Reduced Design Time by Load Distribution with CAD Framework Methodology Information*

Jürgen Schubert[1], Arno Kunzmann[1], Wolfgang Rosenstiel[2]

[1]Computer Science Research Center at the University of Karlsruhe (FZI)
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany
[2]FZI and University of Tübingen, Sand 13, D-72076 Tübingen, Germany

Abstract

This paper is focused on reducing the design time in a CAD framework environment by the optimal use of resources. A user-transparent load distribution system (Framework based LOad DIstribution System - FLODIS) is presented that enables a distributed execution of design tools over a heterogeneous network of workstations. The basic idea of the proposed algorithm is to allocate tools to workstations by using estimations about the required resources. These values can be obtained by user-defined tool characteristics, recorded previous tool executions and by methodology information provided by the framework. Compared with standard distribution approaches that use less information about the activities, experimental data show a significant average reduction (40-50%) of the tool execution time.

1 Introduction

Today CAD designs are typically very complex, and therefore, a lot of design tools are needed and different designers work concurrently on one project. Frameworks are used to solve the problem of concurrent engineering and to manage these large projects [7]. Hereto, the design tools have to be integrated or encapsulated in the framework. Depending on the parameters and the selected input/output data type, every tool can perform different activities. The design methodology is modelled by a design flow which is defined by the methodology manager. A design flow consists of activities and their dependencies.

The activities are typically executed on a heterogeneous net of workstations having different computing power. In a CAD environment, the critical activities typically require an execution time ranging need from several minutes up to several hours. Also, it has to be taken into account that not all activities can be started on every machine. The execution time - the elapsed time period between the start and the end of the tool execution - increases if more than one activity has been started on one workstation. Therefore, an intelligent distribution of activities on hosts is required to reduce the overall execution time, hereby optimizing the use of available resources.

Many approaches of load distribution can be found in the literature (e.g., [12][14]). One general problem of most distribution algorithms is the missing knowledge about the upcoming load. Static approaches use statistics, for instance, fixed job arrival rates. The distribution is calculated without measuring the load on the hosts, assuming an average external load on the hosts [14]. Dynamic load distribution approaches use the current system state, which is measured at the moment when an activity has to be executed to select the least loaded host [12]. However, these approaches do not take the information about loads of previously executed activities into account.

In a framework environment, several load distribution approaches were reported: The MCC CAD framework selects the best suitable workstation based on a ‘Round Robin‘ algorithm [1]. FLOW is a dynamic load distribution system. Depending on the needed static activity resources, FLOW will select the workstation with the least load [8]. PAPYRUS is based on the operating system SPRITE and, therefore, provides easy process migration [4]. In the ULYSSES II framework, resources are dynamically allocated [3]. Here, the appropriate workstation is selected by calculating the processing power and the work load.

The calculation of all these approaches is only based on the number of activities. But obviously, the required resources (e.g., CPU-time, main memory) are essential to improve the allocation of activities. In the presented new approach, activity specific resources are taken into account.

The accessible network of workstations may be heterogeneous with hosts of different computing power. It is assumed that all the load on this network is produced by the integrated CAD tools of the framework and controlled by the load distribution system. Different designer can start activities at any time from any host.

* This work was supported by ESPRIT Project 7364

The presented algorithms are independent of the topology of the network [12]. For an Ethernet it is assumed that the communication time between all workstations is the same. As further boundary condition in this approach process migration is not used, since the overhead to stop a CAD tool, transfer the data and continue the work on another host is very high.

This work is focused on the distribution of tools which are encapsulated in a framework. Other work can be found to perform a parallel execution of the activity itself but this requires the source code of the tool [2].

In the following, the profile of activities is discussed and the way how to calculate the estimated load is presented. Section 4 presents a new load distribution algorithm which uses these estimated load values to provide an optimized allocation of activities. Section 5 describes the architecture of the load distribution system FLODIS. Experimental results with several CAD tools point out the achieved improvements. The summary and the future plans show the extendability of this system to other domains.

2 Estimation of activity-specific resources

If there is more than one activity running on one single processor workstation at the same time, the execution time of all these activities grows. This is caused by the necessity to share CPU and I/O resources of the workstation. But this increase of the execution time heavily depends on the combined types of activities. For example, the combination of a CPU-intensive with an I/O-intensive activity results in a lower time increase compared with two CPU-intensive activities on the same machine.

