Weckstén, M., M. Jonsson, and J. Vasell, "Derivation of implementation constraints – implementation simulation and treatment of multiple design choices," Proc. of the 9th biennial SNART Conference on Real-Time Systems (Real-Time in Sweden – RTiS'07), Västerås, Sweden, Aug. 21-22, 2007, pp. 21-28.

Derivation of Implementation Constraints — Implementation Simulation and Treatment of Multiple Design Choices

P. Mattias Weckstén[†], Magnus Jonsson[†], and Jonas Vasell[‡]

[†] *Centre for Research on Embedded Systems (CERES)*
Halmstad University, Halmstad, Sweden
mattias.wecksten@ide.hh.se, magnus.jonsson@ide.hh.se

[‡] *Generic Systems Sweden AB, Stockholm, Sweden*
jonas.vasell@generic.se

***Abstract*— The industrial use of ad hoc implementation methods for non-functional constrained tasks has resulted in unnecessary expensive projects. In some cases, ad hoc methods result in overly many iterations to be made and in some severe cases, total project breakdown. To be able to solve these problems a method has been developed to derive end-to-end non-functional constraints, such as timing requirements, to task-level constraints and to promote this information to the implementation phase of the project.**

For a tool, as the one described above, to be really useful it is important to be able to show that there is a potential cost reduction to be made. To be able to show that a certain implementation method costs less in work hours than to use an ad hoc implementation method, a model for implementation simulation with support for multiple implementation alternatives has been developed. The experiments show that using the budget based implementation method leads to a significant cost reduction in most cases, compared to the ad hoc method. As far as we know, no similar experiments has been done to compare implementation methods.

I. Introduction

Developing a computer system is a very complex task, typically there is a large set of constraints, interacting in ways hard to predict. This alone makes the process of computer system design very hard, even if only fully implemented software components were used. In the typical case, a few components may already be implemented, evaluated and documented (i.e. bought or reused), but most of the components will initially only exist as specifications.

For modern embedded real-time systems, complexity depends on functional requirements (e.g. the brake functionality of a car) as well as non functional requirements (e.g. the maximum allowed jitter between the different brake circuits) [1]. Since most commercially available system development tools only handle functional constraints and as the complexity of non functional constraints steadily increases, other types of development methods are needed. Related research shows that the lack of validation of non functional constraints will be very expensive in the long run [2]. To detect potentially volatile designs directly leads to lower cost than to detect them after implementation, since the decision can be made at the source of the problem – whether to abort the project or start over. Note that iterative development is probably unavoidable (i.e. all assumptions will not be correct from start), but design decisions leading to dead ends (or more trouble) should be avoided.

Besides the complexity issue, the design of software components and the dimensioning of the resources are normally done at a point in the project when resource limitations may be unknown and very little is implemented [3]. A method is thus needed to guide the designer, to keep within the specified non functional constraints, and validate the design, guaranteeing that there will be a way to implement and execute the system [4]. Since many parameters are unknown until implementation is completed, we would like to keep the design flexible for as long as possible to avoid dead ends in the development [5] [6] [7]. Such a method, the budget based method, has been presented in [8] and fits in the development chain as non-functional design validation (see Figure 1).

In this paper, we present a method that makes it possible to better compare different methods for guiding implementation in terms of implementation cost. We also present an extension to the basic method to treat multiple-implementation alternatives. This paper also presents the results of an experiment using the proposed extended method to compare our budget based implementation method to an *ad hoc* based method.

The traditional method in industry for handling, in particular, non-functional constraints in embedded system design has been *ad hoc*. Very little systematic or formal analysis of, for example, real-time properties (such as worst-case response time and the ability to

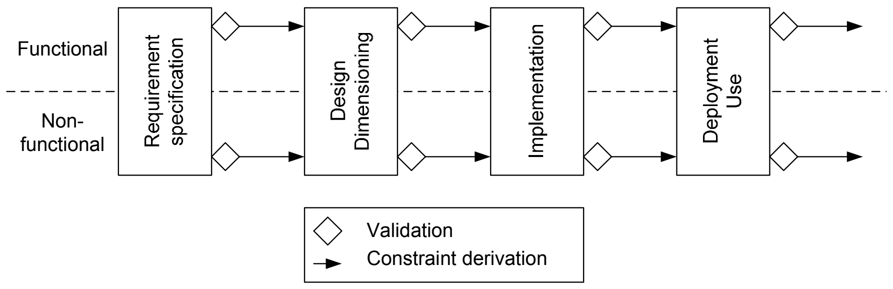


Fig. 1. Overview of development model. Our focus is on the non-functional design validation.

