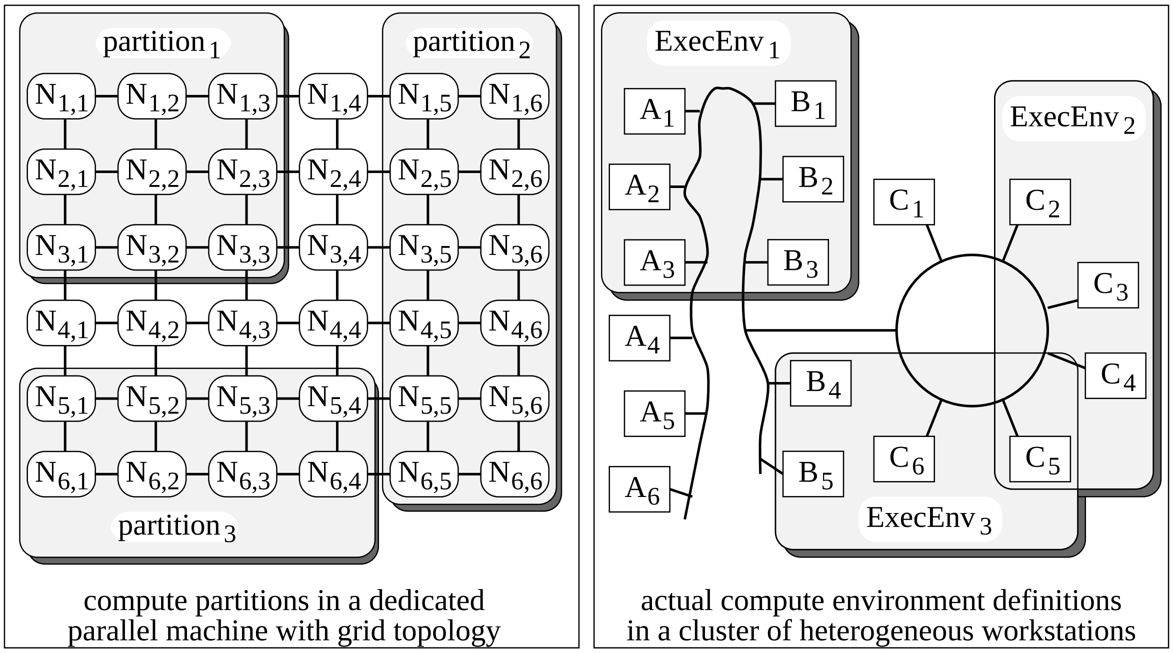


Figure 6: Time-Sharing Usage Model

In contrast to space-sharing, coupled workstations operate in time-sharing mode. Hence, overlapping actual compute environment definitions are likely to happen if several independent parallel applications execute in the system. The promising slogan *resource sharing* [12] serves as guarantee that the resources are always busy as long as work has to be performed on them. They are not blocked or occupied by a waiting job. Waiting situations occurs in both job classes. On the one hand side, concurrency is managed by the local scheduling strategies. As soon as one job waits for an external event the scheduler will remove this job from the node's CPU and grant it to another ready-to-run job. On the other hand side, in case of parallel applications it is likely to happen that tasks wait for messages (i.e., intermediate results or synchronization primitives) from other tasks. Hence, waiting situation are also typical for parallel applications.

The availability of constant node performance in the system can not be guaranteed since users log into and off the nodes at any time [34]. During their interactive sessions they occupy the system resources with different degrees of utilization. Two different kinds of users may be distinguished from the node's viewpoint [41]. Users which are physically logged into a node are called *primary users*, whereas *secondary users* log into that node from another remote node. Primary users should not be disturbed during interactive sessions. This concept of *workstation ownership* is contradictionary to the concept of resource sharing. We do not demand that the tasks of a parallel application have to be moved away from a node as soon as interactive users log into that node. We rather consider the user behavior in more detail to prevent a waste of system resources. From the viewpoint of a parallel application we additionally introduce the term *external user*. Those users are not owner of the currently executing parallel application but are the owner of other processes executing in the system. It is not necessary that the owners of parallel applications or sequential processes are physically logged into the system at the time those are executing. Generally, those jobs (i.e., a parallel application or sequential user process) are called *batch jobs*. Batch jobs need no interaction with the users.

The time-sharing usage model introduces additional implications which have to be considered by the load management system:

1. Workstation ownership constraints implies that interactive usage should not be disturbed by executing a parallel application,

2. but to keep the promising concept of resource sharing the demands of external users have to be considered in more detail.

3. The set of nodes on which the parallel application executes could be changed if the primary user fully utilizes the resources.

We conclude, that the set of nodes on which tasks are distributed is a subset of those nodes which will be observed, i.e., on which load is measured and interactive usage is observed. The load management system has to distinguish between two sets of nodes: the *actual execution environment* denotes the set of nodes on which application tasks currently execute, whereas the *possible execution environment* denotes the set of nodes which is observed.

2.2.4 Requirements in case of multiple Parallel Applications

LoMan is designed to manage the mapping between a single parallel application and a reconfigurable set of workstations. Efficiency reasons encourage this approach since decisions are concluded locally with respect to the resource requirements of this application. Information needed to achieve the performance goals are kept as local as possible within the management system. Thus, costly information exchange operations between different components are avoided. We will briefly discuss the effects on the design of LoMan if several parallel applications are executed concurrently in the system.

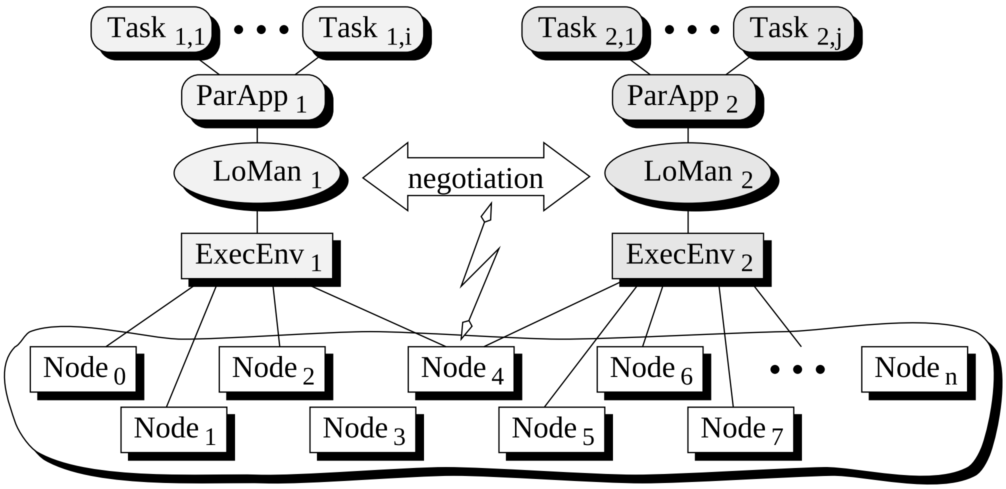


Figure 7: Negotiation between multiple LoMan systems

In §2.1 we stated that from the viewpoint of one parallel application another one is only seen as background load. Assume, each parallel application is managed by its own load management system, either LoMan or any other load management system. Each of the separated load management systems independently conclude decisions without knowledge of the decision results of other management systems. Locally, the decisions are adequate to meet the expected performance improvement for a single application. Globally, the decisions might be contrary to the expected performance benefits. For example, a lightly loaded node is independently selected as target node. The management systems might map some of their associated application tasks onto this node. Hence, the load on the node increases and might exceed a predefined load threshold. These tasks might be immediately migrated to other nodes as soon as the management systems evaluate the load situation on the node. Similarly, newly selected target nodes might get overloaded if decisions are concluded independently again. This scenario is similar to node thrashing (see §1.3), except it is not caused by an instable control loop. Hence, mechanisms have to be considered and provided to prevent contrary node selections.

In Fig. 7 we term these mechanisms with *negotiation*. Negotiation is performed by the load management systems LoMan_1 and LoMan_2 (although, more than two load management systems are possible). The associated execution environments ExecEnv_1 and ExecEnv_2 share the workstation Node_4. Hence, this workstation is a possible candidate for overloading. Nodes in the system can not be arbitrarily selected if several load management systems use a negotiation protocol. Thus, it is guaranteed that the nodes are selected in coordinated manner. Additionally, unnecessary management operations are avoided. Beside negotiation protocols, one can also imagine a global resource management system which is responsible to grant or refuse nodes [24] to a single load management system. In this document we are not interested in the details of negotiation or similar techniques. We rather state that LoMan will provide management units which may be easily extended to incorporate such techniques.

2.3 A Model for Cost Sensitive Load Management

We summarize the above system implications by listing the objects which have to be managed during the control loop by LoMan. These external constraints define the management operations and directly lead to the internal management constraints. From this point of view, we define an extended control loop for LoMan in §2.3.2.

2.3.1 Costs of Management Operations

Load management introduces additional *management costs* which will be considered during the decision making process. We will define these costs by summarizing the objects to be managed by LoMan and listing its necessary management operations. The managed objects and their relevant management operations are depicted in Fig. 8:

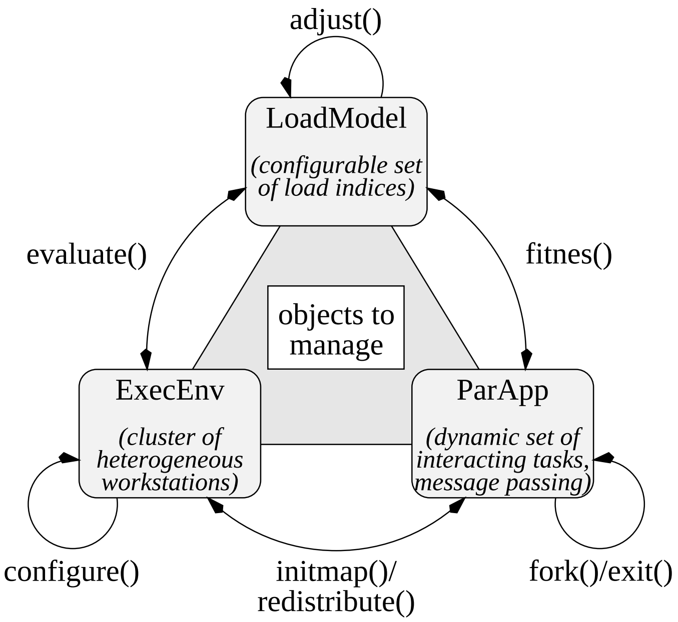


Figure 8: Objects to be managed by LoMan

1. The *ExecEnv* — *<u>Exe</u>*cution *<u>Env</u>*ironment — is the set of all nodes in the environment. LoMan is responsible to select and configure the nodes on which the application tasks might execute. The configuration of actual target nodes and setting up the necessary runtime environment introduces *configuration costs*. At startup time of the application it is important to initially select a "stable" machine configuration, although this is difficult in non-dedicated environments. An initially "wrong" selection of nodes might outweigh the expected performance benefits since it is likely to happen that nodes have to be added and released to the actual set of target nodes. Hence, the reconfiguration of an already existing environment will be expensive – although it might be compulsory due to the interactive session of an external user.

2. The *ParApp* — *Par*allel *App*lication — is the set of cooperating tasks. Both phases of a load management system introduce additional overhead, i.e., load mapping by task placement introduces *mapping costs* and load redistribution by task migration introduces *migration costs*. In general, the operation of task migration is more complex and thus, more expensive than task mapping due to its protocol overhead. Furthermore, it has been shown [10, 9] that an initially "good"[3] mapping will prevent the load management system to redistribute the load too often. As stated above, this argument is only of limited benefit if unpredictable background load has to be considered. Nevertheless, we try to fulfill static resource requirements. The process of mapping application tasks near to resources they actually need is often called *affinity scheduling* [46]. For example, I/0 intensive application tasks should be mapped onto nodes which have access to local disks. Thus, stored data has not to be transferred via the communication links.

3. The adaptive load model *LoadModel* is a configurable set of load indices. Measurement will introduce increasingly more overhead to the control loop as soon as more indices have to be measured. Additionally, the evaluation will be increasingly more complex. This overhead will be called *load modeling costs* [18, 32].

All three types of the operation costs will be considered by LoMan.

2.3.2 The Extended Control Loop of LoMan

Current load management systems base their decisions exclusively on the actual load situation of the system. For a load management mechanism which considers its own overhead, i.e., the management costs, we will introduce an *extended control loop*. The main difference to the standard control loop (see §1.2) is illustrated in Fig. 9.

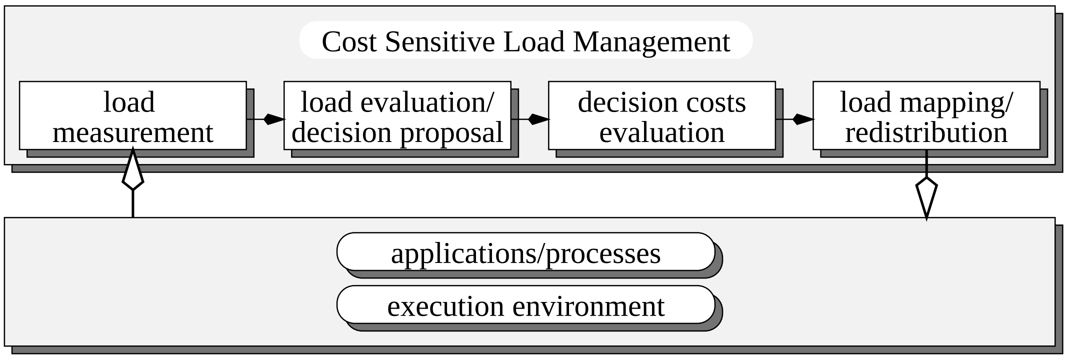


Figure 9: Extended Control Loop of a Cost Sensitive Load Management Facility

[3]In [21] it has been shown, that the problem of computing optimal task mappings is NP-complete. Therefore, only suboptimal mappings which can be found in reasonable short time are of practical interest. Heuristics and load thresholds are the most common techniques used in practice.

If necessary and possible, the decision component proposes several regulation alternatives to fulfill the performance criteria. The *cost evaluation unit* either picks the cheapest decision alternative or rejects all alternatives if the expected performance benefits are not worth an expensive operation. Thus, unnecessary management operations are prevented.

2.4 Summary and Design Considerations

To summarize the above topics §2.1 – §2.3, we list the most important requirements for the design of LoMan.

1. LoMan has to merge the two phases of initial task placement and dynamic task redistribution due to possible dynamic task creation during the runtime of the application (see §2.1).

2. Heterogeneity has to be considered in more detail. The heteromerous property of the system might exclude some nodes to be selected as possible targets. The heterometric property forces LoMan to scale load values to a possibly common base (see §2.2.1).

3. LoMan has to distinguish between the subset of nodes on which load is measured and the subset of nodes on which tasks currently execute (see §2.2.3). The presence or absence of interactive users forces or enables LoMan to dynamically reconfigure the actual execution environment.

4. LoMan is designed to manage a single parallel application in a possibly changing machine configuration. LoMan has to be extendible since mechanisms for coordination are needed if several parallel applications and hence, several LoMans execute concurrently (see §2.2.4).

5. LoMan has to judge management decisions according the overhead and resource requirements, i.e., the management costs, they introduce to the system (see §2.3.2). Management operations are rejected if they do not meet the expected performance benefits.