Figure 1 shows the protocol of a perfmeter on a SPARC 2 where a FPGA was designed. This design requires a partitioning and a state-encoding tool. The X-axis, representing the time shows the execution order of these tools with a simultaneous execution of both tools at the end. The upper part shows the used CPU percentage, in the lower part the number of disk accesses is protocolled. As indicated, the partitioning tool requires 100% CPU and only a few disk accesses. The state encoder shows an average CPU usage of about 50% CPU along with a significant number of disk accesses. To find the exact I/O usage also other aspects (e.g., paging) must be taken into account. In Unix systems this information can be found in the protocol of the in-/out blocks (e.g., using the ‘rusage‘ command). If both tools are executed at the same time on one host, the partitioning tool requires twice as much execution time while the state encoder execution time does not change. The operating system assigns for both tools 50% of the CPU computing time. While the partitioning tool uses the CPU, the state encoder makes I/O operations.

The activities and resources can be modelled as follows: The host provides two resources, namely the CPU and the I/O resource. An activity needs a specific percentage of the CPU and the I/O distributed over the time. In this model the mean value of the resource usage is used to estimate the overall execution time of an activity. One tool uses only one resource at the same time, no simultaneous CPU and I/O usage are assumed. This model allows to calculate the execution time of activities running in parallel, if the relative resource usage and the execution time is known for each activity determined exclusively on one host.

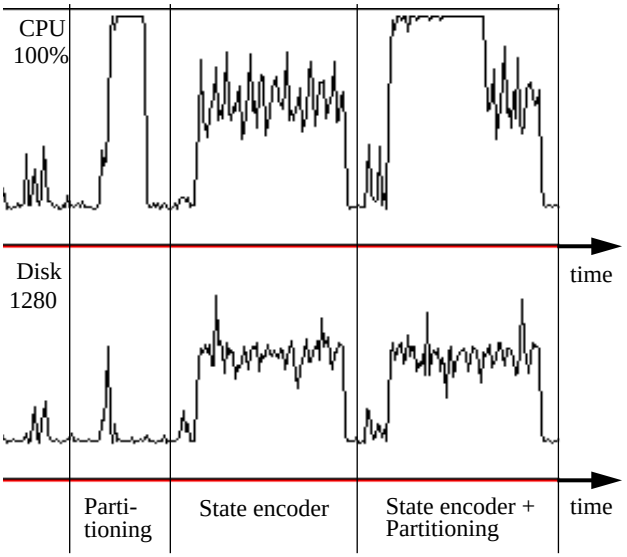

Figure 1: Perfmeter protocol of FPGA partitioning and state encoder

The described model has been validated by a number of typical CAD tools including a testpattern generator, logic simulators, state encoders, FPGA partitioning and placement algorithms which are offered by different design systems (e.g., MENTOR, LOGIC, MIGRATE, XILINX).

Figure 2 presents a subset of these tools which have been evaluated in more detail. The times have been measured with the UNIX ‘time‘-command, and all the tools have been executed exclusively on one host. The highest ratio between CPU usage (CPU time) and execution time is 98% for the analyzed testpattern generator while the state encoder shows the lowest ratio with 48%. In the latter case a lot of I/O resources are used while the CPU can be used by other processes.

The CPU and I/O profile of an activity depends on several parameters, especially on the algorithm, the execution parameters and the input data. Obviously, for each activity both, the algorithm and the parameters are fixed. Therefore, activities can be classified in CPU intensive or I/O intensive. One could conclude that this classification is valid for all activity executions. However, there is an influence of the (sub-) project specific data and therefore, the same activity can have different CPU and I/O characteristics. The follow-

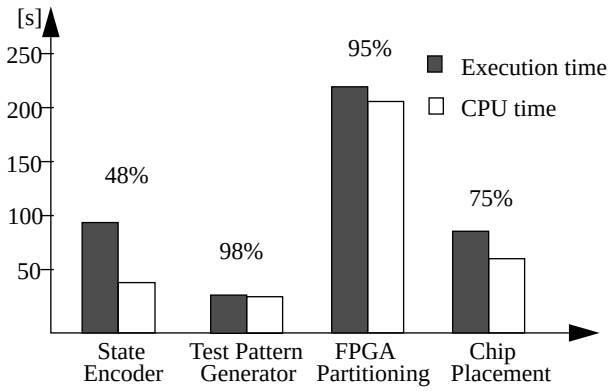

Figure 2: Execution times and CPU-time relation

ing subsections present three different methods to specify the data dependent relation between CPU and I/O time.