meet deadlines) has been supported by the commonly used development tool. Such properties have instead often been checked through more or less thorough testing after system implementation. In recent 10-15 years, however, some new analysis techniques have been proposed, which in a more systematic and formal fashion allow some properties to be predicted. Especially in the real-time case such analysis often relies on the assumption that some runtime constraints (RTCs), such as individual task deadlines, are enforced. A well-known example of a real-time analysis technique is Rate Monotonic Analysis (RMA), first described in [9]. In conjunction with this kind of analysis techniques, systematic methods to derive the required RTCs have also been proposed, see e.g. [10][11].

System co-design methods (see, e.g., [12] and [3]) form another related area. The main focus of these methods is to simultaneously generate a hardware architecture and the software which runs on it, where the architecture may consist of a mix of programmable CPUs, DSPs, and ASICs. These methods are based on quite detailed specifications, for example, a hardware descriptive language. Further, co-design methods could be seen as design space exploration, prototype creation, and implementation generation. In this context, our budget based method could be used in the early design stage to later reduce the design cost when using a co-design method.

In the area of schedulability analysis there has been a lot of research done and several methods and implementations for scheduling task sets on multiple processors have been proposed. However, the main difference between these scheduling methods and our work is that our proposed method does not require information about the actual outcome for the worst case execution time of the tasks, and that the proposed method is supposed to be used at the design stage rather than the implementation stage.

These methods of predicting whether a system will meet its constraints are helpful in the way that they avoid runtime problems that can result in very high costs. They may also point out where problems (bottlenecks) in the design or implementation of the system are to be found. However, in general, these methods typically depend on a full implementation or very detailed description of all tasks in a system, or at least knowledge of the worst-case execution time (WCET) for each of the tasks, which means that fundamental design problems can remain undetected until the implementation of all tasks is finished.

This paper focuses on the presentation of the basic method for budget generation (Section II), the basic method for evaluation of design methods by simulated implementation (Section III), a multiple-implementation alternatives extension to the basic method (Section III), and experiments using the proposed evaluation methods (Section IV). The generation of ITCs is limited to handle directed acyclic task graphs and homogeneous single bus architectures. The task graphs used for the experiments are a mix of well documented cases and automatically generated task graphs with up to 80 tasks.

II. The budget based method

In this section we present a method that, based on a functional system description with end-to-end constraints and a given platform, generate a set of task level implementation time constraints (ITCs). The functional system description is a directed acyclic task graph, where the designer has estimated the execution time for each task in the graph, in relative terms. Each task can have optional timing constraints (i.e., in the form of release time and/or deadline) and optional locality constraints limiting the possible processor allocations. For each precedence constraint of a task, an optional communication package of estimated relative length can be defined. The platform is modeled as a network with homogeneous processors connected to a single bus. The output from the method is a set of budgets where each budgets in the set consists of ITCs for each task and the associated communication tasks. It is assumed that the deadline for a task is shorter than the task's period.

Here follows an overview of the method illustrated in Fig. 2, further described below. Initial data formatting is done by the preprocessor that generates constrained task graphs, based on the input data (resource parameters, performance restrictions, and task set). The generation of schedules (scheduler in the figure) for the constrained task graph consists of two steps, ordering and allocation. The purpose of the ordering step is to generate execution orders of the