6. The resource requirements of application tasks have to be considered in more detail. Affinity scheduling is used during the mapping phase to fulfill static resource requirements (see §2.3.1).

At this point we stress the need for a structured modeling approach during the design phase of the system integrated load management mechanism. We use SADT to cover the complexity of the load management system and to cover the implications of parallel applications and the execution environment in a consistent manner. The next section §2.4 introduces the basic mechanism of the SADT methodology. In §3.4 we start with modeling the load management system. We continue with a more detailed description of the activities in §4.4.

3 Introduction to SADT

In this section we will briefly review the methodology and terminology used by SADT™ [25] — Structured Analysis and Design Techniques[4] — which will guide the modeling of the load management system.

3.1 Systems and Models

SADT is a complete methodology for developing system descriptions, centered on the concepts of system modeling. A *SADT model* (short: model) employs both natural and graphics language to convey meaning about a particular system. The graphics language comes from SADT. A SADT model that focuses on system activities is called *activity model*, and one that focuses on the system things (which is a more general expression than data) is called *data model*. Activity models present system activities in a successively detailed manner, and they define the relationship among those activities through the transformed things of the system.

SADT models are developed for a particular motive. That reason, called the *purpose of an SADT model*, is derived from the way SADT formally defines a model:

> M is a model of a system S if M can be used to answer questions about S with an accuracy of A.

The purpose of a model is, by definition, answering a set of questions. Commonly, the question set for a model is developed very early in a SADT modeling effort, and is usually summarized by one or a few sentences. These sentences become the stated purpose of the model, and the question set remains as detailed backup to that sentence. When the model is complete, the questions listed will be answered by the information contained in the model.

The system itself is called the *subject* of the SADT model. SADT stresses the need for precisely defining the boundary of a system. Therefore, a SADT model always bounds its subject. Since the quality of a system description is dramatically reduced if it remains unfocused, SADT requires that a model be developed from *one and only one particular* perspective. This perspective is called the *viewpoint* of the model. A viewpoint is best thought of as a place, person or thing one can stand in to view the system in operation. From this fixed perspective, a consistent system description can be developed. The model will not drift around the topic and incorrectly mix together unrelated descriptions.

The subject defines what to include in, and exclude from, the model. The purpose becomes the criterion for determining when to stop the model. The final result of this process is a collection of carefully coordinated descriptions, starting from a very high

[4]SADT is a trademark by SofTech, Inc.

level description of the entire system and ending with detailed descriptions of system operation. Each of these coordinated descriptions is called a *SADT diagram*. SADT models collect and organize diagrams into a hierarchical structure, in which diagrams at the top of the model are less detailed than those at the bottom. In other words, a SADT model can be thought of as a tree-shaped collection of diagrams.

3.2 Diagram Syntax and Usage

The diagram is the basic work unit in making a model. Diagrams have their own syntax rules which are separate from the syntax rules for models. A diagram is named with a *title* and has standard *identification information* (e.g., author, date of creation or revision, status). All identification information is placed at the top of the diagram. The diagram contains *boxes* and *arrows*. The boxes represent activities of the system being modeled. Arrows connect boxes and represent interfaces or interconnections between the boxes.

Boxes are named with verbs or verb phrases as they represent the active parts of the system. For readability, SADT requires that not less than three and not more than six boxes appear in one diagram. Thus, diagrams and models are comprehensible.

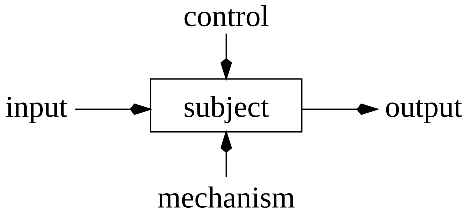


Figure 10: General notion of SADT boxes

Unlike all the other graphic-based structured analysis methods, each side of a SADT box has a specific and special meaning. As shown in Fig. 10 the left side of a box is reserved for *input information*, the right side is reserved for *output information*, the top side is reserved for *control information*, and the bottom side is reserved for used *mechanisms*. This notation represents certain principles: inputs are transformed into outputs, controls constrain or dictate under what conditions transformations occur, and mechanisms describe how the function is accomplished. In comprehensive words, we say:

> Under *control*, *input* is transformed into *output* using the *mechanism*.

SADT boxes are never placed randomly on a *diagram form*. Instead, they are placed according to their relative order of importance as judged by the author. SADT calls this relative ordering *dominance*. Dominance can be thought of as the influence one box has over other boxes in a diagram. Usually, the most dominant box is placed in the upper left-hand corner of the diagram, and the least dominant box is placed in the lower right-hand corner. In SADT, boxes must be numbered. Box *numbers* provide unique identifiers for system activities and automatically organize those activities into a model hierarchy.

The arrows of a SADT activity diagram represent a collection of things and thus, they are labeled with nouns or noun phrases adjacent to some part of the arrows line. *Input arrows* represent those things used and transformed by activities. *Control arrows* represent things that constrain activities. Usually, control arrows stand for information that directs what activities to perform. *Output arrows* represent those things into which inputs are transformed. *Mechanism arrows* represent, at least in part, how activities (i.e., the functions of the system) are implemented. Thus, mechanisms represent the physical aspects of an activity (e.g., storage places, people, organizations, devices).

Good system descriptions contain a variety of things in order to adequately explain how a system works, both in detail and with its environment. The SADT graphics language distinguishes between the things transformed by a system and things that govern how the transformations are done. SADT calls these *inputs* and *controls*, respectively. SADT arrows can be thought of as cables that organize and carry this variety of things. Sometimes an activity splits an arrow into its components, just as a prism splits white light into its colors.

To summarize these concepts, we state that SADT diagrams are neither flowcharts nor just dataflow diagrams. They are constraint diagrams that describe both input-to-output transformation and the constraint rules on those transforms. In this way, arrows document the interfaces between system activities and between the system and its environment. As shown in Fig. 11 only five kinds of box interconnections are required by SADT to describe these relationships:

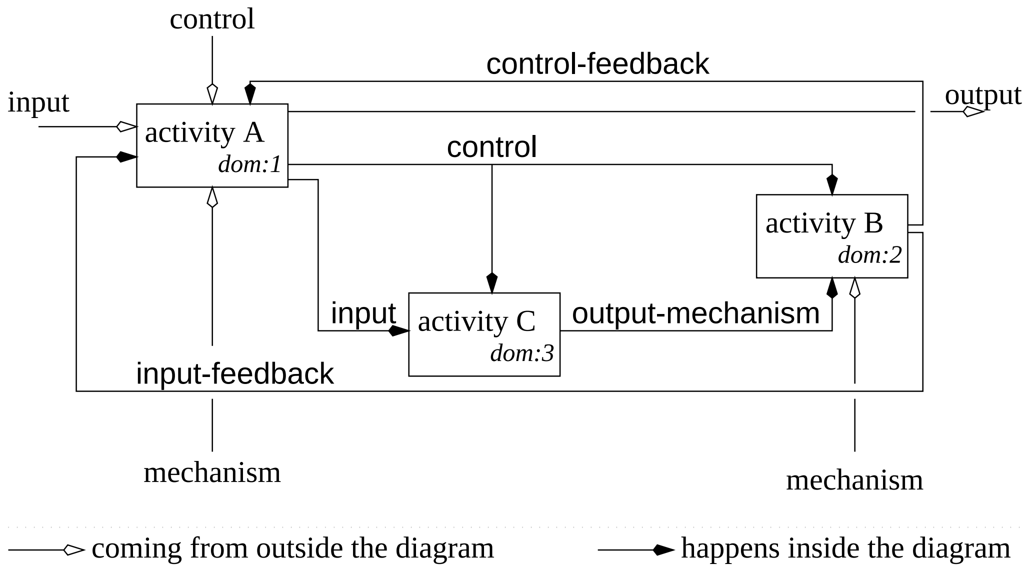


Figure 11: SADT arrows describing activity relationships

1. *control*: The output of one box directly constrains a box of less dominance.

2. *input*: The output of one box becomes the input of a less dominant box.

3. *control-feedback*: The output of one box constrains a box of greater dominance.

4. *input-feedback*: The output of one box becomes the input of a box of greater dominance.

5. *output-mechanism*: They represent situations in which the output of one activity becomes the means by which another activity gets done.

Control and input connections are the simplest, because they represent direct influences that are intuitively obvious and very common. Control-feedback and input-feedback connections are more complicated, because they represent iteration or recursion. That is, the outputs of one activity influence the future operations of other functions that subsequently influence the original activity. They both are drawn as *backward-looping* arrows. Output-mechanism connections are interesting and somewhat unusual. They may be used to denote sequencing relationship (e.g., preparations must be complete before work can start). Therefore, output-mechanism often deal with resource allocation issues (e.g., required tools, trained people, physical space, equipment, funding, materials).

Control-feedback has a more profound affect on system operation than *dataflow-feedback*, since this kind happens when the output of two functions constrain each other (the classic chicken-and-egg scenario is an example of this control-feedback). Therefore, control-feedbacks are considerable stronger than dataflow-feedbacks in the way they bind functions together.

In general, an arrow represents a collection of things, rather than one thing by itself. Because of this, they may have multiple tails (sources) and multiple heads (destinations). Arrows have the ability to *branch* apart and *join* together in arbitrarily complex ways. Arrow branches are always labeled before their branch and might be labeled after the branch point if the things are only part of the previous collection of things. Arrow joins are always labeled after the aggregation to describe the resulting collection of things. *Tunneled* arrows may be used to hide the source or destination of this arrow. In diagrams they are graphically shown with braces at their heads or tails.

SADT uses a diagram configuration control scheme to manage different *versions* of reworked diagrams. The scheme is build upon *C-Numbers* (i.e., *chronological numbers*) to distinguish different versions of the same diagram. A C-Number is constructed of the author's initials and unique sequence numbers. A log of C-Numbers is kept to identify the versions.

3.3 Model Syntax and Usage

Several diagrams are required to adequately describe a system. A SADT model is built by collecting and connecting diagrams. SADT models have syntax rules distinct from, but complementary to, the syntax rules for diagrams. The syntax rules define the boundary of a model, connect diagrams into a unified whole and ensure "plug-to-plug" compatibility between diagrams.

A SADT model is a hierarchically organized set of diagrams. Each box of a diagram can be thought of as a single, well defined subject. The division of such a subject (i.e., the name of the box) is called *functional decomposition*. Decompositions form boundaries and each box is considered to be the formal boundary around a portion of the entire system being described. At the very top, a single box and several arrows are used to define the boundary of the entire model. A single box describes the overall task to be performed by the system. The arrows that touch the box describe the major controls, inputs, outputs and — if any — mechanisms of the system. The diagram that consists of this single box and its arrows is called the *context diagram* for the model. The box therefore represents the system boundary: everything inside the box is part of the system being described; everything outside the box constitutes its environment.

SADT models evolve through a process of top-down structured decomposition. The title for each diagram is taken verbatim from the box it decomposes. Each diagram is identified by the *node number*. The node number for the context diagram has the form: model name (or abbreviation), slash, the capital letter *A* (for activity model), a hyphen and zero. The node number of the diagram decomposing this box is the same without the hyphen. All other node numbers are formed by taking the node number of the *parent* diagram and appending to it the number of the box that is being decomposed.

C-Numbers are used to identify different versions of the same diagram. Additionally, they are used to link together the diagrams both downward and upward the model hierarchy. Inside the diagram the box that is being decomposed by another one is marked with its C-Number.

SADT diagrams have external arrows — those coming from and going to the outside of the model. To ensure that several diagrams plug together SADT uses an encoding scheme called *ICOM*, which stands for *I*nput, *O*utput, *C*ontrol and *M*echanism. ICOM codes give the analyst a way to quickly verify that the external arrows of the corresponding diagram match the boundary arrows of the corresponding box on the parent diagram.

Node numbers, C-Numbers and ICOM codes handle the vast majority of model intra-connection situations.

3.4 Summary and Advanced Topics

In this section we discussed the SADT methodology to model the activities of a system. We introduced the basic ideas and concepts which prove that SADT is a detailed modeling technique. Two of the major differences between SADT and other structured analysis methods are

1. The seperation of *inputs* and *controls*: SADT gives an analyst the capability to describe explicitly the constraints imposed on transformation functions. Constraints give a more detailed picture of *how* the system operates, for they describe

the rules and facts that must be followed by the transformation function. In some cases it might be useful to join the input and control information into a single control arrow if input information would also control the behavior of the next activity.

2. The importance of *mechanism*: A system is described first from a functional perspective. SADT mechanism arrows give an analyst the ability to precisely define how a particular function will operate, what storage is required, who will do it, and so on. Final details are added to the functions. By using the output-mechanism interconnection a sequencing of system functions is enforced.

During the authoring (i.e., the development of the model) a functional decomposition strategy which is based on the functional relatedness of system activities is often the best. By doing so, the modeler is forced to think about *what* the system does, independently of *how* the system operates. Functional decomposition also tends to deemphasize sequencing while exposing the constraints on system activities.

One major disadvantage of the SADT modeling approach has to be mentioned. SADT activity models exactly describe *what* the system does and *how* the activities are guided. But, *when* the activities are actually executed is left unattended and is not visible from the diagrams (except sequencing which is enforced by output-mechanism arrows). Thus we state, that SADT is a sufficiently comprehensive methodology during the design phase of a system, but the diagrams have to be converted into other models if timing constraints have to be considered. One possibility to overcome this drawback is to describe the timing order of the activities when the diagrams are explained with natural language. Additionally, [29] presents techniques to convert SADT diagrams into colored Petri Nets [15] — a special form of high-level Petri Nets [16]. High-level Petri Nets could be used during the analysis phase if the concurrency and timing order of the activities are of major importance. For our purposes, it is sufficient to describe the timed control flow with natural language.

Finally, we introduce some new notations which are used during the modeling phase. We briefly explain these notations according to Fig. 11. Arrows coming from outside diagrams are drawn with empty heads, whereas arrows which are used inside them are drawn with filled heads. Boxes and arrows of each diagram will be explained by natural language. We use the notation "[B.n]" to refer to a box in the diagram which has dominance n. The associated activity will be written in *Italic* mode. The information which is carried by the arrows will be written in Sans Serif mode.

Until now, we just introduced the syntax of SADT diagrams and models, but left the process of modeling unattended. The steps required for modeling the system are introduced in §3.4. Hand in hand with the description of those steps we develop a model for the load management system.

4 Load Management Design

In this section we will try to adopt the SADT methodology to the design of LoMan. In §4.1 the purpose and viewpoint of the model will be defined. From the viewpoint of SADT activity modeling, *data decomposition* must initially precede and be done concurrently with *activity decomposition*. We will discuss these steps in §4.2. From this starting point we develop the context and the top diagram of LoMan in §4.3 and §4.4. We continue by stepping down the hierarchy uncovering more details in §4.4.

4.1 Purpose and Viewpoint of the LoMan model

For the design of LoMan we will model its activities. According to §3.1, SADT activity models are developed for the particular reason to answer a set of questions.

The set of questions which have to be answered, the resulting purpose of the model, possible perspectives and the chosen viewpoint are depicted in Fig. 12.