2.1 Information from tool integrator

The relationship between the data and the profile of the activity typically depends on several complexity dimensions (CD), for instance, the gates, transistors or the primary inputs. Often, the tool integrator knows the dependencies between CD value and execution time (e.g., the execution time for test pattern generation grows cubically with the number of gates on average). The tool integrator should not use a worst case estimation but the most typical relationship. The CD values can be found in the database or by analyzing the input files. In [3] CD values are used to predict a theoretical CPU time value. If several tools can be used for the same design task, the tool with the least CPU time requirement is selected. In this paper, and in contrast to [3], CD values are used to allocate tools to workstations.

2.2 Information from activity protocol

With every activity execution, the used resources are recorded and stored together with the CD value. This leads to a list of CD values and the required resources (CPU time, I/O time and main memory). Based on the recorded CD values, the expected amount of needed resources can be calculated using mathematical approaches. In FLODIS, the method of linear regression is used to forecast CPU time, I/O time and main memory requirements. The coefficients of the straight line are estimated by the method of minimum squares [5].

Assuming that only a few values are available the linear regression might not provide good results. But in general, a load distribution system is robust against a few incorrect estimated resources of a job. Additionally, with every execution of the activity, a better resource estimation is available.

For a non-linear relationship it is possible to use a linear transformation, which can also be found in [5]. To store the data in the data base related with the activities will not significantly increase the access time. The meta data which are protocolled have the size of some bytes so even thousands of activity executions result in a few kilobytes of meta data.

2.3 Information from methodology management

The results of the linear regression enable an efficient distribution of activities. However, needing little calculation effort, further improvements can be achieved in a framework environment. If an activity is repeatedly executed within the same subproject, the CD values are expected to be similar. Therefore, the currently needed resources are the mean value of the previous resource requirements. Also, it is possible to include the information of a designer reexecuting an activity within the same design flow. This is typically the case for incremental electronic CAD developments, including debugging and several optimization and verification phases. In this situation, the input data are only slightly modified. Therefore, the resource requirements usually are very similar compared with the last activity execution.

3 Resource estimation algorithm

Based on the three presented variants to estimate the required resources, a combined estimation algorithm was developed and implemented in FLODIS (Figure 3).

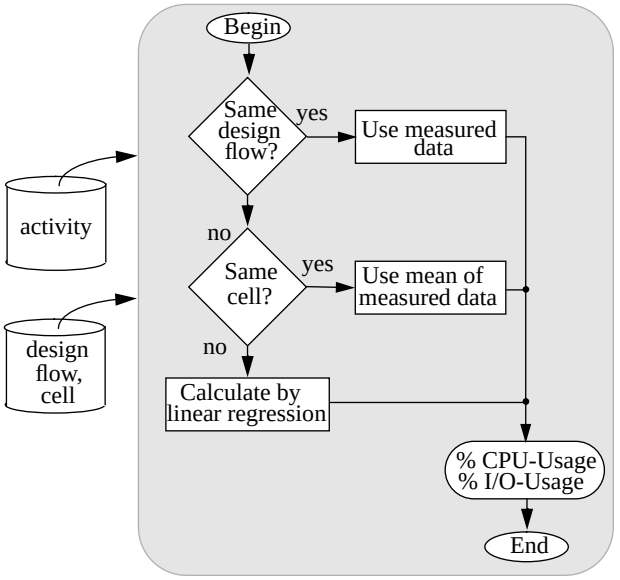

Figure 3: Algorithm for resource estimation

The attributes of the activity object are checked for the number of the executed design flow. If an activity is executed in the same flow a second time, the previous measured resource values are used. If the activity was already executed within the same subproject, the mean value is calcu-

lated and used. In the JESSI-COMMON-Framework [11], this information is provided by the design management component. Here, the same cell (subproject) version means that the same design flow is used. However, the resource estimation can be found in every framework where information about design flows and subprojects can be found.

If the activity is used for the first time in a new subproject, the resource requirements are calculated by using the linear regression (cf. Section 2.2). Based on the resulting CPU and I/O ratio of the activity, the allocation algorithm can select a workstation for efficient activity execution.

4 Allocation on resources

The aim of the distribution system is to find an allocation of activities on a network of workstations such that the overall execution time is minimized. The efficiency of an allocation algorithm heavily depends on the information which is available about activity related resources. Such an allocation algorithm has to provide good results by keeping the extracted and recorded information manageable. In the last chapters it has been described how activity specific estimations about the percentage of CPU and I/O resources can be determined with low effort. However, other allocation algorithms use less information. In this paper two of those algorithms are presented and compared to the FLODIS allocation algorithm which uses the information about the percentage of the CPU and I/O resources.