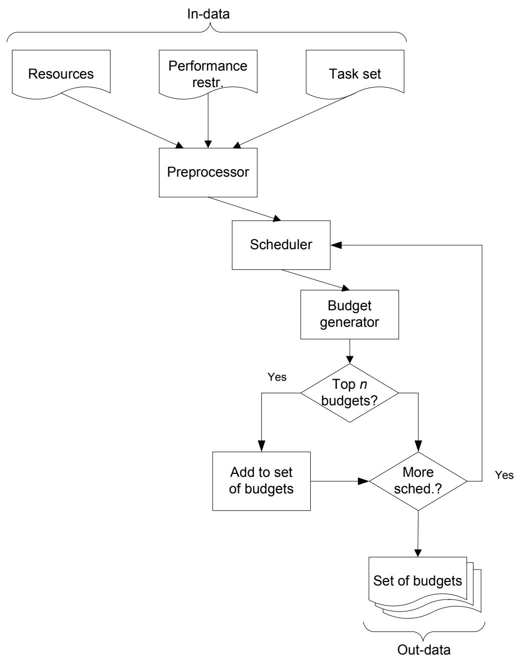


Fig. 2. Flowchart for the suggested implementation of the method. The schedules are turned into budgets and the n best are kept.

tasks according to the precedence constraints in the task graph. As part of the allocation, communication tasks are generated. The allocation of the tasks in the task graph is quite straightforward, generating allowed allocations according to the locality constraints. When allocation is decided, communication tasks can be generated based on the information about which tasks that have to communicate using the bus. Finally, ITCs (budgets) are generated based on the scheduled task graphs. The top n budgets with minimal tightnesses are saved in the set of budgets that is the out-data from the budget generation method.

To calculate the optimization metric, the budget tightness T, for the scheduled task graph, G (of tasks, V, and precedence constraints, E) the tightness for each path, $p \in P$, in the graph has to be found. As mentioned earlier, for each task there is an estimated execution time V_e. For (at least) the first and last task on the path, there is also a corresponding offset V_o and deadline V_d, respectively. The tightness for a path $p_T, p \in P$ is equal to the fraction between the work W, that is the sum of the task's estimates, and the path length L, that is the difference between the path's deadline and offset. The path's offset is equal to the first task's offset; while the path's deadline is equal to the last task's deadline. In summary we have the following definitions:

$$G = < V, E > \tag{1}$$

$$W = \sum_{v \in p} v_e \tag{2}$$

$$L = p_d - p_o \tag{3}$$

$$p_T = \frac{W}{L} \tag{4}$$

$$T = \max(p_T : p \in P) \tag{5}$$

When the tightest path has been found, the ITCs for the tasks on this path can be generated. The sum of the ITCs for the tasks on the tightest path shall fill out the whole length L of the path. This is done by, for each task on the path, dividing the estimated execution time, v_e, with the path tightness, p_T, giving us the allowed execution time (AET):

$$v_{AET} = \frac{v_e}{p_T} \tag{6}$$

The tasks on the tightest path may now be removed since they all have been given implicit offsets and deadlines (to be used as implementation budgets). One way to calculate the offset and the deadline explicitly for a task on the tightest path is to set the offset for a task to the previous task's deadline, and the deadline to its task's offset plus its AET:

$$v_{i_o} = v_{(i-1)_d} \tag{7}$$

$$v_{i_d} = v_{i_o} + v_{i_{AET}} \tag{8}$$

Since all tasks on the tightest path have explicit offsets and deadlines, it is possible to remove them from the task graph, replacing them with offsets and deadlines. This means that all parents of the task we are going to remove will get new deadlines equal to the removed task's offset. Similarly, all the children of the task we are going to remove will get new offsets equal to the removed task's deadline. When all tasks on the tightest path have been removed from the task graph, the second tightest path is calculated and the corresponding AETs generated. This procedure is then repeated until all tasks have been assigned AETs. Note that only the first tightest path's tightness is of interest when describing the tightness of the whole task graph budget. The consecutive paths' tightnesses are just calculated to be able to generate all tasks' AETs. Although the implementation space (success probability) is maximized in some sense, using the global tightness as optimization metric, there is no guarantee that the implementation will be possible. To avoid the need to start the whole budgeting process over, just because a certain budget did not hold, multiple promising budgets are generated and kept for later use. Extensions of the presented method and evaluation of the optimality are found in [8].