To be more precise, we state that the chosen viewpoint is the heart of every load management mechanism: the decision component. All activities which have to be performed by LoMan are controlled by this component. The decision component evaluates the current load situation and additional demands. One exceptional property compared to other load management systems is that LoMan will also consider the costs of the activities before it actually activates the distribution components.

4.2 Data and Activity List

The first step for developing the LoMan model is to determine the data and activity lists. The SADT methodology advises the separation of necessary data items — which are a collection of "things", and not specific data types in programming languages — and the actions to be performed. In §4.1 we defined the purpose of the model by identifying the question set and fixed the viewpoint on the system. The *decomposition* of data items and activity is done *according to the roles each component plays in the LoMan cycle*.

In Fig. 13 we listed a vast collection of information used and transformed by LoMan. Two major groups of input may be identified. The first group covers all information and descriptions which relate to a parallel application and its constituting tasks. The second group covers the information about the workstations on which the application may execute. The physical entities in those groups have to be managed by LoMan, whereas the logical entities are used as information guides. Additionally, we identified the external users as a source of input for LoMan. External user demands and their sequential jobs have to be considered by LoMan. Users may restrict the set of possible targets (i.e., workstations on which tasks may execute) by interactive node usage. All information regarding the interdependency between the executing application and the involved workstations will be handled inside the LoMan cycle.

used at:	*$Author: roeder $* *Project: LoMan* *Notes: 1 2 3 4 5 6 7 8 9 10*	*Date: Febr 4th, 1997* *$Revision: 1.2 $*	■ *Working* / *Draft* / *Recommended* / *Publication*	*Reader:* *Date:*	*Context:* none

Questions:

how to obtain balanced system utilization
while preserving minimum runtime?
how to handle external users and their loads?
how may the application influence the load determination?
when is the load model activated?
what are the load managers tasks?
how are dynamically changing machine configurations handled?
what information is needed inside the load manager?
what costs have the LoMan activities
and what are their influences?

Purpose:

Identify the tasks and their causal ordering of a load management system in heterogeneous execution environments in which sequential user processes execute concurrently with parallel applications.

Perspectives:

the parallel application
the execution environment
the load manager
the user of the parallel application
an external user

Viewpoint:

The decision component of the load management system.

Node: LoMan/A-0	*Title:* Purpose and Viewpoint for LoMan $RCSfile: LoMan-purpose+viewpoint.fig,v $	*Number:* CR000

Figure 12: Purpose and Viewpoint of LoMan's SADT model

used at:	*Author:* Roeder *Date:* 02/04/97 *Project:* LoMan *Revision:* 1 *Notes:* *1 2 3 4 5 6 7 8 9 10*	■ *Working* *Draft* *Recommended* *Publication*	*Reader:* *Date:*	*Context:* none

- parallel application
- task group
- task description
- task communication relation
- task resource requirements
- priority of tasks
- user host requirements
- external user
- sequential jobs
- application architecture constraints
- application characteristic
- application historical info
- application progress
- application execution time
- task execution time
- task auxiliary signals
- task states
- task identifier
- task location
- task home node (creator)
- task architecture constraints
- application’s file system location
- task load
- unfinished application
- unfinished task
- workstation (=node)
- set of homogeneous nodes
- set of heterogeneous nodes
- restricted architecture set
- unconditional architecture set
- operating system version
- node static description
- node dynamic description
- node functional description
- node performance description
- file system location
- node load values
- user activity
- set of actual nodes
- set of possible nodes
- node identifier
- prepared node
- node states
- actual mapping
- recommended mapping
- list of mappings
- load model
- load value
- set of load values
- possible load model
- configured load model
- node attached
- node detached
- list of load values
- cost estimates
- time estimates
- configuration costs
- migration costs
- possible load model
- actual load model
- task data location
- task communication frequency
- task average message size

Node:	*Title:*	*Number:*
LoMan/A-0	Data List for LoMan Modelling	CR000

Figure 13: Information moved during the LoMan cycle

used at:	*Author:* Roeder *Date:* 02/04/97 *Project:* LoMan *Revision:* 1 *Notes:* *1 2 3 4 5 6 7 8 9 10*	■ *Working* *Draft* *Recommended* *Publication*	*Reader:* *Date:*	*Context:* none

- receive parallel application
- release parallel application
- analyse application constraints
- analyse application architecture
- map tasks
- migrate tasks
- initialize mapping list
- update mapping list
- suspend application
- suspend task
- evaluate task requirements
- determine runtime of parallel appl.
- determine runtime of task
- migrate tasks
- update task execution time
- redistribute load

- evaluate costs
- evaluate action costs
- evaluate migration costs
- evaluate configuration costs
- evaluate mapping costs

- determine load fluctuations
- determine load values
- configure load model
- receive load values
- compare load values
- reset load models
- initialize load models
- reconfigure load models
- observe node load
- observe load of node set
- receive node status

- evaluate optimization criteria
- conclude decision
- evaluate action's tradeoff
- select migration tasks
- select target node
- select source node
- execute selected actions
- prepare possible actions
- prepare mapping
- startup LoMan components
- create mapping list
- update mapping list
- suggest mapping

- observe user activity
- receive user logins
- receive user logouts
- receive user contraints
- receive sequential job
- map sequential job
- release sequential job

- setup node set
- release node set
- reserve node
- select nodes
- evaluate node performance
- setup possible environment
- configure node set
- reconfigure node set
- expand actual node set
- reduce actual node set
- determine system performance

Node: LoMan/A-0	*Title:* Activity List for LoMan Modelling	*Number:* CR001

Figure 14: Activities to be performed during the LoMan cycle

Hand in hand with the development of the data list we listed and organized the activity list shown in Fig. 14. According to the groups of information we synthesized five activity groups: (1) activities for application management, (2) activities for controlling the execution environment, (3) the decision making process of LoMan, (4) the load, cost and mapping evaluations and the (5) observation of the external users. The activity groups will serve as a basis for the top diagram of LoMan which will be developed in §4.4

4.3 The LoMan context diagram

The context diagram of LoMan (see Fig. 15) identifies the general view on what has to be done by the load manager. Additionally, it repeats the purpose and the selected viewpoint derived in §4.1.

We will briefly discuss the context diagram. The overall question to be answered by the load manager is "*when* to distribute *which* load producing object *to which* load consuming component?". Hence, the overall activity to be performed by LoMan to answer this question is to *manage the load distribution* [B.1]. With the term "distribute" we refer to both initial task placement and dynamically redistribution of already executing tasks. The box and the arrows in the context diagram build the boundary between the LoMan system and its environment. Any information handled inside LoMan is not shown by the context diagram. This information will rather be the output of internal activities. For example, the mapping between tasks and workstations is a result of internal activities and therefore it is not shown in Fig. 15.

To be more precise, we have to discuss the boundaries in more detail. Recall, that the semantics of an SADT box may be formulated as follows: "Under control, input is transformed into output using the mechanism." Thus, we have to describe these information items.

The collection of input things is subdivided into three parts:

1. The term parallel application (I-1) refers to all static and dynamic characteristics of an application which are needed by LoMan. For example, this includes the name of the workstation on which the application has been started, the resource requirements of the tasks, the number of tasks, the architecture constraints and so on.

2. The term execution environment (I-2) refers to all the information which relate to the coupled heterogeneous workstations. The execution environment is under control of LoMan as long as a parallel application executes. For example, this includes the static descriptions of the available workstations, the actual configuration of workstations on which the application tasks execute, and so on.

3. The term external user demands (I-3) covers all possible input given by the users of the cluster, whether explicitly or implicitly. The demands may have influence on how LoMan reacts. For example, the presence of an user may exclude a

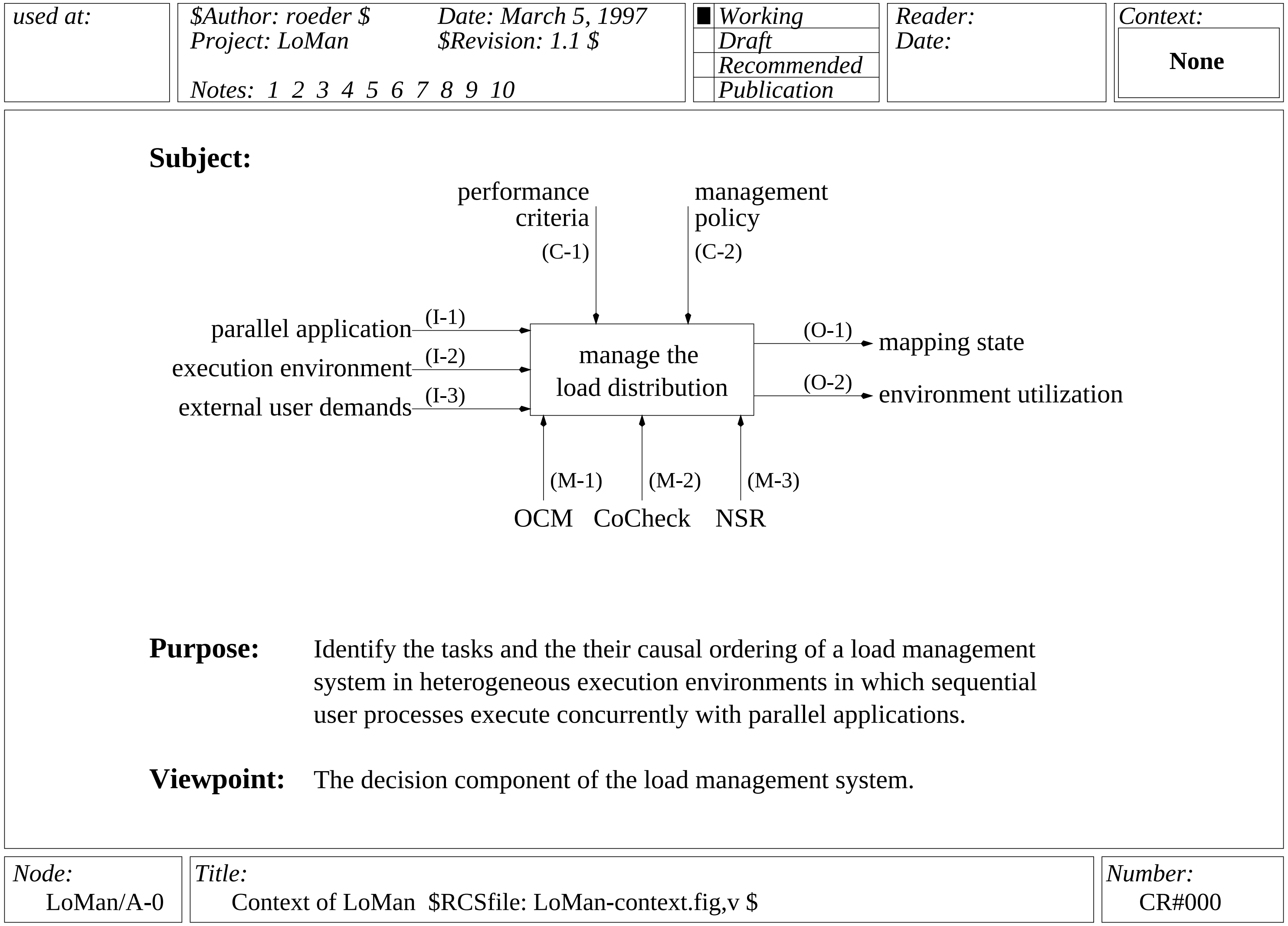


Figure 15: The SADT context diagram of LoMan

workstation for further usage. On the other side, it may be possible that LoMan distributes a single sequential job if a user gives permission to do so, thus not disturbing the execution of an application task (until now, this is not explicitly modeled).

We are interested in a balanced resource utilization in the cluster while minimizing the execution time of the parallel application. The output covers exactly those two goals. At this point, we do not consider how to define these measures. The mapping state (O-1) can be seen as an online display of the actual mapping of application tasks to workstations. The environment utilization (O-2) is strongly interrelated with this mapping state. It can be viewed as an online statistics of the current utilization of the workstations.

The control input are the objectives and directives for the activities. The performance criteria (C-1) have to be considered during the decision making process of LoMan. In general, the performance criteria cover the balanced resource utilization and the minimum execution time of the application. The management policy (C-2) may be thought of as blueprints which indicate how the activities have to be coordinated (e.g., sender-, receiver-, mixed- or centralized strategies).

The mechanism inputs cover all auxiliary components and tools which are used by LoMan, but which are not part of LoMan itself. The mechanisms provided by the operating system of each workstation and its additional services are mainly used for the low level functionality of LoMan (e.g., for the communication between the LoMan components). Three different auxiliary tools are used: OCM (M-1) [22] is an *O*MIS [23] *C*ompliant *M*onitoring System. The OMIS project aims at defining a standard interface between tools for parallel systems and monitoring systems. OCM is designed for PVM programs running on workstation clusters. CoCheck (M-2) [41, 42] (*Co*nsistent *Check*pointing) is used to create application checkpoints and enable task migration. Finally, NSR (M-3) [14] (*N*ode *S*tatus *R*eporter) is used to determine static and dynamic characteristics of the nodes in the system. For example, static characteristics cover architecture description and performance indices of the nodes, whereas dynamic characteristics cover the load situation and interactive usage.

All information things will be described in more detail when they are actually used, i.e., while we are stepping down the hierarchy by further functional decompositions.

4.4 The LoMan top diagram

SADT activity modeling is based on *functional decomposition*. In the following, we will describe the top diagram of LoMan as shown in Fig. 16. The top diagram is located at hierarchy level 0. The most important activities of LoMan are subdivided into four main parts. Each of the parts influences the others in some ways. We will discuss those influences by describing the data- and controlflow between the boxes. Further decompositions of the four boxes are discussed in §4.4.