Round robin: Using a round robin algorithm, a minimum of information is needed. The algorithm has to identify the available hosts and has to know the fact that an activity is ready to be executed. The activities are allocated to the hosts one after another.

Class allocation: For this allocation algorithm the activities are classified by the tool integrator in CPU intensive or I/O intensive tasks. Therefore, the protocol of the activity execution is not needed for repetitive activity executions. The idea of this allocation is to avoid the execution of two activities which belong to the same class on the same host. The hosts are ordered by their computing power and the most powerful host not executing an activity of the same class is selected. However, an activity can skip into the other class depending on the input data. In this case, the algorithm may allocate activities on the same host which are both CPU intensive (or I/O intensive). This could only be avoided if the protocol of the activity execution is available.

FLODIS allocation: The allocation procedure is based on the needed percentage of CPU and I/O time. Additionally, memory resources are taken into account. The distribution algorithm uses a list of hosts which is ordered according to the available computing power. The algorithm always starts at the top of the list to find a workstation which provides the previously estimated resources for the activity. The value of the available resources of this workstation is decreased by the value of the activity. For the next activity, the algorithm starts again at the top of the list, but the last selected host now may not have the needed resources available. However, this algorithm guarantees that the powerful machines will be used more often than others. It is based on the fact that the execution on a powerful host that already executes an activity of different activity profile, is faster than the selection of the least loaded but less powerful host.

To take workstations with different computing power into account, transfer factors are calculated in advance for adapting CPU and I/O time estimations on the selected workstation. These transfer factors represent the relation of computing power on the selected host and a reference host. For both, the CPU and I/O resource, such a factor is necessary. Until now the factors are calculated using a set of typical CAD tools, and a linear relation between the hosts is assumed. However, it is also possible to take non linear and activity specific transfer factors into account. For a heterogeneous environment, a reference host must be used for each CPU architecture.

5 FLODIS system architecture

Until now, the load distribution system FLODIS is a domain-neutral tool, but it may also be a part of the design management of the framework. The different hosts which are available in the network are known to the system.

As described in the previous sections, the distribution system provides an estimation of the activity resources and an allocation of the activities on the different workstations. To fulfil these tasks, different subtasks have been defined that are realized by the related modules of the FLODIS architecture (Figure 4). This architecture is very flexible and modular which is very important for controlling the system and for extensions (e.g., other domains).

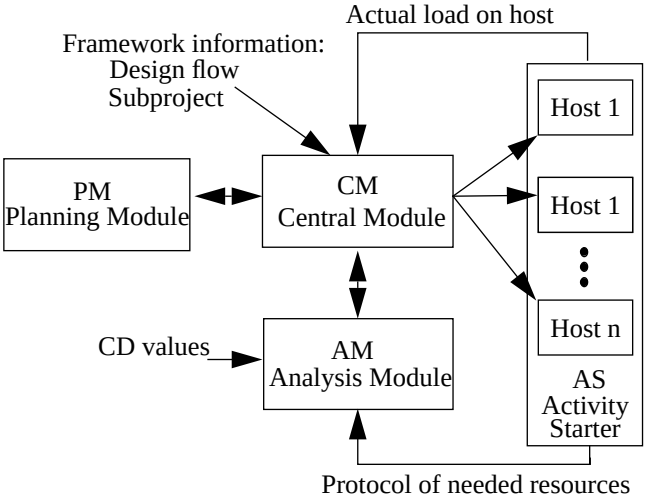

Figure 4: FLODIS architecture

The **central module** provides the communication between all the modules [10]. Several services are offered to other modules to get information even if they are distributed over the network.

The activities are started via inter-tool communication between the central module and the **activity starter** on the different hosts. The activity starter executes the activity on the selected host. The current load of this host is reported to the central module to update the system state information. As a second task of the activity starter, the needed resources of the executed activity are reported to the analysis module after the successful execution.

The **analysis module** has the task of estimating the needed resources for an activity as presented in Section 3. The used design flow, the subproject and the complexity of input data can be provided by the framework.

The **planning module** information is provided about the activity profile to plan the load distribution on the workstations as presented in Section 4. Until now, three presented allocation algorithms are realized as planning module of FLODIS, using more or less the available information about the activities [15]. The planning algorithms are based on a static load distribution. However, it is possible to send the actual load of the hosts to the central module. In case of an overload on the selected host which is not caused by a framework controlled activity, another host can be chosen.