A. Example

To illustrate how budgets for a task graph are generated, we calculate the ITCs step by step for the task graph in Table III. In this example, tasks M_{n-0}, M_{n-2}, and M_{n-4} are allocated on processor 1, while tasks M_{n-1}, M_{n-3}, and M_{n-5} are allocated on processor 2. The execution order on each processor is (in this case) given explicitly by the precedence constraints in the task graph. However, since the two communicating tasks M_{n-0} and M_{n-3} are allocated on different processors we have to create a communication task

TABLE I

TIGHTEST PATH

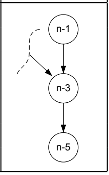
M_v	v_o	v_d	v_e	v$_{AET}$	Graph
n-1	10	52	40	42	
n-3	52	73	20	21	
n-5	73	100	25	27	

TABLE II

SECOND TIGHTEST PATH

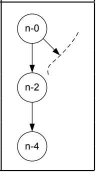
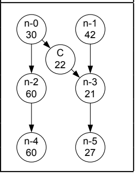
M_v	v_o	v_d	v_e	v$_{AET}$	Graph
n-0	0	30	25	30	
n-2	30	90	50	60	
n-4	90	150	50	60	

TABLE III

ALL TASKS BUDGETED

M_v	v_o	v_d	v_e	v$_{AET}$	Graph
n-0	0	30	25	30	 n-0 30 n-1 42 C 22 n-2 60 n-3 21 n-4 60 n-5 27
C	30	52	5	22	
n-1	10	52	40	42	
n-2	30	90	50	60	
n-3	52	74	20	21	
n-4	90	150	50	60	
n-5	74	100	25	27	

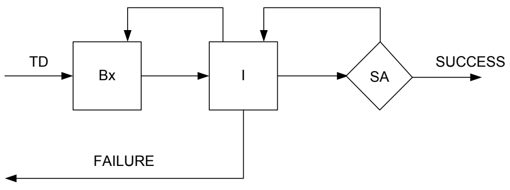


Fig. 3. The model for the implementation simulation.

$C_{n-0,n-3}$, allocated on the communication processor (the bus seen as a processor). Let us assume that, according to the specification, the new task $C_{n-0,n-3}$ has the estimated communication time 5. To calculate the tightness and assign AETs for all tasks we have to find the tightest path in the graph. There are three different paths in this example, $p_{n-0,n-4}$ with the total work of 125 and the length 150, $p_{n-0,n-5}$ with the total work of 75 (including the communication task $C_{n-0,n-3}$) and the length 100, and $p_{n-1,n-5}$ with the total work of 85 and the length 90. The tightness is calculated for each path (0.667, 0.75, and 0.944, respectively), which gives that $P_{n-1,n-5}$ is the tightest path with a tightness of 0.944.

Since the calculations show that the path $\{M_{n-1}, M_{n-3}, M_{n-5}\}$ is be the tightest it will also be the first to be assigned AETs (see Table I). The assigned tasks are then removed from the task graph and replaced with release times and deadlines where this applies. In this case it means that the communication task $C_{n-0,n-3}$ will get a deadline of $100 - 27 - 21 = 52$. The tightness calculations are now repeated for the remaining tasks in the graph. The path $\{M_{n-0}, M_{n-2}, M_{n-4}\}$ is now the tightest and therefore assigned AETs (see Table II). The tasks is then removed from the task graph and replaced with a release time for $C_{n-0,n-3}$ that will start at $0+30 = 30$. Now there is just one task left ($C_{n-0,n-3}$) that have a release time and a deadline, which makes the generation of it's AET trivial ($52 - 30 = 22$). All the tasks in the graph have now been assigned AETs (see Table III) and the tightness for this graph is equal to the tightness for the tightest path (0.944).

III. SIMULATED IMPLEMENTATION

We would like to show that implementing according to budgets is more efficient than to use an *ad hoc* method (i.e. try and fail). To be able to evaluate different implementation methods the implementation process has to be clearly defined as well as the cost for applying the process. In this paper it is assumed that the time for an engineer to implement a task is assumed to be the main part of the cost. Further it is assumed that the implementation process starts out with a set of tasks with unknown worst case execution time (WCET) and that each attempt to implement a task results in an actual WCET (AWCET) and a cost. Additional attempts to lower the WCET will result in additional cost as well.

Evaluation of an implementation method could be done in different ways, each with their drawbacks. For example, the design could be given to real programmers who try to implement the design. Another example would be to put the design in a simulator where the implementation is simulated. The gain with using real programmers is that it is the real implementation cost that is measured. The drawback with this method is that it is quite hard to repeat the experiment, which is needed if two or more methods are to be compared. The gain with using an implementation simulator is that the experiments can be repeated with the exact same result each time. The drawback is that we just get an approximation of the real implementation cost.