The decomposition of the context diagram results in the following activities (the data- and controlflow will be described further down):

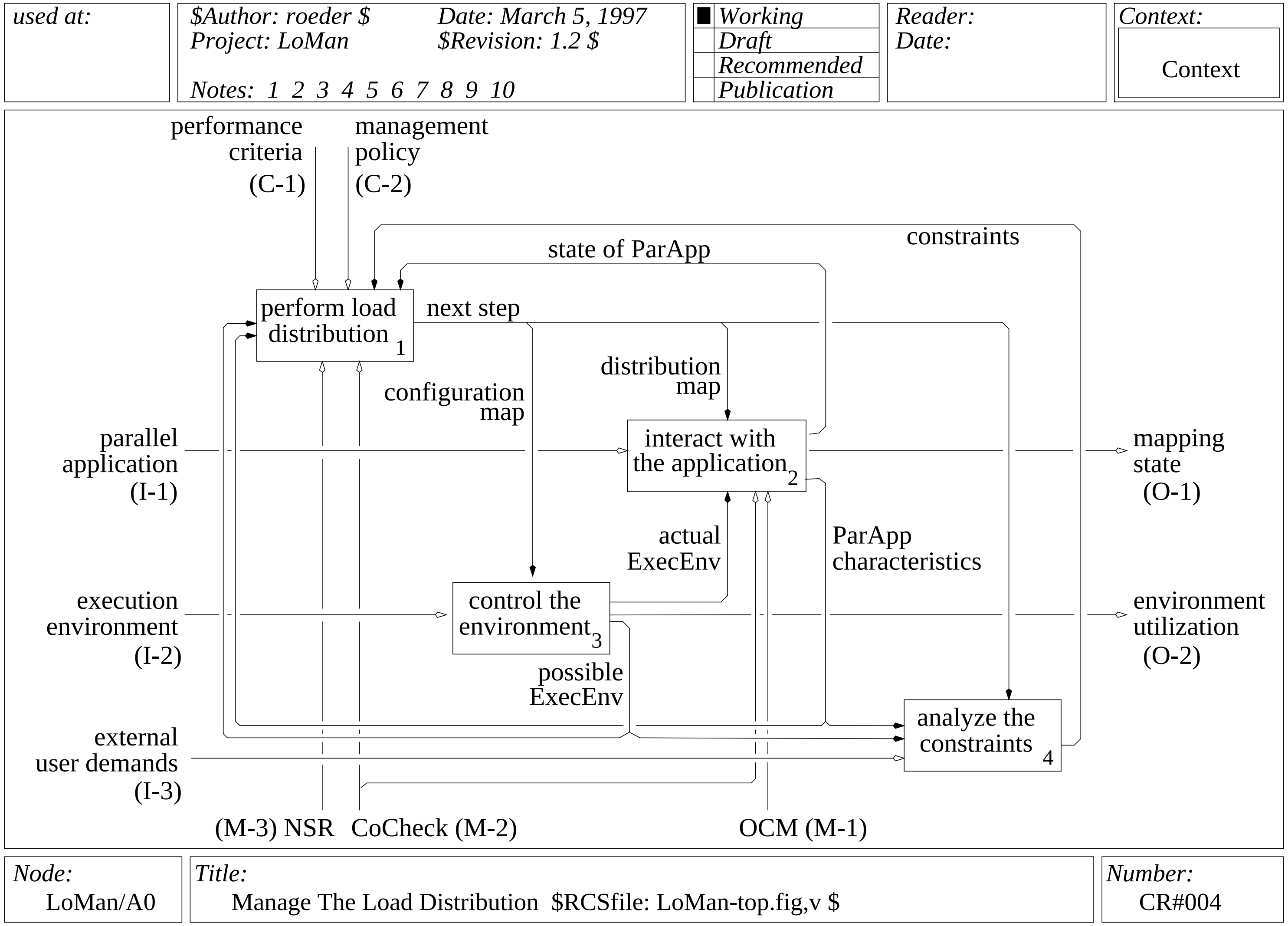


Figure 16: The top level SADT diagram of LoMan

1. The most dominant box defines the main activity to be performed by LoMan, i.e., to *perform load distribution* [B.1]. The standard control loop of the load management cycle is located within this activity, i.e., the load measurement and the decision component. The main purpose of this activity is to determine the sequence of the subsequent activities (see §5.1 for further details). NSR will be used to determine the load situation on the nodes.

2. The load management system has to *interact with the application* [B.2]. These interactions manipulate the mappings of the application tasks to the execution nodes and influence the decision component of LoMan. The first activity of the load management system is to accept the application as soon as it is submitted by the user (see §5.2 for a more detailed decomposition).

3. Additionally, LoMan has to *control the environment* [B.3], i.e., the nodes which have to be configured to execute the application tasks. In general, the set of nodes on which the application tasks are executed is not known in advance. The set will rather be configured before the application tasks are initially mapped (see §5.3). If necessary and possible, the preferences of the application user will be considered.

4. The environment implications *heterogeneity* and *time-sharing* are evaluated while LoMan will *analyze the constraints* [B.4]. These constraints are outside the scope of the standard control loop. The extended control loop of LoMan will be directed by the objects to be managed (i.e., the execution nodes and the application tasks) and by external user requirements (see §5.4 for more details).

Input arrows coming from and output arrows going to the context diagram have been discussed in §4.3. We will now describe the dataflow and controlflow within the top diagram as seen from the perspective of LoMan's decision component. According to the dominance of boxes in Fig. 16 we will start our discussion at the top left corner of the diagram.

The activity *perform load distribution* has two input items: the ParApp characteristics and the possible ExecEnv. Both inputs are results of the two less dominant boxes *interact with the application* and *control the environment* respectively. Thus, the arrows representing those input items are drawn as input-feedback. The term ParApp characteristics groups static and dynamic information about the application, e.g., the actual number of application tasks and their resource requirements. The possible ExecEnv covers information about the workstations which are possible targets to distribute the application tasks. This set of nodes could contain more workstations than are actually involved for executing the application. Similarly, the execution of *perform load distribution* is controlled by results of two less dominant boxes. The control-feedback information state of ParApp and constraints are outputs of *interact with the application* and *analyze the constraints* respectively. For example, the application's executable code versions may be limited to a certain architecture or external users might restrict the use of dedicated workstations. These are examples for constraints. An important control-feedback is expressed by the state of ParApp since the decision component operates and produces different results depending on this state. The two most obvious differences are whether the application tasks have to be initially mapped or dynamically redistributed. Thus, a "ready-to-run" application will direct the operation in another way than already "executing" application tasks. The output of *perform load distribution* could be characterized twofold. On the one hand side, the decision component determines the next step to be performed, i.e. whether to *interact with the application*, to *control the environment* or to *analyze the constraints*. Parts of these information include the configuration map for the workstations to be included into the actual ExecEnv. Other parts cover the distribution map for the mapping between the application tasks and the target workstations. Which part of the information — covered by the next step — is actually used depends on the selected activity.

A classic output-mechanism connection between two boxes is given by actual ExecEnv. As discussed in §3.2 this kind of interconnection denotes a sequencing relationship, like the resource allocation problem. In our case, the workstations have to be configured and prepared by *control the environment* before the application tasks are distributed by *interact with the application*.

Each of the above boxes and their relevant information will be discussed in more detail in §4.4. The strength of the SADT modeling approach is now visible: having a general idea of what LoMan will do we are able to uncover more details while stepping down the diagram hierarchy.

5 Structured Decomposition of LoMan

In §4.4 we described the top diagram of LoMan. The four activities depicted in that diagram (see Fig. 16) serve as a starting point for further functional decomposition. In this section we want to step down the hierarchy of LoMan's activities. We will recognize the strength of the SADT modeling technique as an byproduct during that journey. The interdependency of the activities, i.e., to *perform load distribution* (see §5.1), to *interact with the application* (see §5.2), to *control the execution environment* (see §5.3) and to *analyze the constraints* (see §5.4), will be described in consistent manner. Note, that we stay strictly in the perspective of the decision component of the load management system.

5.1 Performing Load Distribution

The main goal of LoMan is to distribute the load producing tasks to the load consuming nodes. The *standard* control loop of every load management mechanism is located within the activity *perform load distribution* (see §1.2). This includes the measurement and collection of the load values in the execution environment and the subsequent comparison of the load values. Additionally, the decision component has to consider the two system implications heterogeneity and the time-sharing usage model. It might be necessary to refine and optimize the decisions which are based only on load values. The costs of reconfiguring the set of actually involved nodes and the migration of tasks between nodes might outweigh the expected performance benefits (see §2.3). In this case, it might be better not to change the actual mapping between tasks and nodes or to find another cost effective mapping.

5.1.1 Functional Decomposition

The functional decomposition of *perform load distribution* at level 1 of the SADT hierarchy leads to four activities which are illustrated in Fig. 17. The following list also discusses the information items to be transformed, the resulting output, the guiding constraints and the mechanisms used. The separation of topic 2. and 3. enlights the difference between the standard and the extended control loop.

1. LoMan has to *determine the load situation* [B.1] in the possible ExecEnv since the system wide loading situation, i.e., the system load, serves as one main determining part during the decision making process. The NSR processes which are located on the nodes in the possible ExecEnv are used to measure and collect the load values. The system load is the description of the load situation on each possible target node. Initially, the load model configuration describes a default set of load values to be measured. The timer step initiates the activity. In our first approach we assume that this activation signal is enabled at fixed time intervals, although more sophisticated mechanisms might be used. For example, activities

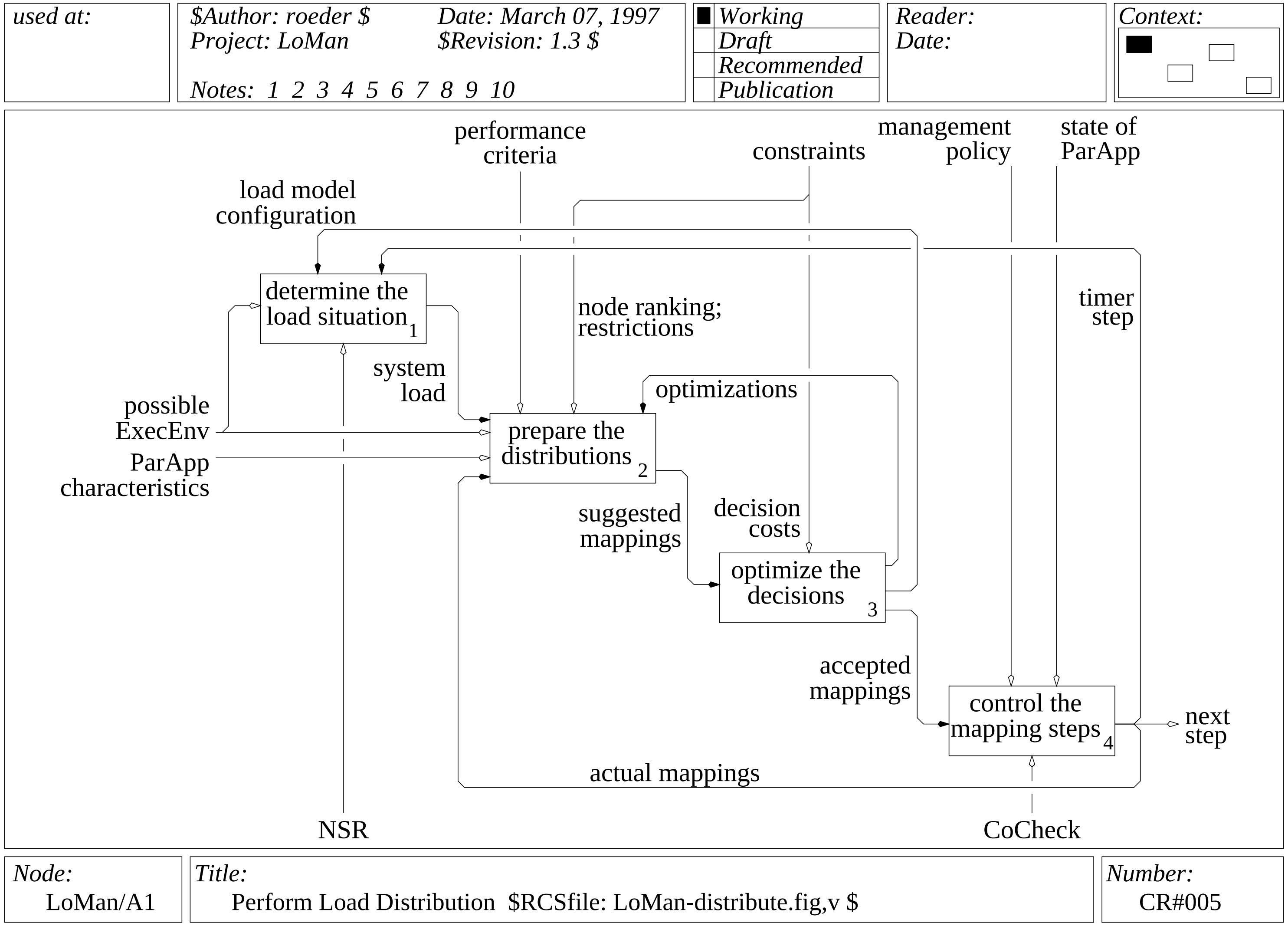


Figure 17: LoMan activity: perform load distribution

are only initiated if changes of load situation exceed a predefined threshold. The succesive activities depend on the availability of the new load values (see §5.1.2).

2. LoMan will *prepare the distributions* [B.2] according to the system load by calculating the mappings between the application tasks and the nodes in the possible ExecEnv. The resource requirements of the application tasks are described by the application characteristics. The actual mappings are considered if application tasks already execute or new tasks are created. The output of this activity are the suggested mappings which contain both the configuration map for the set of nodes to be included into the actual ExecEnv and the distribution map for the mappings between the application tasks and the target nodes. Several constraints coming from outside the diagram direct the activity. The performance criteria and the management policy describe how the mappings have to be calculated. The constraints will provide a priority order of currently available nodes (see §5.4). The optimizations which are output of the less dominant box may force a recalculation of the suggested mappings. We will discuss this item right below and refine the activity in §5.1.3. This activity mainly covers the standard control loop which was reviewed in §1.2.

3. LoMan might *optimize the decisions* [B.3] by evaluating the suggested mappings according to the management costs they introduce to the system (see §2.3.1). One of the major impacts to the efficiency of the decision are the costs to reach the given performance criteria. The directing constraints, i.e., the costs for migrating tasks between nodes will finally determine the accepted mappings. As a control-feedback, the optimizations guide the new calculation of the suggested mappings if the costs outweigh the expected performance benefits. Similarly, the load model configuration depends on the resource fitness between the nodes and the tasks' requirements (see §5.4). Thus, not only the system load but also the constraints direct the list of mappings and configurations in the accepted mappings.

4. The behavior and interrelation of all activities in the diagram is mainly guided by the state of the application. The timer step starts with the extended management cycle as soon as the application is ready-to-run. The accepted mappings are part of the next step and therefore, will direct and *control the mapping steps* [B.4] which are subsequently performed by LoMan. The output is represented by next step and is visible to the upper diagram. Two different kinds of control inputs to other boxes are visible: on the one side, next step controls the ordering of subsequent activities, and on the other side detailed information is provided on what the directed activities have to do.

The next sections discuss further functional decompositions of the activities. In §5.1.2 we show what activities have to be performed during the determination of the system load. The preparation of the task distribution will be discussed in §5.1.3.

5.1.2 Determine the Load Situation

As mentioned in §2.3 we do not consider load modeling in detail, but we describe here what activities have to be performed to determine the system load situation. A further decomposition of the first activity of the diagram shown in Fig. 17 results in additional four activities. The NSR is used to determine the load of the nodes in the possible ExecEnv. The activities are illustrated in Fig. 18 and explained in the following list:

1. Initially, the NSRDs (*NSR* *D*aemon processes) measure and propagate a default set of new load values which indicate the load on the nodes. LoMan might *configure the NSRD* [B.1] according to the load model configuration. The node locations (I-1) determine the nodes in the possible ExecEnv on which the NSRD reside.

2. LoMan will *receive the load vectors* [B.2] with the aid of NSR. Hence, LoMan receives plain load values at every timer step. The plain load values might even have different semantics for different operating systems. The schedulers and memory management strategies of the local operating systems might differ to an extend to which the load values are calculated in different ways. A pre-scaling with respect to the heterometric characteristics of the nodes is performed by the NSRD and has not to be considered by LoMan.

3. The performance indices (I-2) of the nodes are used to *scale the load values* [B.3]. The scaling operation produces a comparable load description for each node.

4. Finally, LoMan produces a description of the system load during the activity *aggregate to a global view* [B.4]. The system load can be viewed as a list of scaled load values.