The implementation of FLODIS was done in the object oriented language C++. For the inter-process communication, the C++ functions of Base Communication Module (BCM) [6] have been used. It is planned to integrate the load distribution system [13] in the JESSI-COMMON-Framework which will provide an open C++ interface to the design management and the data base.

6 Experiments and results

In the selected experimental environment FLODIS has been used to control the execution of two, four and six design flows in parallel (Figure 5; n,m = 1,2,3). One part of the design flows consists of three subsequent activities for FPGA partitioning (mapping, clustering, partitioning). The other flows consist of an activity for state encoding and an activity for minimizing the state table. The flows represent different designers using the activities with different data. Six SUN workstations with the same CPU architecture but different computing power are used (1×Sparc 20, 1×Sparc 10, 2×Sparc 2, 2×Sparc 1) in the test environment.

Both the mean execution time of the different flows and the overall execution time have been measured. Table 1 and Table 2 show the results for two ‘standard approaches‘ (Round Robin algorithm and Dynamic Load Distribution), the classification algorithm and the FLODIS algorithm. The Round Robin and the classification algorithm have been implemented as a planning module in the FLODIS system,

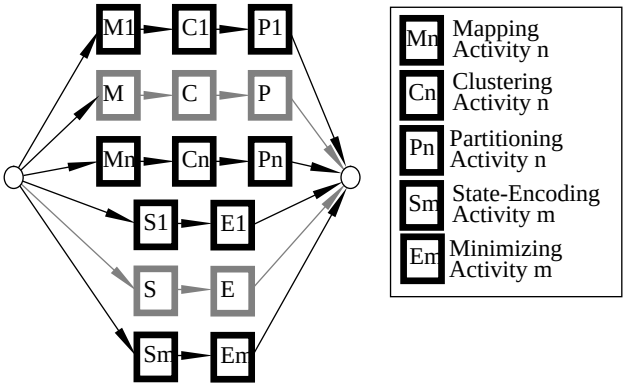

Figure 5: Evaluated design flows

for the Dynamic Load Balancing the tool ‘LB‘ has been used [9]. This tool allocates a workstation based on its actual load and a selection factor which corresponds to the computing power of the machine. The actual load is measured as load average during the last minute. For the selection factor the same values as used for the transfer factors of the FLODIS allocation have been selected. LB uses the „rsh“ command to perform the activities on other workstations. To avoid the influence of the communication overhead caused by this time consuming daemon, the execution time of this load balancer has been measured locally. In the measurements of the other allocation algorithms all communication overhead is included in the presented results.

The Round Robin algorithm distributes activities to every workstation. Also, the workstation with the least computing power has to perform activities, and if two activities are executed on one host both activities may have the same resource profile. Therefore, this results in execution times that are distinctly longer than the FLODIS results for 2, 4 and 6 flows in parallel. As expected, the Dynamic Load Balancing provides better results, but FLODIS is still the better alternative with an execution time reduction of about 40% to 50%. This result is mainly caused by the fact that dynamic approaches allow an allocation of activities with the same profile on the same host, i.e. the decision to execute an activity on a host is only based on the number of processes on a workstation. Another major problem of this approach is that the measurement of the load can be performed right in the moment when a very short process is using a lot of computation power. To avoid this, it is possible to measure the load average over a period of time, but high loads can be measured, caused by recently finished activities in the design flow. In both situations, a powerful workstation may not be selected although there is no process running.

The classification algorithm provides good results especially for activities with short execution time (flow 1, flow 5). However, the FLODIS algorithm can improve the overall execution time by over 10% compared with the classification algorithm. This is mainly caused by the differentiated view on the resource usage of the activities, e.g., two I/O

.

Algorithm	Flow 1	Flow 2	Σ	Flow 1	Flow 2	Flow 3	Flow 4	Σ
Round robin	100,17	115,31	215,48	112,66	106,86	266,99	224,9	711,41
Dynamic(LB)	68,15	105,2	173,35	88,96	101,47	230,23	205,05	625,71
Class	55,78	63,36	119,14	51,08	88,38	112,83	131,46	383,75
FLODIS	49,24	58,5	107,74	47,77	57,51	113,29	104,05	322,62

Table 1: Execution time of two and four design flows [s]

Algorithm	Flow 1	Flow 2	Flow 3	Flow 4	Flow 5	Flow 6	Σ
Round robin	133,12	146,75	233,95	217,38	149,32	129,8	1010,32
Dynamic(LB)	103,88	144,21	213,77	252,27	75,8	128,66	918,59
Class	68,99	118,8	130,5	159,38	56,58	88,54	622,79
FLODIS	77,53	88,86	121,81	113,51	63,43	83,27	548,41

Table 2: Execution time of six design flows [s]

intensive jobs may not exceed a resource limit on a more powerful workstation. Additionally, depending on the data, an activity can switch to another class which can not be taken into account without activity profiles and resource estimation (e.g., state encoder in flow 4).