The proposed generic simulation model used for the implementation simulator (see Fig. 3) starts with information about the tasks from the design (TD). This information is used as input to the guideline generator (Bx) which in turn generates implementation-time guidelines. Bx can, for example, be represented by our budgeting method or an *ad hoc* method, as explained below. If no guidelines could be generated this will be considered as a failure of implementation of the pending design and re-design or re-specification may then be needed. Available budgets are forwarded to the (simulated) implementation (I) where WCETs for the tasks are calculated and cost is accumulated. Implementations are evaluated by a schedulability analysis (SA) tool to decide whether it is possible to create an executable schedule for the set of implemented

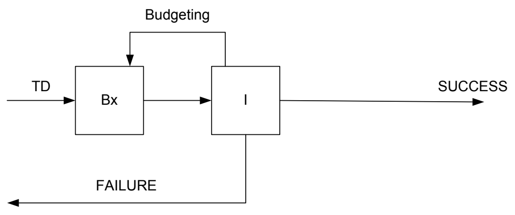


Fig. 4. The model for simulation of budget based implementation.

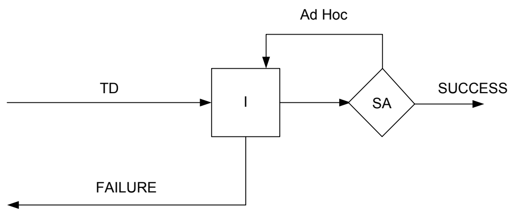


Fig. 5. The model for simulation of *ad hoc* implementation.

tasks. If this is the case the simulation will terminate positively (SUCCESS) signaling that there exists a schedulable implementation at a certain implementation cost. If a complete system implementation does not exist according to the schedulability analysis we have to reiterate, modifying the guideline generation parameters (e.g. updating estimates). If no changes (no new implementations) has been made since the last guideline generation, further guideline generation is pointless and the simulation will terminate negatively, signaling FAILURE.

The model for the budget based implementation simulation is shown in Figure 4. In the budget based implementation simulation the guideline generator (Bx) is represented by the budgeting algorithm, which generates the implementation guidelines. If implementation according to the guidelines are possible, this indicates SUCCESS since there exists at least one possible schedule. Since at least one schedule is know for the implementation there is no need for schedulability analysis. If implementation is not possible according to the guidelines, new guidelines are generated. If no guidelines could be generated fitting the already implemented tasks this indicates FAILURE.

The model for the *ad hoc* implementation simulation is shown in Figure 5. In this case there is no need for implementation guidelines, since all possible implementations are allowed. Each implementation generated in the implementation simulator (I) is tested in the schedulability analysis. If the implementation is schedulable SUCCESS is reported, otherwise implementation is repeated until no more implementations are possible. If there are no more implementations available the implementation results in FAILURE.

A. *Generation of cases*

To be able to simulate the implementation each task has been equipped with a list of implementation alternatives. Each implementation alternative in the list consists of a WCET for that implementation and an associated cost in work hours. The purpose with the implementation alternative is to model the fact that a given implementation will take a certain time in work hours to complete and that the more work hours put into the implementation, the lower the WCET will be.

Depending on how the implementation alternatives are used, different models for the cost could be thought of. In the simple case all implementation alternatives are independent and you are charged with the full cost for the implementation each time you decide to implement. An extension to this is the case where the implementation alternatives are dependent. This means that the cost for incremental implementations would be lower since code could be re-used.

To simulate the process of implementation using these implementation alternatives two things are needed, a method for deciding in what order the implementation alternatives should be implemented and a stopping criterion that indicates that no further implementation is needed (i.e. no more alternatives need to be implemented). This will be different for the single and multiple alternatives approach, described in detail below.