5.1.3 Prepare the Distributions

The preparation of the suggested mappings is illustrated in Fig. 19. The functional decomposition of *prepare the distributions* results in three activities:

1. LoMan has to *propose the node selection* [B.1] on which the application tasks should be mapped. Under control of the constraints a load ordered list of nodes is calculated according to the system load. Additionally, LoMan will consider all nodes in the possible ExecEnv to suggest the node selection. Although, nodes which are included in the actual mapping will receive higher priority. A configuration history of the nodes is kept inside LoMan to avoid (or at least reduce) the problem of node fickleness (see §1.3).

2. According to the performance criteria LoMan will *calculate heuristical task mappings* [B.2] for the current system load. From the dynamic ParApp characteristics (e.g., the number of tasks to be mapped) possible (node,task) mappings are suggested which may include several alternatives.

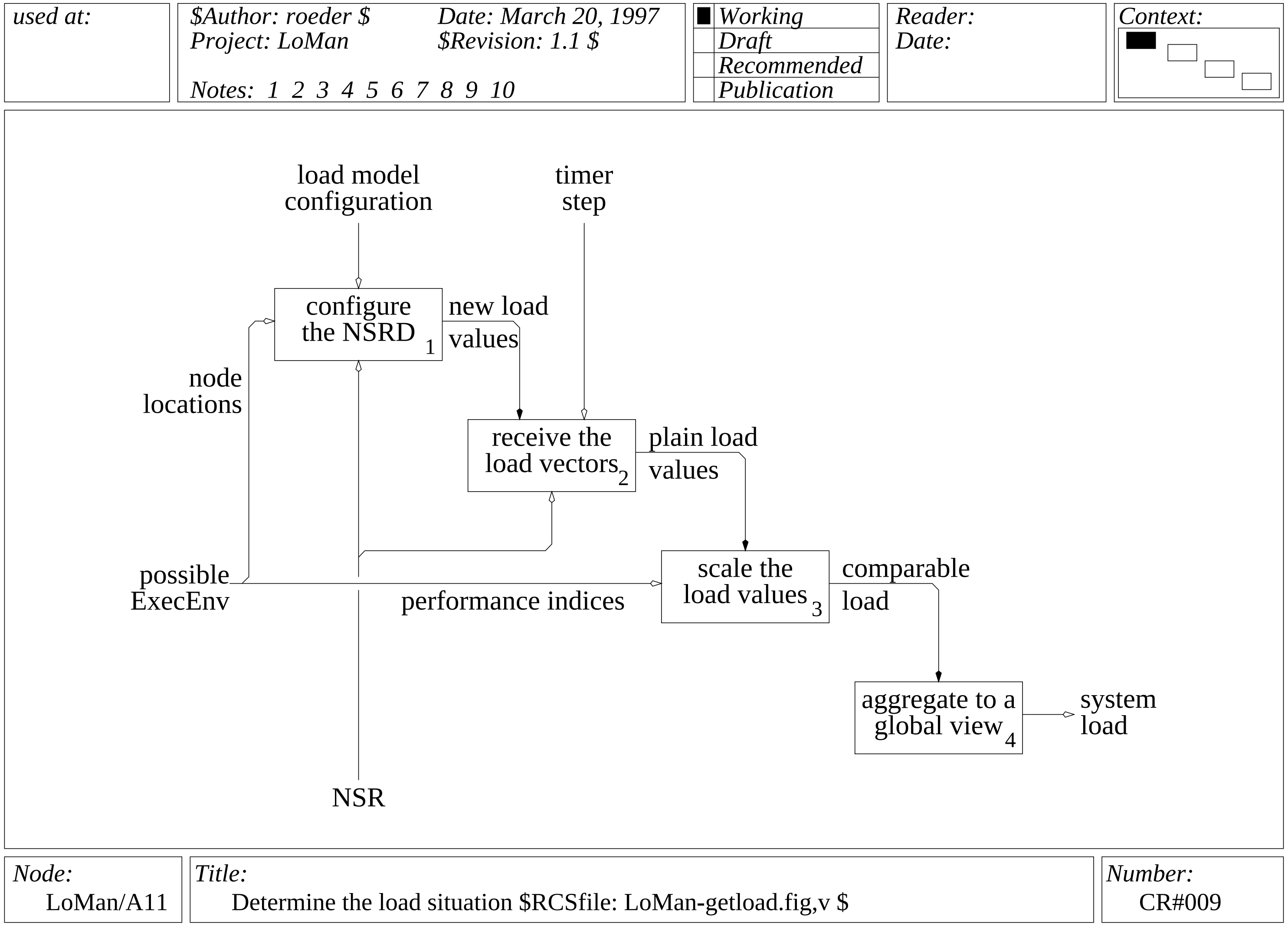


Figure 18: LoMan activity: determine the load situation

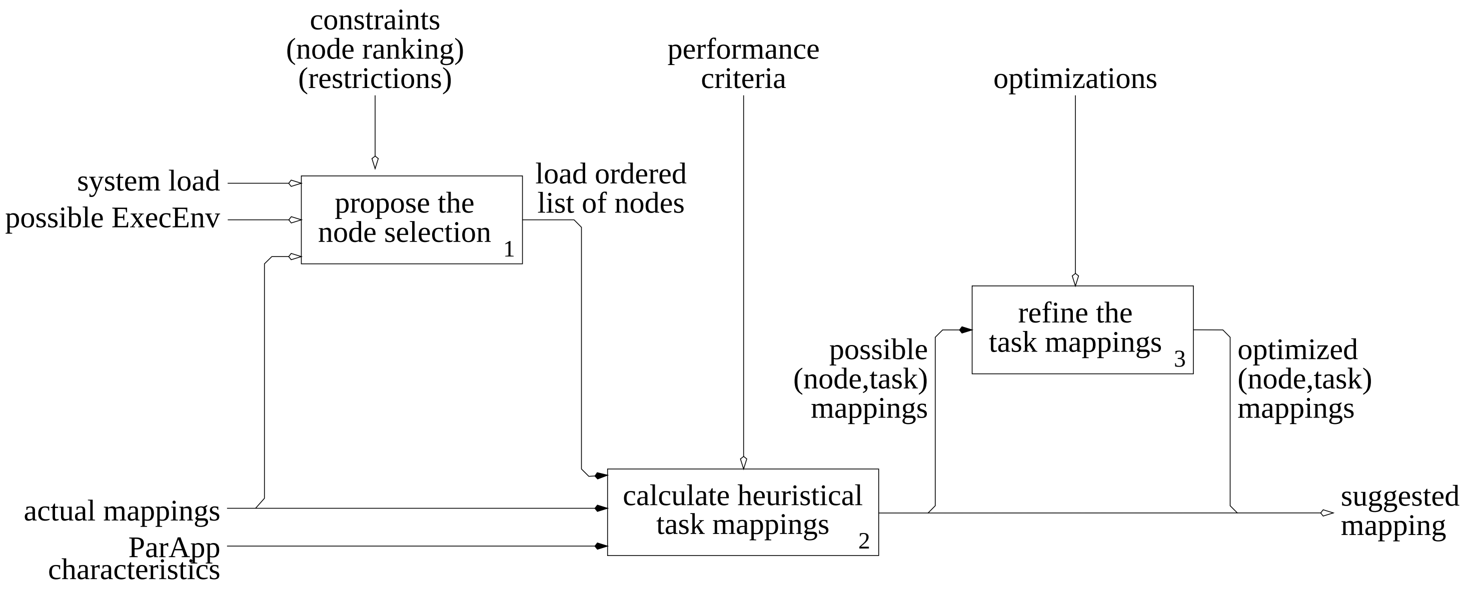
used at:

$Author: roeder $ *Date: March 24, 1997*
Project: LoMan *$Revision: 1.1 $*
Notes: 1 2 3 4 5 6 7 8 9 10

■ *Working*
Draft
Recommended
Publication

Reader:
Date:

Context:



Node: LoMan/A12

Title: Prepare the Distributions $RCSfile: LoMan-prepare.fig,v $

Number: CR#012

Figure 19: LoMan activity: prepare the distributions

3. Due to the optimizations LoMan will *refine the task mappings* [B.3]. Therefore, it selects the optimized (node,task) mappings out of the possible (node,task) mappings. The suggested mappings will be analyzed when LoMan will *optimize the decisions*.

5.2 Interacting with Parallel Applications

The interaction of the load management system with the parallel application and its constituting tasks mainly controls the mapping between the tasks and nodes. On the other side, it also directs the behavior of and the decision to be concluded by LoMan since the extended cycle starts as soon as the user submits the application. These interrelations between the application and the management cycle is formally expressed in the top diagram (see §4.4). We will now refine the steps which are executed to *interact with the application*.

5.2.1 Functional Decomposition

The functional decomposition of *interact with the application* at level 1 of the SADT hierarchy results in four activities which are illustrated in Fig. 20. The following list also discusses the data- and controlflow between the boxes in the diagram.

1. The first interaction is guided by the initial step (C-1) of LoMan during which it has to *accept the application* [B.1]. The description of the parallel application is the only input for this activity (and the whole diagram). Some information has to be stored and prepared after LoMan has accepted the application. For example, LoMan stores the identifier of the node on which the application was submitted. Three different output items indicate the subsequent behavior of LoMan. The ready-to-run state of the application indicates that LoMan has to subsequently prepare the actual ExecEnv on which the initial tasks have to be mapped. No task is executing yet. The static ParApp characteristics, i.e., the statics, are extracted during this activity. For example, static application characteristics include the number of initially created tasks (see §2.1) and their architecture requirements.

2. The activities necessary to actually distribute the tasks to nodes are performed while LoMan will *manipulate the task mappings* [B.2]. The control input for the whole diagram is the next step. This control input splits into more detailed information items which depend on the state of the application. One refinement of the next step is the required mapping (C-2) which may be seen as a recipe. This recipe indicates which tasks have to be distributed to which nodes. The tasks execute on the nodes in the actual ExecEnv after they have been distributed and hence, the state of the application changes from ready-to-run to executing. The state executing is kept as long as at least one application task is still executing. The mapping state is updated every time at which new distribution is required, i.e., when the current mapping changes.

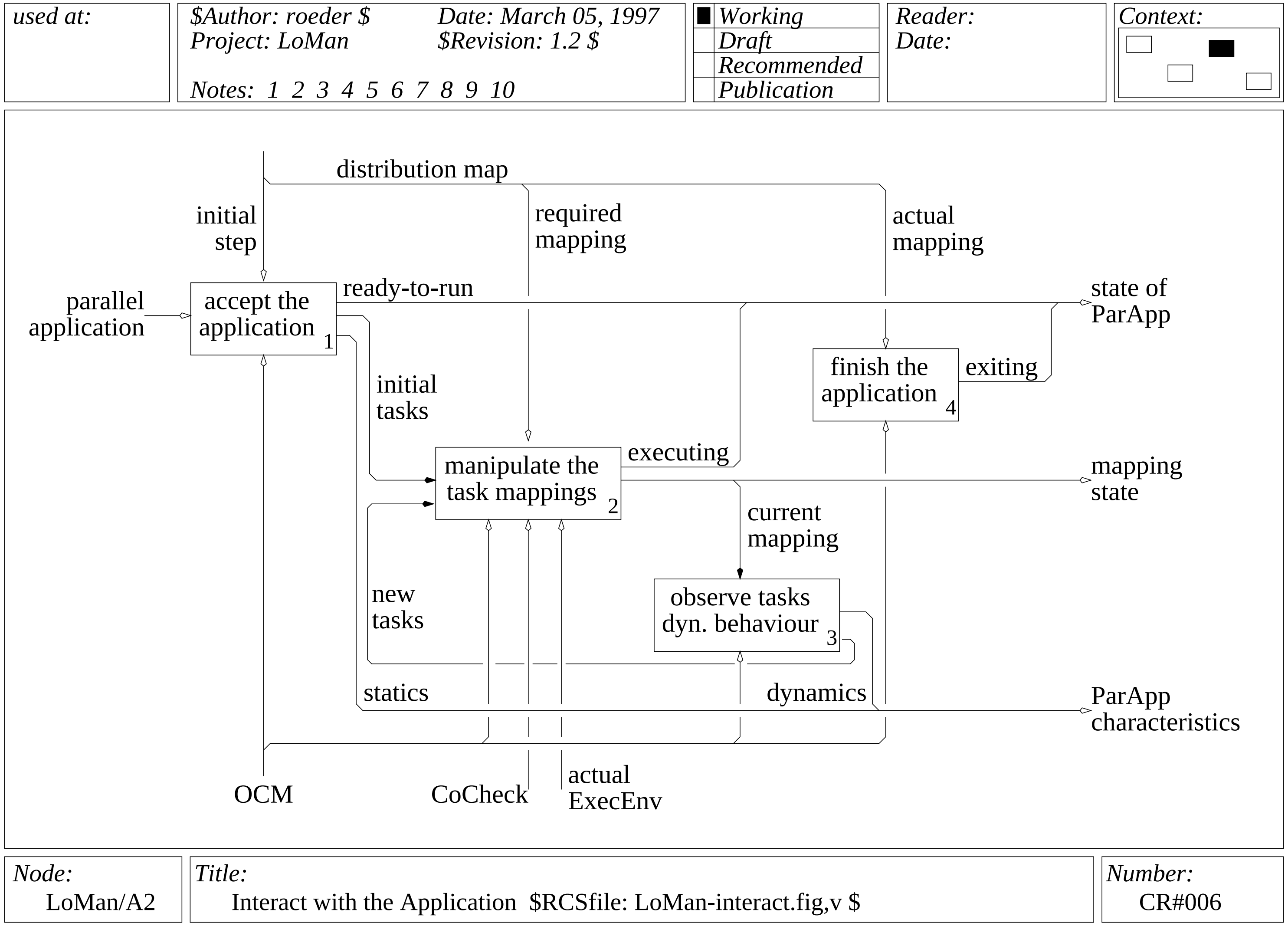


Figure 20: LoMan activity: interact with the application

3. In order to configure the load model, i.e., to identify the resource requirements of the tasks which have the major impact on the system load, LoMan has to *observe tasks' dynamic behaviour* [B.3] in the actual ExecEnv. According to the current mapping the dynamic ParApp characteristics, i.e., the dynamics, are used to configure the load model. There is no input for the activity since there is nothing to be transformed but rather to be monitored. Additionally, LoMan recognizes if new tasks are created during the runtime of the application. Those new tasks will also serve as input for *manipulate the task mappings*, since they have also to be mapped to nodes.

4. Finally, LoMan will *finish the application* [B.4] as soon as no more tasks are executing and no more tasks are created. The actual mapping dictates the garbage collection in the execution environment, i.e., which additional processes have to be removed by LoMan from the possible ExecEnv. Those additional processes may serve as the runtime environment for the parallel application. Again, the cleaning of the possible ExecEnv depends on the state of the application. The state exiting starts cleaning up of the environment.

5.2.2 Manipulate the Task Mappings

LoMan steps through several activities to *manipulate the task mappings*. Additionally, the states of application and tasks are under control of LoMan. The functional decomposition is illustrated in Fig. 21:

1. As soon as the parallel application is ready-to-run and creates its initial tasks, LoMan will *accept the tasks* [B.1]. Accepting the initial tasks includes the preparation of internal management lists. For example, for each task its creator is stored. Thus, LoMan knows the location of the home node for every task. Additionally, dynamically created new tasks will be caught in a first stage. Both types of tasks are under control of LoMan and they are ready to be mapped.

2. LoMan will *create the tasks* [B.2] by mapping them onto the nodes in the actual ExecEnv. Which task has to be mapped onto which node is indicated by the initial mappings. Although the tasks are created, they are not yet executing.