The measurements show that an efficient execution of CAD tools can be achieved if the activity profiles are taken into account. It should be noted that the CPU time for the load distribution algorithm itself is distinctly below one second, and it therefore can be neglected.

7 Summary and future work

The presented load distribution system FLODIS is based on activity profiles. This information is used to minimize the execution time of all design flows controlled by and started via a framework. It could be shown that every activity has a specific relative usage of CPU and I/O with respect to the overall execution time. This profile can be estimated based on the methodology information of the framework or on statistic calculations. The main idea of the presented FLODIS load distribution approach is the combined execution of activities with different CPU and I/O usage on the same host. The experimental results show that FLODIS significantly reduces the necessary execution time compared with other allocation algorithms that use less information about the activities. The presented architecture of the load distribution system can optionally be integrated into a CAD framework, but FLODIS can also be used as a domain neutral tool.

As all the executable design flow is sent to the load distribution system, FLODIS has information about activities which will be executed in the future. This allows to add priority strategies to the FLODIS allocation algorithm. Further improvements can be achieved if the available design flows are used to calculate the start time of future activities.Therefore, future work will also include the development of a look-ahead distribution algorithm.

In a CAD environment a large number of tools is interactive. Therefore, the execution of interactive tools on the same host with tools of other resource profile should be advantageous, but has to be examined in more detail.

References

[1] W. Allen, D. Rosenthal, K. Fiduk: *"Distributed Methodology Management for Design-in-the-Large"*, ICCAD Santa Clara, 1990, pp. 346-349

[2] T. Bubeck, W. Rosenstiel: *"Verteiltes Rechnen mit DTS (Distributed Thread System)"*, SIPAR Workshop, CH-Fribourg, 1994

[3] M. Bushnell et al.: *"Distributed Computing, Automatic De-sign and Error Recovery in the ULYSSES II Framework"*, EuroDAC, Paris, France, March 1994

[4] T. Chiueh, R. Katz: *"A Distributed Transaction Facility for Design Task Management"*, EuroDAC, Paris, March 1993

[5] N. R. Draper, H. Smith: *"Applied Regression Analysis"*, New York, 1966

[6] J. Friedrich et al.: *"BCM Reference Manual"*, January 1992

[7] D. Harrison et al.: *"Electronic CAD Frameworks"*, Proceeding of the IEEE, Vol. 78, No. 2, February 1990

[8] Y. Kashai: *"Flow - A Concurrent Methodology Manager"*, EDAC, Brussels, Belgium, March 1992

[9] D. Kassabian, T. Soyata: *"LB - The Load Balancer"*, Dept. of Electrical Engineering, University of Rochester, April 1994

[10] S. Kirschke, *"Aufbau eines modularen, planenden Lastverteilungssystems für den Einsatz in Frameworks"*, Master thesis, Universität Karlsruhe, January 1994

[11] D. Liebisch, A. Jain: *"JESSI-COMMON-Framework Design Management - The means to Configuration and Execution of the Design Process"*, EuroDAC, Hamburg, 1992

[12] L. Ni, C. Xu, T. Gendreau: *"A distributed drafting Algorithm for load balancing"*, IEEE transactions on software engineering, Vol. SE-11, No. 10, October 1985, pp. 1153-1161

[13] J. Schubert, A. Kunzmann: *"Resource-Oriented Load Distribution in a Framework Environment"*, 4th International Working Conference on Electronic Design Automation Framework, Gramado , Brazil, November 1994

[14] A. Tantawi, D. Towsley: *"Optimal static load balancing in distributed computer systems"*, Journal of the Association for Computing Machinery, Vol. 32, No. 2, pp. 445-465, April 1985

[15] C. Weiler, *"Lastverteilung von CAD-Werkzeugen unter Einbeziehung von Framework-Informationen"*, Master thesis, Forschungszentrum Informatik, November 1993