B. *Single alternative*

In the special case where only a single implementation alternative is available for each task it is very easy to simulate the implementation. For the *ad hoc* implementation the criteria for stopping simply is when all tasks have an implementation. Since all tasks should be implemented and there is only one alternative per task any implementation order could be used. For the budget based method it is a little more complicated. The order in which the tasks are implemented is based on the tightness of the budget for each task. The reason for this is that if the task with the tightest budget is not implemented within the budget this is an early indication of that the design and initial assumptions are bad. In the case with just a single implementation alternative this forces the generation of new budgets, trying to fit the under-budgeted task. Implementation is continued until all tasks have been implemented. If the under-budgeted task does not fit any of the new budgets the implementation process is aborted.

C. *Multiple alternatives*

For the case where multiple-implementation alternatives exist for each task the basic requirements are the same, an implementation order is needed as well as a stopping criteria. For the *ad hoc* strategy the initial implementation order is the same as used for the single implementation alternative, all tasks gets a single implementation. A simple *ad hoc* implementation strategy is assumed where the implementation order is in task order, one by one, if there are more implementation alternatives. The stopping criteria for the *ad hoc* strategy is when a solution has been found or when there are no more implementation alternatives to try out.

For the budget based strategy the implementation order is still the same as for the single implementation alternative case; in tightness order. The implementation is continued for the tightest task until the implementation fits within the budget. If an implementation fails to fit the budgets for the task this forces re-budgeting. Note that while the *ad hoc* strategy works horizontal, implementing one implementation alternative for each task, the budget based strategy works vertically, implementing all the implementation alternatives needed for a task before the next task is implemented. The stopping criteria for the budget based strategy is when all tasks are implemented in such a way that all implementations fit within their given budget or that no further implementation is possible. No further implementation is possible if all implementation alternatives for a task have been implemented and the re-budgeting does not find any new budgets that will fit the task's implementation.

IV. Analysis

The task sets used for the implementation simulation are described in detail in [13]. It is a mix of hand made cases and automatically generated cases. The sizes of the cases range from a couple of nodes to the size of hundreds of nodes. The interconnects of precedence constraints are sparse in most cases.

It is assumed that the implementation of a single task starts with a coarse proposal that is refined during the implementation into the final implemented component. To model this in simulation, it is assumed that all tasks have an estimated WCET for the component to be implemented (EWCET) and an AWCET. The AWCET is the WCET for a specific implementation of the component and is unknown until implementation has been completed. During implementation, the cost in work hours is accumulated to give the total number of work hours needed to complete the project.

The proposed implementation simulation method allows several implementation alternatives for a task, with different AWCET and cost, making incremental implementation possible. There are many ways to generate a set of implementation alternatives, and in this paper the solution simply is to generate several alternatives for each task and then sort them in AWCET order, according to the method below.

For each task, a random AWCET is generated from a rectangular distribution centered at the EWCET, where the width of this window is proportional to the certainty of the estimate. To avoid trivial implementations, a constant can be added to the EWCET to shift the distribution, resulting in higher AWCETs and therefore generating implementation simulations of increased difficulty (i.e. introducing an offset for optimistic estimates). The AWCETs in the experiments are generated from a distribution that is 1.5 times the estimate in width, and with an offset that sets the lowest AWCET to 5/8 of the estimate. For example, for a task with an estimate of 100, the offset $AWCET_o$ is 62.5 and the width of the distribution is 150, meaning that the AWCET will be between 62.5 and 212.5 (i.e. $AWCET_o$ and $AWCET_o + 1.5e$, respectively).

The cost for a task's implementation is considered as a function inverse proportional to the AWCET, where a lower AWCET leads to higher cost. This is implemented in the experiments as the estimate, e, divided by the AWCET minus the offset $AWCET_o$, plus a constant, c (to avoid division by zero):

$$cost = \frac{e}{AWCET - AWCET_o + c} \qquad (9)$$

This is based on the assumption that the cost for implementing a certain task will be higher the closer to the $AWCET$ limit ($AWCET_o$) the implementation gets. The described cost function is used for the single implementation alternative evaluation as well as for the multiple alternatives evaluation. When having multiple-implementation alternatives several cost-AWCET pairs will be generated and then ordered according to descending AWCET.