3. LoMan will *control the task states* [B.3]. Hence, it is responsible to start, stop and continue the tasks on the nodes to which they have been mapped. The instructions to perform these controlling steps (and the preparation of the task execution) are provided by OCM. The decision unit of LoMan indicates the migration of already executing tasks by providing the redistribution mappings. Thus, the activity is also responsible for migration control. Two subsequent activities will be performed in that case:

4. LoMan will *prepare the migration* [B.4] by providing the source/target nodes which have been extracted from the redistribution mappings. The checkpointing facility of CoCheck is used to set up a skeleton task.

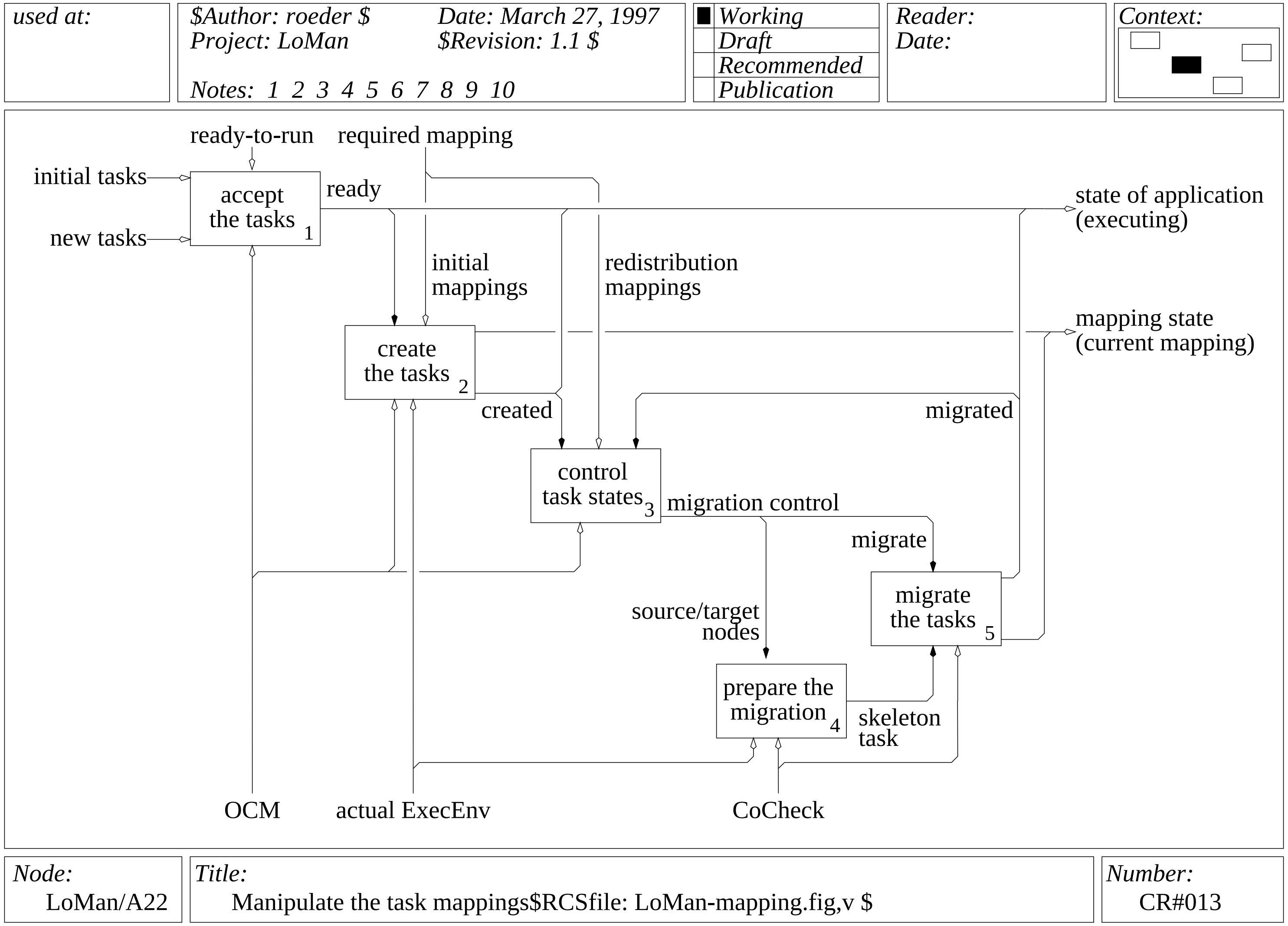


Figure 21: LoMan activity: manipulate the task mappings

5. CoCheck is also used to initialize the skeleton task with a previously saved checkpoint. Hence, LoMan is also responsible to *migrate the tasks* [B.5].

Both activities *create the tasks* and *migrate the tasks* will update and provide detailed information about the current mapping state.

5.3 Controlling the Execution Environment

One of the first steps of the decision component is to select a set of nodes on which the application tasks should execute as soon as the application is ready-to-run. The activities which are necessary to manage the execution environment are directed by the results of the extended load management cycle. The main purpose of this activity is to set up and configure a target set of workstations. The set of workstations has to satisfy the architectural requirements imposed by the parallel application. In case of a changing load situation in a primarily configured set of nodes this set might be reconfigured. Less loaded nodes will be added while overloaded nodes will be excluded from the set of actually involved nodes. Similarly, if external users restricts or allows the use of single workstations (see §5.4 for details) an already configured set of nodes might change.

5.3.1 Functional Decomposition

The functional decomposition of *control the environment* in the SADT hierarchy leads to four activities at level 1 which are illustrated in Fig. 22. Which activity will be executed is determined by the information of next step — the general result of *perform load distribution*. The next step contains the configuration map if this activity is selected by the decision component.

For the discussion we will distinguish between the two terms list of nodes and set of nodes. The former describes the information controling the configuration of the execution environment while the latter is used to describe the physical set of nodes. The roles of Node Marker, Node Attacher and Node Detacher are explained in more detail further down.

1. LoMan will *setup the ExecEnv* [B.1] after the application is submitted by the user and is ready-to-run. During this activity the available heterogeneous execution environment which represents the unconditional set of all nodes is restricted to the nodes which are compatible to the executable code versions of the application tasks. The resulting possible ExecEnv is a subset of all workstations in the heterogeneous cluster which meet the architecture requirements. Thus, the possible ExecEnv may also be viewed as a set of at least one homomerous subsystems (see the definitions in §2.2.1) and is the observation space for LoMan. Initially, the possible ExecEnv is identical to the possible set of nodes which serves as input for the next activity. The architecture requirements are part of the configuration map

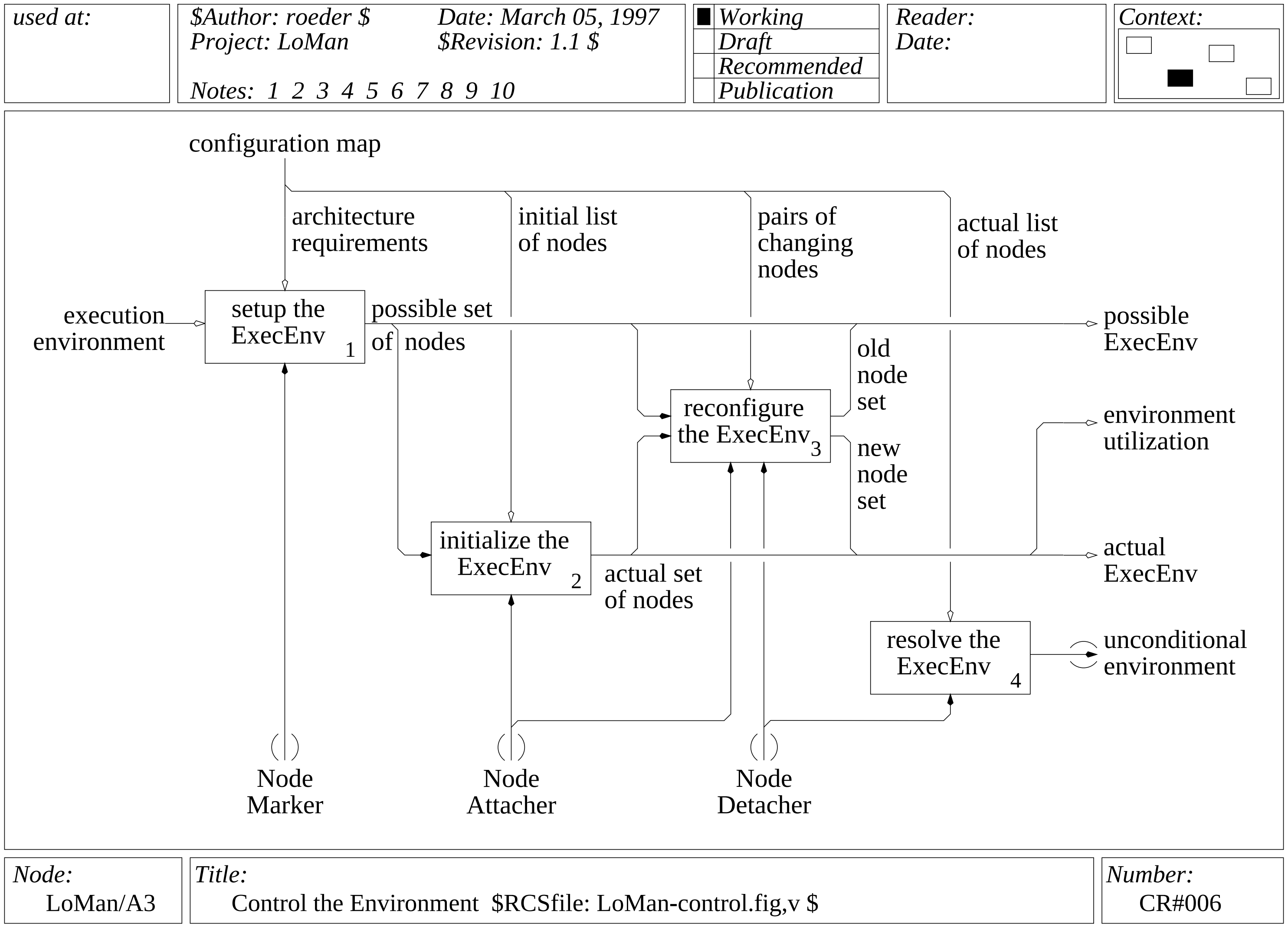


Figure 22: LoMan activity: control the execution environment

and are essentially a list of required architectures. This list is used by an auxiliary tool — the Node Marker — to mark those nodes for possible future usage (see below). It is likely to happen that not all selected workstations are necessary or currently allowed to serve as targets for executing the application (e.g., if 8 nodes are selected for an application which consists of only 4 tasks). For example, the load management system will configure the NSR processes only on the marked nodes.

2. LoMan will *initialize the ExecEnv* [B.2] by selecting nodes of the possible ExecEnv according the dynamic application constraints and the system load. The constraints are evaluated in *analyze the constraints* (see §5.4). The result is the actual ExecEnv on which the application tasks will be mapped and executed, i.e., LoMan's activity space. From a logical point of view, the Node Attacher takes the initial list of nodes to set up all processes which are necessary to execute the application tasks (i.e., the runtime environment for the applications). Additionally, the Node Attacher is responsible to set up any process which is required for the load management cycle. The recording of the environment utilization is initialized.

3. Several conditions might force LoMan to *reconfigure the ExecEnv* [B.3]. For example, the load situation in the possible ExecEnv changes dramatically or nodes which have been previously occupied by external users are available again. A list of pairs of changing nodes dictates which nodes have to be included into and excluded from the actual set of nodes. Remember, that the actual ExecEnv is a subset of the possible ExecEnv. The attached nodes are grouped in the new node set, whereas nodes which are removed are grouped in the old node set. Again, the Node Attacher is responsible to set up the necessary processes to execute the application tasks. On the other side, the Node Detacher (M-1) removes those processes. Furthermore, the environment utilization is updated.

4. LoMan will *resolve the ExecEnv* [B.4] as soon as the application exits. The Node Detacher uses the actual list of nodes to detach the actually involved nodes, i.e., to remove all relevant processes. The result of this activity is an unconditional environment, i.e., the nodes are not longer under control of LoMan. (One of the last activities could be the reconfiguration of the load models within the NSRD processes.)

In Fig. 22 the unconditional environment is located at the head of a tunneled output arrow. For our purposes, this result is only of limited interest. Similarly, the auxiliary tools Node Marker, Node Attacher and Node Detacher are located at the tail of tunneled arrows, i.e., these mechanisms do not come from outside the diagram. Instead, they are assumed to be available within the diagram (this reduces the complexity of the SADT modeling approach). For the current approach, they only cover the simple functionality to setup or delete additional processes of the PPE which are needed to execute the application tasks. The unconditional environment and those tools become more important if several load management systems concurrently execute in the network. According to §2.2.4, LoMan is designed to manage the mapping of a single parallel application. In order to be extendible, we seperated the functionality to control the PPE from LoMan. The tools serve as entry points to incorporate mechanisms of negotiation for future extensions of LoMan. Nodes in the unconditional environment are available to other load management systems.

5.3.2 Reconfigure the actual execution environment

In §2.3.1 we stated that operations to reconfigure the actual execution environment introduce additional costs, i.e., the *configuration costs*. At this point we want to look into more details which activities are actually performed (see Fig. 23).

For simplicity, we assume that pairs of changing nodes are denoted by (n, m), where n is the number of the node to be attached while m is the number of node to be detached. Thus, it is easy to express which nodes are attached, detached or exchanged. Nodes which are added from the possible ExecEnv to the actual ExecEnv if for some n in the list $n \neq -1$. Nodes are detached from the actual ExecEnv if for some m in the list $m \neq -1$. Nodes are exchanged if for some pairs (n, m) in the list $n \neq -1$, $m \neq -1$ and $n \neq m$. Hence, the activities are:

1. The attaching node list guides the Node Attacher when LoMan will *reserve new nodes* [B.1]. Those nodes are not yet used within the possible set of nodes, but will be part of the enhanced set of nodes as soon as the activity is finished.

2. Again, the Node Attacher will be used to finally *setup the tasks exec. context* [B.2]. The context of the tasks describes all necessary processes of the parallel programming environment without which the tasks cannot execute. A new set commit is the output of the activity and will finally determine the new node set. In general, LoMan will now migrate the tasks between pairs of exchanging nodes. LoMan will continue with detaching the nodes as soon as the tasks are migrated.

3. The new set commit also forces the sequencing order for the next activity, i.e., to *resolve the tasks exec. context* [B.3] on the nodes which are no longer used. The Node Detacher removes all parts of the active PPE from the nodes which are listed in the detaching node list. The output is the reduced set of nodes.

4. An old set commit will finally determine the old node set. LoMan will also guide the Node Detacher to *release old nodes* [B.4] From the viewpoint of the application these operations are transparent. LoMan instead, will update its configuration history to prevent the node thrashing effect.

Especially the activities to setup and to resolve the tasks execution contexts will introduce the costs. One major difference exists between reservation and release of nodes and actually setting up and resolving the tasks execution contexts respectively. That is, the former activities control the components of LoMan, whereas the latter control the additional application components as introduced by the parallel programming environments.