A. Discussion

In the simple case with a single implementation alternative per task the budget based implementation method and the *ad hoc* implementation method will generate the same cost in the case of successful implementation. This is because there is only a single implementation alternative and all tasks have to be implemented to generate a successful implementation. In the cases of implementation failure, the budgeting method will identify this at a lower or equal cost in work hours compared to the *ad hoc* method. Figure 6 shows a plot of the cost for implementing using the budget based method divided by the cost for using the *ad hoc* method. To make the plot clearer, all successful implementations has been filtered out, since they result in equal cost. Note that if a solution is possible to find using one of the methods, it will be found using the other method too.

With several implementation alternatives per task the *ad hoc* method might theoretically, in some cases, result in a cost lower than the cost for using the budget based method. However, still the budget based method don't have to implement all tasks before analyzing the result. Further, the budget based method has the possibility to detect problems that cannot be solved by further implementation, resulting in a lower cost. In cases where no schedulable implementation exists we know that the budget based method will not lead to a higher cost than the *ad hoc* method. This since the *ad hoc* based method has to implement all possible alternatives before it is allowed to quit. The relative cost for using the budget based method turned out to be as low as 20% of the cost for using the *ad hoc* method (see Fig. 7). Figure 7 shows a plot of the cost for implementing using the budget based method divided by the cost for using the *ad hoc* method. Note

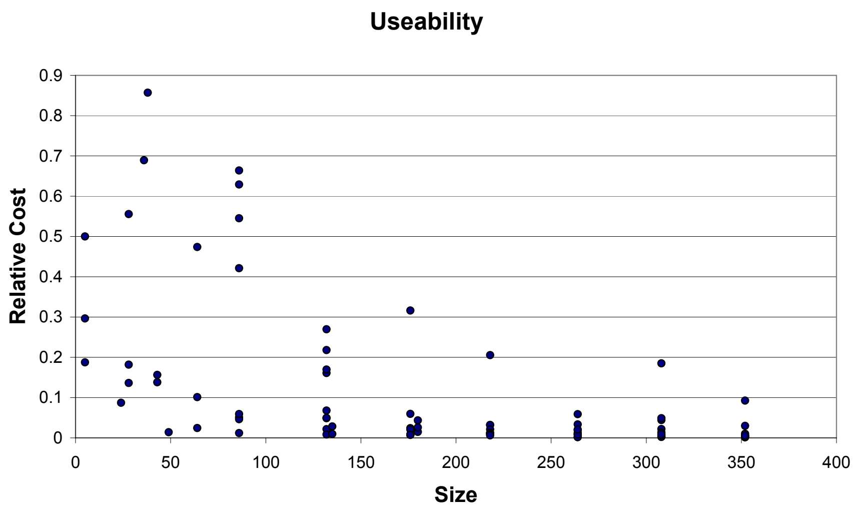


Fig. 6. In the case with a single implementation alternative for each task the relative cost drops with the task graph size in favor of the budget based method.

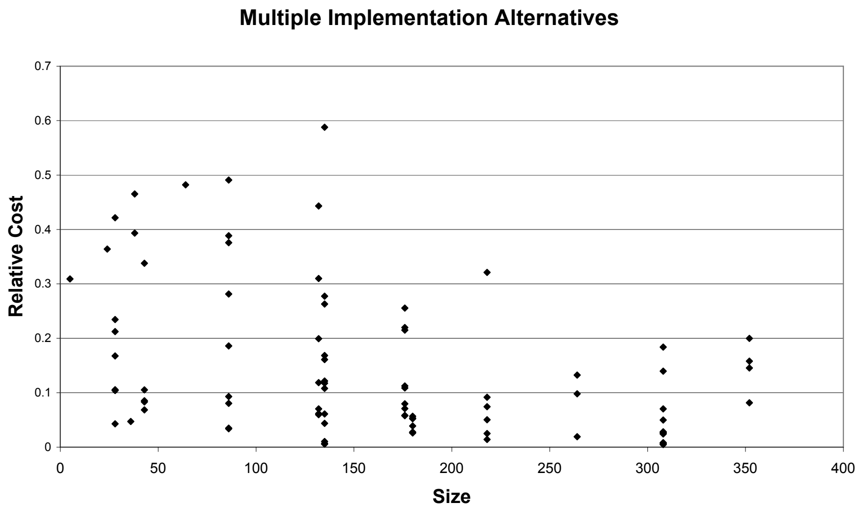


Fig. 7. In the case with multiple implementation alternatives for each task the relative cost drops with the task graph size in favor of the budget based method.

that in a few cases the *ad hoc* method outperforms the budget based method (these values has been filtered out for clarity). As stated earlier this is fully possible, but luckily not that common. A more detailed investigation of these cases showed that this may be the result for cases where a schedulable implementation is close to the highest cost. This could be seen as if the budgeting method do not think there is a solution to the problem and stops early, while the stubborn *ad hoc* method keeps on and find the solution. It could be argued that designs leading to these kind of, close to failure, implementations should be avoided in favor of redesign.

These results clearly point out that the proposed single alternative budget-based method leads to lower design cost when it is needed most – when the implementation is leading to a dead end. For the multiple-implementation alternative simulation, it has been shown that the budget based method leads to a lower cost when there exist no schedulable implementations. Also, it leads to a lower cost in most cases where there exists a schedulable implementation.

The proposed method can be extended further, allowing better modeling of implementation cost and supporting incremental implementations. It can also be interesting to evaluate other popular implementation methods for comparisons.

It may be argued that the *ad hoc* implementation method is simplistic but on the other hand this sets a base on which one could build other, more sophisticated, implementation methods. A natural step would probably be to allow the *ad hoc* method to use information that could be extracted from the schedulability analysis. Examples of such information are laxity and bottlenecks.

V. Conclusions

In this paper a basic method for implementation budget generation has been presented. Further, a method, for evaluation of the described budget generation method, using simulated implementation has been presented.

The experiments has shown, using simulated implementation, that the proposed design/ implementation method using budgets, is more efficient in most cases (in terms of work hours) than the pure *ad hoc* method. In other words, the proposed method using budgets accelerates the development of a system to the point where you know if it will be possible to implement it using the chosen design or not. The proposed implementation simulator also makes it possible to evaluate and compare different design/ implementation methods, in terms of cost reduction.

References

[1] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, and N. Bånkestad, "Experiences from introducing state-of-the-art real-time techniques in the automotive industry," in *Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*. IEEE, Apr. 2001, pp. 111–118.

[2] D. C. Gause and G. M. Weinberg, *Exploring requirements: quality before design*. Dorset House, 1989, ch. 2, pp. 17–18.

[3] J. Plantin and E. Stoy, "Aspects of system-level design," in *Proc. 7th International Workshop on Hardware/Software Codesign*. ACM Press, 1999, pp. 209–210.

[4] A. Dasdan and R. Gupta, "Timing issues in system-level design," in *Proc. IEEE Computer Society Workshop on VLSI*, Apr. 1998, pp. 124–129. [Online]. Available: citeseer.nj.nec.com/article/dasdan98timing.html

[5] G. Le Lann, "A methodology for designing and dimensioning critical complex computing systems," in *Proc. IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, Mar. 1996, pp. 332–339.

[6] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *Software Engineering*, vol. 21, no. 7, pp. 579–592, 1995. [Online]. Available: citeseer.nj.nec.com/gerber95guaranteeing.html

[7] D. Sciuto, F. Salice, L. Pomante, and W. Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *Tenth International Workshop on Hardware/Software Codesign*. ACM SIGDA, May 2002, pp. 55–60.

[8] M. Wecksten, J. Vasell, and M. Jonsson, "Towards a tool for derivation of implementation constraints," in *Proc. International Conference on Engineering of Complex Computer Systems (ICECCS'2004)*. IEEE, Apr. 2004, pp. 119–127.

[9] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.

[10] M. D. Natale and J. A. Stankovic, "Dynamic end-to-end guarantees in distributed real time systems," in *Proc. IEEE Real-Time Systems Symposium*, 1994, pp. 216–227.

[11] M. Saksena and S. Hong, "Resource conscious design of distributed real-time systems – an end-to-end approach," in *Proc. IEEE International Conferece on Engineering of Complex Computer Systems*, Oct. 1996, pp. 306–313.

[12] G. de Jong, "A UML-based design methodology for real-time and embedded systems," in *Proc. Design, Automation and Test in Europe*, 2002, pp. 776–779.

[13] M. Wecksten, *Resource Budgeting as a Tool for Reduced Development Cost for Embedded Real-time Computer Systems*. Chalmers, Apr. 2004, 33L.