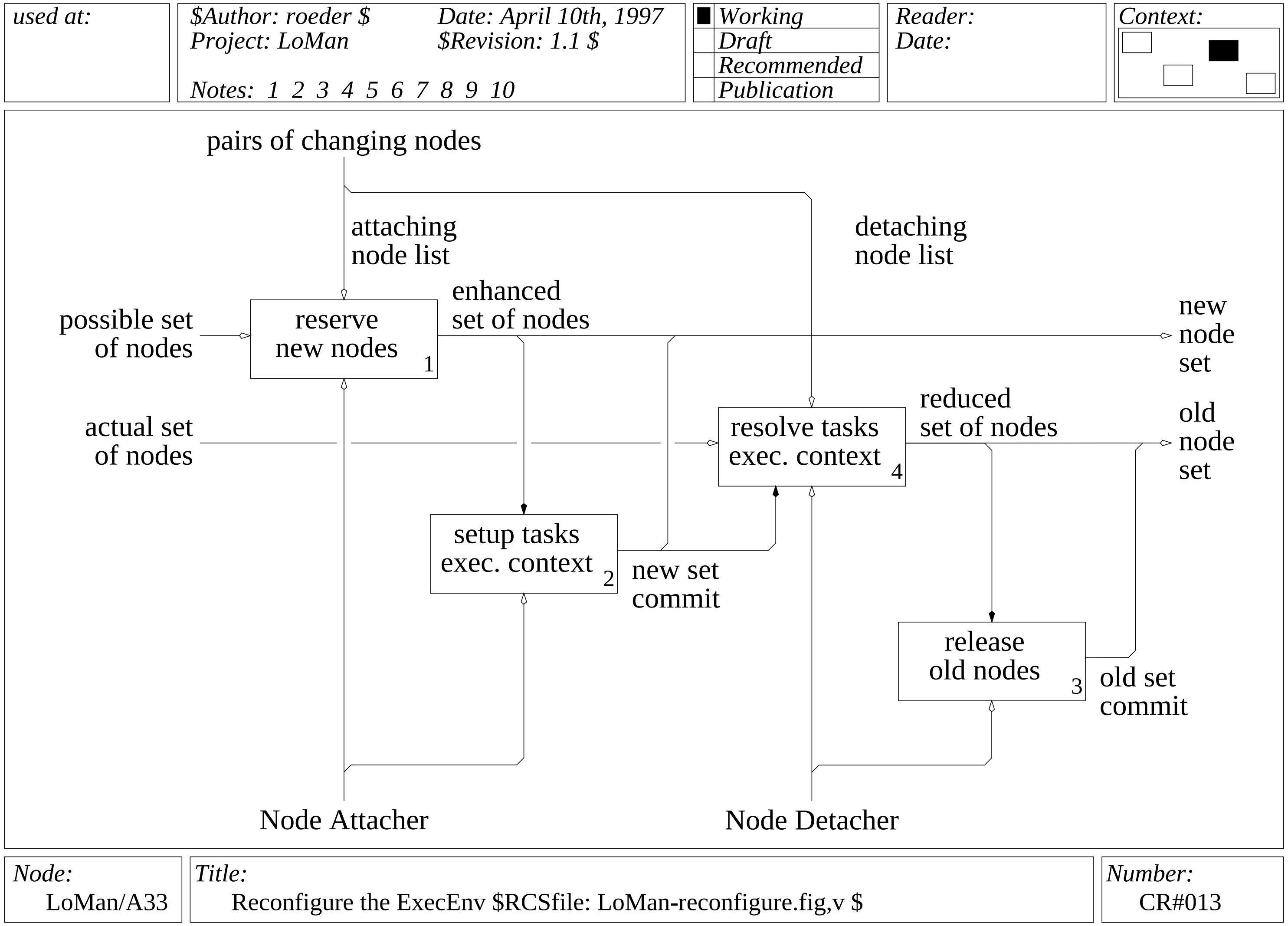


Figure 23: LoMan activity: reconfigure the actual execution environment

5.4 Analyzing the Constraints

This activity is responsible for evaluating the constraints imposed by the system and application. On the one side, the architecture and performance requirements of the application tasks have to be evaluated, and on the other side the external user demands might restrict the currently available set of nodes.

The functional decomposition of *analyze the constraints* at level 1 of the SADT hierarchy results in four activities which are illustrated in Fig. 24. Again, the next step directs the selection of one internal activity. Which activity is selected depends on the state of the application. We state that the boxes within the diagram are only loosely interrelated.

1. At the beginning of the management cycle, LoMan will *evaluate the architecture constraints* [B.1] as soon as the application is accepted and its architecture requirements (I-1) are known. The result is the list of nodes for load acquisition (O-1), i.e., what is later called the possible ExecEnv. Additionally, homomerous subsets of these nodes within the node sets for task migration are collected. We assume that process migration is only possible between homomerous nodes (see §2.2.1).

2. Not only the architecture requirements but also the resource preferences and limits (I-2) of the application tasks has to be considered when selecting possible target nodes. Therefore, LoMan will *evaluate the resource fitness* [B.2] according to the tasks' requirements and the nodes' performance and logical configuration (I-3). For example, preferences could guide LoMan to select the nodes which have local disk access and fast interconnection links. On the other side, it is possible that the user of the application has access restrictions to some nodes in the possible ExecEnv even if the performance requirements are met by those nodes. This information is part of the resource limits. The result will be a node ranking for the tasks (O-2) according to expected performance benefits.

3. Especially for the migration of tasks, the decision costs (O-3) might outweigh the expected performance benefits. LoMan has to *determine the costs of mapping* [B.3] to overcome this drawback by analyzing the number and size of tasks (I-4) to be migrated and the configuration costs (I-5). The decision costs will differ depending on the actually necessary steps. For example, the migration of tasks will be more expensive if the configuration of the actual ExecEnv has to be changed. This reconfiguration involves additional steps which would not be neccessary if migration is performed within an unchanged actual ExecEnv.

4. LoMan has to avoid conflicts with interactive users and hence, it has to *observe the users' node locking* [B.4]. LoMan will check for interactive usage (I-6) and the explicitly un-/locking of nodes (I-7). The result of this observation is a set of current node restrictions (O-4) which may reorder the node ranking for the tasks.

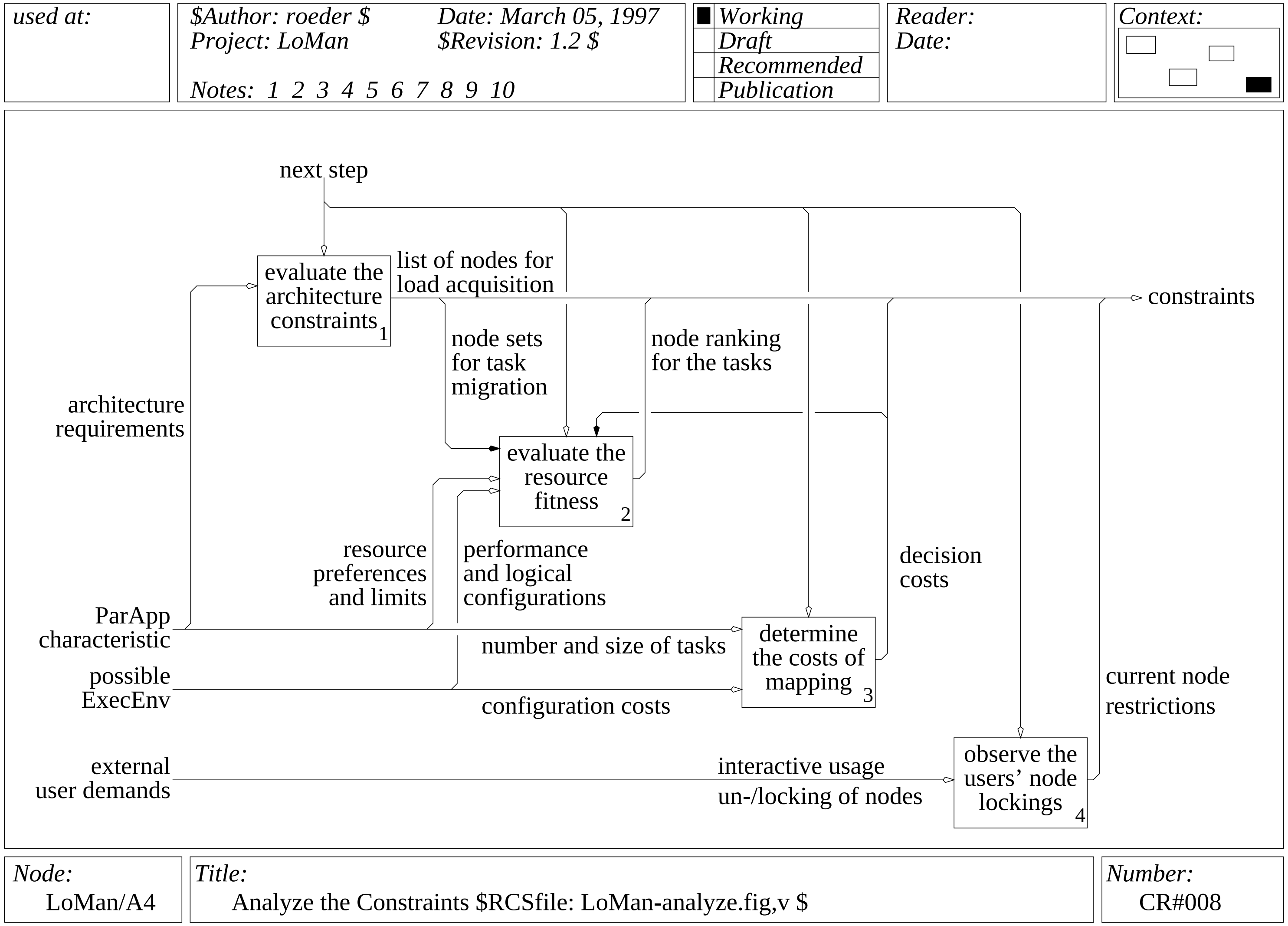


Figure 24: LoMan activity: analyze the constraints

5.5 Summary and Outlook

We conclude and summarize the SADT modeling approach with an overview of LoMan's activity hierarchy which is shown in Fig. 25. The hierarchy is given by stepping from the outer activities (darker gray color) to the inner activities (white color).

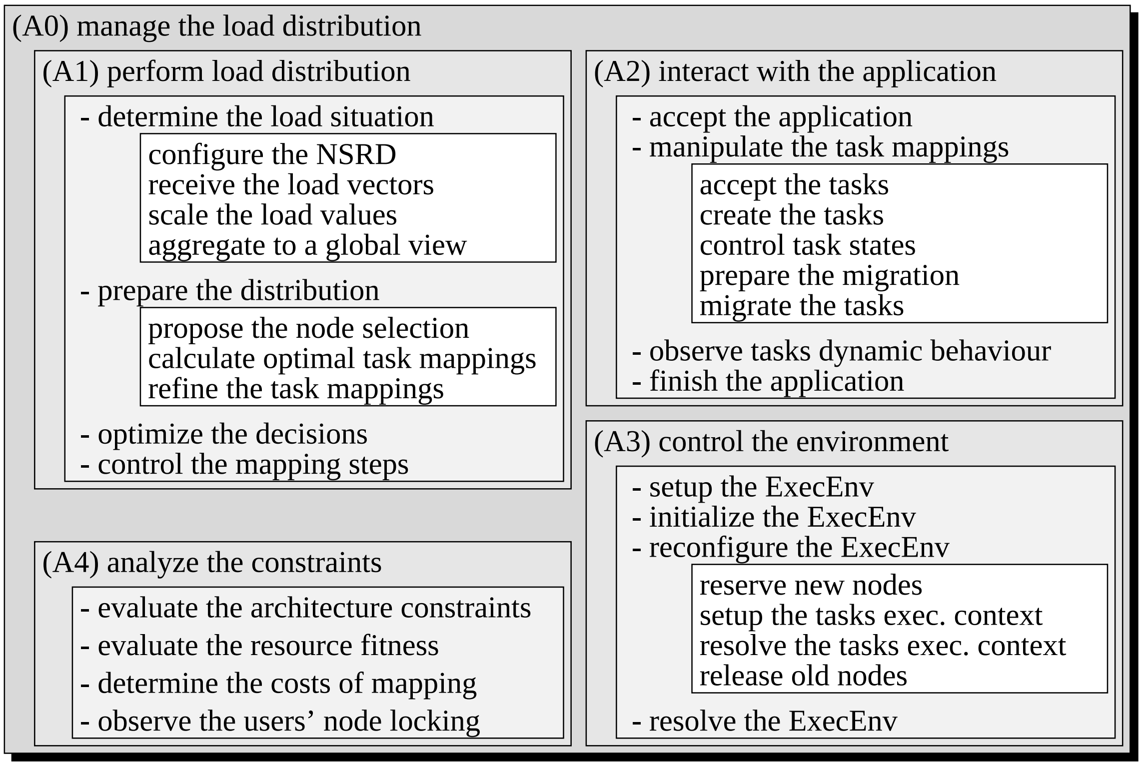


Figure 25: Hierarchy of LoMan activities

In §2.4 we listed the most important requirements which have to be met by LoMan. Each topic of this requirement list will be discussed and reviewed at according topics in the following list.

1. Parallel applications may dynamically create tasks during their runtime. LoMan merges the two phases of initial task placement and dynamic task redistribution. In §5.1.3 (*prepare the distributions*) it has been shown that LoMan considers the actual task mapping as soon as it has to *calculate heuristical task mappings*.

2. Heterogeneity is considered at two points. The architecture requirements (see §5.3.1: *setup the ExecEnv*) of the application restricts the execution environment to a subset of homomerous node sets. This subset may contain just a single node set if only a single executable version of the application exists. Additionally, the heterometric property of the system is considered in §5.4 as soon as LoMan has to *evaluate the resource fitness*. The performance indices are also used when LoMan has to *scale the load values* (see §5.1.2).

3. The separation of the possible ExecEnv and actual ExecEnv enables LoMan to distinguish the availabale nodes for two different purposes. The set of nodes in the actual ExecEnv is a subset of nodes in the possible ExecEnv[5]. Load inidices are measured on nodes of the possible ExecEnv, whereas tasks execute only on nodes of the actual ExecEnv.

4. The auxiliary tools Node Marker, Node Attacher and Node Detacher are entry points if negotiation has to considered (see §5.3.1). Negotiation (or similar techniques) are needed if several parallel applications are under control of load management systems. Future extensions of LoMan will provide mechanisms for coordinated access to the nodes in the system.

5. LoMan considers management costs when it optimizes the mapping decisions which are just based on load values (see §5.1.1: *optimize the decisions*). Management operations which are selected due to the current load situation are rejected if they do not meet the expected performance benefits.

6. Affinity scheduling is enabled since LoMan compares the resource requirements and preferences of the tasks with the configuration of the nodes (see §5.4). Nodes will be ordered and preferable selected according to those requirements.

Hence, the requirements which have to be considered during the design phase are fulfilled. As an byproduct, the strength of the SADT methodology has been shown during the design phase. As already mentioned in §3.3 the node numbers, the C-Numbers and the ICOM encoding scheme are used to review and perform syntax checking of the diagrams.

The next steps will be the specification of the data structures and functions used inside LoMan. Additionally, the distribution of its components (we state here, that an centralized decision component should be sufficient) has to be fixed.

[5] The two sets actual ExecEnv and possible ExecEnv may be identical

Acknowledgement

We would like to express our gratitude to Ursula Maier for long discussions and helpful suggestions which made the document more understandable. Stefan Petri provided us with thoughtfull criticism which allowed to produce the document in its current version. Finally, we want to thank Christian Trennhaus for reviewing and critical reading of the manuscript.

References

[1] V. A. F. Almeida, J. N. C. Arabe, E. F. Loures, and G. S. Rimolo. Scheduling Parallel Jobs on a Cluster of Heterogeneous Workstations. In *Proc. of the High Performance Computing Conference '94*, pages 103–108, Sep. 1994.

[2] T. E. Anderson, D. E. Culler, and D. A. Patterson. A Case of NOW (Networks Of Workstations). In *IEEE Micro*, volume 15 (1), pages 54–64, 1995.

[3] R. M. Butler and E. L. Lusk. *User' s Guide to the p4 Programming System.* Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4801, Oct. 1992.

[4] R. M. Butler and E. L. Lusk. Monitors, Messages, and Clusters: The p4 Parallel Programming System. *Parallel Computing*, 20(4):547–564, Apr. 1994.

[5] T. L. Casavant and J. G. Kuhl. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. In *IEEE Trans. Softw. Eng.*, pages 141–154, Feb. 1988.

[6] T. L. Casavant and J. G. Kuhl. Effects of Response and Stability on Scheduling in Distributed Computing Systems. In *IEEE Trans. Softw. Eng.*, volume 14 (11), pages 1578–1588, Nov. 1988.

[7] M. Cermele, M. Colajanni, and G. Necci. Dynamic Load Balancing of Distributed SPMD Computations with Explicit Message Passing. In *Proc. 6th Heterogeneous Computing Workshop*, pages 2–16. IEEE Computer Society Press, Apr. 1997.

[8] Intel Supercomputer Systems Division. Paragon Supercomputers. Technical report, Intel Corporation, 15201 N.W. Greenbrier Parkway, Beaverton, Oregon 97006, 1992.

[9] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive Load Sharing in Homogeneous Distributed Systems. In *IEEE Trans. Softw. Eng.*, volume SE-12, pages 662–675, May 1986.

[10] D. Ferrari and S. Zhou. An Empirical Investigation of Load Indices for Load Balancing Applications. In *Performance'87, 12th Int. Symp. on Computer Performance Modeling, Measurement and Evaluation*, pages 515–528, 1987.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, Oak Ridge, TN 37831-8083, 1. edition, May 1994.

[12] A. Goscinski. *Distributed Operating Systems - The Logical Design.* Addison-Wesley Publishing Company, Inc., Sidney, 1991.

[13] P. Hanks, editor. *The Collins English Dictionary (2nd ed.).* William Collins Sons & Co. Ltd., 1986.

[14] M. Hilbig. Design and Implementation of a Node Status Reporter in Networks of Heterogeneous Workstations, Dec. 1996. Fortgeschrittenenpraktikum am Lehrstuhl für Rechnertechnik und -organisation, Institut für Informatik, TU München.

[15] K. Jensen. Coulered Petri Nets: A High Level Language for System Design and Analysis. In *Advances in Petri Nets*, pages 342–416. Springer, LNCS 483, 1990.

[16] K. Jensen and G. Rozenberg, editors. *High-level Petri Nets: Theory and Applications.* Springer Verlag, Berlin-Heidelberg, 1991.

[17] W. Joosen, S. Bijnens, B. Robben, J.v. Oeyen, and P. Verbaeten. Flexible Load Balancing Software for Parallel Applications in a Time-Sharing Environment. In B. Hertzberger and G. Serazzi, editors, *High-Performance Computing and Networking – Europe*, pages 398–406. Springer, LNCS 919, May 1995.

[18] T. Kunz. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Sheme. In *IEEE Transactions on Software Engineering*, volume 17(7), pages 725–730, July 1991.

[19] Longman Group UK Limited, editor. *Dictionary of English Language and Culture.* Longman Group UK Limited, 1992.

[20] C. Lu and S-M. Lau. An Adaptive Load Balancing Algorithm for Heterogeneous Distributed Systems with Multiple Task Classes. In *16th Int. Conf. on Distributed Comp. Systems*, pages 629–636. IEEE Computer Society Press, May 1996.

[21] T. Ludwig. *Automatische Lastverwaltung für Parallelrechner.* Reihe Informatik, Band 94. BI-Wissenschaftsverlag, 1993.

[22] T. Ludwig, R. Wismüller, and M. Oberhuber. OCM — An OMIS Compliant Monitoring System. In *Parallel Virtual Machine — EuroPVM'96: Third European PVM Conference, Munich, Germany, October, 7-9*, pages 81–90, Berlin, Oct. 1996. Springer. Lecture Notes in Computer Science, 1156.

[23] T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. OMIS — On-line Monitoring Interface Specification. Technical Report TUM-I9609, SFB-Bericht Nr. 342/05/96 A, Technische Universität München, Munich, Germany, Feb. 1996.

[24] U. Maier and G. Stellner. Distributed Resource Management for Parallel Applications in Networks of Workstations. In *HPCN Europe 1997*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[25] D.A. Marca and C.L. Clement. *SADT – Structured Analysis and Design Techniques.* McGraw-Hill Company, Inc., 1989.

[26] Message Passing Interface Forum. MPI: A Standard Message-Passing Interface. Technical report, University of Tennessee, Knoxville, TN, Nov. 1993.

[27] H. Meuer and J. Dongarra. TOP500 Supercomputer Sites at Mannheim University. Technical report, Mannheim University and Oakridge National Laboratory, http://parallel.rz.uni-mannheim.de/top500, June 1996.

[28] Meyers, editor. *Enzyklopädisches Lexikon.* Bibliographisches Institut, Mannheim, 1974.

[29] K.H. Mortensen and V. Pinci. Modelling the Work Flow of a Nuclear Waste Management Program. In R. Valetta, editor, *15th Int. Conf. Application and Theory of Petri-Nets*, pages 376–395. Springer, LNCS 815, 1994.

[30] C. Pleier. Ein Verfahren zur Prozeßmigration in heterogenen UNIX Rechnernetzen. In *PIK — Praxis der Informationsverarbeitung und Kommunikation*, volume 18 (2), pages 75–81, Apr. 1995.

[31] F. Ramme and K. Kremer. Scheduling a Metacomputer by an Implicit Voting System. In *3rd IEEE Int. Symp. on High Performance Distributed Computing*, pages 106–113, San Francisco, 1994.

[32] C. Röder. Classification of Load Models. In T. Schnekenburger and G. Stellner, editors, *Dynamic Load Distribution for Parallel Applications*, pages 28–42. to be published, Spring 1997.

[33] C. Röder. Lastverwaltung auf Arbeitsplatzrechnern. In *Graduiertenkolleg Kooperation und Ressourcenmanagement in verteilten Systemen (Arbeits- und Ergebisbericht zum ersten Fortsetzungsantrag)*, volume TUM-I9707, 1997.

[34] A. L. Rosenberg. Needed: A Theoretical Basis for Heterogenous Parallel Computing. In U. Vishkin, editor, *Developing a Computer Science Agenda for High-Performance Computing*, pages 137–146. ACM Press, 1994.

[35] T. Schnekenburger. *Adaptive Lastverteilung für parallele Programme.* Dissertation, Techn. Univ. München, 1994.

[36] B. Schnor, S. Petri, and H. Langendörfer. Load Management for Load Balancing on Heterogeneous Platforms: A Comparision of Traditional and Neural Network Based Approaches. In *2nd Int. Euro-Par Conference*, pages 615–620. LNCS 1124, Springer, 1996.

[37] B. Schnor, S. Petri, R. Oleyniczak, and Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In K. Yetongnon and S. Hariri, editors, *9th Int. Conf. on Parallel and Distributed Computing Systems*, pages 330–337, 1996.

[38] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and Load Balancing in Parallel and Distributed Systems.* IEEE Computer Society Press, 1995.

[39] N. G. Shivaratri, P. Krueger, and M. Singhal. Load distributing for locally distributed systems. *Computer*, 25(12):33–44, Dec. 1992.

[40] S. Steinmetz et al., editors. *Langenscheidt's Compact Unabridged Dictionary.* Random House, NY, 2nd edition, 1996.

[41] G. Stellner. Using CoCheck on a Network of Workstations. SFB-Bericht 342/17/95 A, Technische Universität München, D-80290 München, Nov. 1995.

[42] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the International Parallel Processing Symposium*, pages 526–531, Honolulu, HI, Apr. 1996. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264.

[43] G. Stellner. *Methoden zur Sicherungspunkterzeugung in parallelen und verteilten Systemen*, volume 2 of *Research Report Series LRR-TUM.* Shaker Verlag, 1996.

[44] G. Stellner, A. Bode, S. Lamberts, and T. Ludwig. Developing Application for Multicomputer Systems on Workstations. In W. Gentzsch and U. Harms, editors, *High-Performance Computing and Networking, International Conference and Exhibition, Volume II*, number 797 in LNCS, pages 286–292. Springer, Apr. 1994.

[45] G. Stellner, A. Bode, S. Lamberts, and T. Ludwig. *NXLib Users' Guide.* Technische Universität München, Institut für Informatik, D-80290 München, v1.1.2 edition, May 1994.

[46] S. Subramaniam and D. L. Eager. Affinity Scheduling of Unbalanced Workloads. In *Proceedings of the Supercomputing'94*, pages 214–226, Nov. 1994.

[47] V. Sunderam, A. Geist, J. Dongarra, and R. Manchek. The PVM Concurrent Computing System: Evolution, Experiences and Trends. *Parallel Computing*, 20(4):531–545, Apr. 1994.

[48] M. M. Theimer and B. Hayes. Heterogeneous Process Migration by Recompilation. In *11th Int. Conf. on Distributed Computing Systems*, pages 18–25. IEEE, 1991.

[49] S. W. Turner, L. M. Ni, and B. H. C. Cheng. Time and/or Space Sharing in a Workstation Cluster Environment. In *Proc. Supercomputing'94*, pages 630–639, Nov. 1994.

[50] D. W. Walker. The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers. *Parallel Computing*, 20(4):657–673, Apr. 1994.

[51] A. Weinrib and S. Shenker. Greed is not enough: Adaptive Load Sharing in Large Heterogeneous Systems. In *Proc. of the IEEE INFOCOM*, 1988.

[52] X. Zhang and Y. Yan. A Framework of Perfomance Prediction of Parallel Computing on Nondedicated Heterogeneous NOW. In *Int. Conf. on Parallel Processing*, volume I, pages 163–167, 1995.

[53] Y. Zhang, K. Hakozaki, H. Kameda, and K. Shimizu. A Performance Comparision of Adaptive and Static Load Balancing in Heterogenous Distributed Systems. In *Proc. 28th Simulation Symposium*, pages 332–340. IEEE Comp. Soc. Press, 1995.

Index

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs

Reihe A

342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen
342/18/90 A	Christoph Zenger: SPARSE GRIDS
342/19/90 A	Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems
342/20/90 A	Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids
342/21/90 A	V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results
342/22/90 A	Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions
342/23/90 A	Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement
342/24/90 A	Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm
342/25/90 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)
342/26/90 A	Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)
342/27/90 A	Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems
342/28/90 A	Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras
342/29/90 A	Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics
342/30/90 A	Michael Griebel: Parallel Multigrid Methods on Sparse Grids
342/31/90 A	Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits
342/32/90 A	Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs
342/33/90 A	Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes
342/1/91 A	Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?
342/2/91 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets
342/3/91 A	Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

Reihe A

342/4/91 A	Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity
342/5/91 A	Robert Gold: Dataflow semantics for Petri nets
342/6/91 A	A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften
342/7/91 A	Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions
342/8/91 A	Walter Vogler: Generalized OM-Bisimulation
342/9/91 A	Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen
342/10/91 A	Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System
342/11/91 A	Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen
342/12/91 A	Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken
342/13/91 A	Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage
342/14/91 A	Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines
342/15/91 A	Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?
342/16/91 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS
342/17/91 A	Ulrich Furbach: Answers for disjunctive logic programs
342/18/91 A	Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs
342/19/91 A	Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme
342/20/91 A	M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover
342/21/91 A	Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids
342/22/91 A	Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems

Reihe A

342/23/91 A	Astrid Kiehn: Local and Global Causes
342/24/91 A	Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine
342/25/91 A	Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition
342/26/91 A	Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch
342/27/91 A	Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C
342/28/91 A	Claus Dendorfer: Funktionale Modellierung eines Postsystems
342/29/91 A	Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems
342/30/91 A	W. Reisig: Parallel Composition of Liveness
342/31/91 A	Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments
342/32/91 A	Frank Leßke: On constructive specifications of abstract data types using temporal logic
342/1/92 A	L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI
342/2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS
342/2-2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)
342/3/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/4/92 A	Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS
342/5/92 A	Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstützung paralleler und verteilter Programmierung
342/6/92 A	Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications
342/7/92 A	Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study
342/8/92 A	Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement

Reihe A

342/9/92 A	Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp
342/10/92 A	H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems
342/11/92 A	M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks
342/12/92 A	Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions
342/13/92 A	Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme
342/14/92 A	Michael Griebel: Grid– and point–oriented multilevel algorithms
342/15/92 A	M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems
342/16/92 A	J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme
342/17/92 A	Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen
342/18/92 A	Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS
342/19/92 A	Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries
342/20/92 A	Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets
342/21/92 A	Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids
342/22/92 A	Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken
342/23/92 A	Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency
342/24/92 A	T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity
342/25/92 A	Ekkart Kindler: Invariants, Compositionality and Substitution
342/26/92 A	Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs
342/1/93 A	M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique
342/2/93 A	Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents
342/3/93 A	Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments

Reihe A

342/4/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation

342/5/93 A Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals

342/6/93 A Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms

342/7/93 A Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits

342/8/93 A Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving

342/9/93 A Peter Slavkovsky: The Visibility Problem for Single-Valued Surface ($z = f(x,y)$): The Analysis and the Parallelization of Algorithms

342/10/93 A Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods

342/11/93 A Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods

342/12/93 A Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination

342/13/93 A Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids

342/14/93 A Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation

342/15/93 A Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms

342/16/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation

342/17/93 A Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib

342/18/93 A Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component

342/19/93 A Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems

342/20/93 A Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism

Reihe A

342/01/94 A Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation
342/02/94 A Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency
342/03/94 A Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router
342/04/94 A Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2
342/05/94 A Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations
342/06/94 A Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems
342/07/94 A Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach
342/08/94 A Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls
342/09/94 A Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits
342/10/94 A Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style
342/11/94 A Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus
342/12/94 A Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids
342/13/94 A Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems
342/16/94 A Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization
342/19/94 A Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method
342/20/94 A Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -

Reihe A

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control

Reihe A

342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project
342/15/96 A	Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus

Reihe A

342/16/96 A	Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A	Richard Mayr: Model Checking PA-Processes
342/18/96 A	Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States
342/01/97 A	Tobias M"uller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf"ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken
342/02/97 A	Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A	Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop
342/04/97 A	nicht erschienen
342/05/97 A	Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A	Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A	Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A	Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A	Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A	Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A	Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A	Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A	Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A	Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A	Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A	Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung
342/17/97 A	Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing
342/18/97 A	Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

Reihe A

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS