Technische Universiteit
Eindhoven
University of Technology

Pareto analysis with uncertainty

Martijn Hendriks, Marc Geilen and Twan Basten

ES Reports
ISSN 1574-9517

ESR-2011-01
3 August 2011

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems

This report is an extended version of the following paper:

M. Hendriks, M. Geilen, T. Basten. *Pareto Analysis with Uncertainty*. In *9th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, EUC 2011*, Proceedings. Melbourne, Australia, October 24-26, 2011. IEEE CS Press, 2011.

The report includes the proofs omitted in the EUC paper.

http://www.es.ele.tue.nl/esreports
esreports@es.ele.tue.nl

Eindhoven University of Technology
Department of Electrical Engineering
Electronic Systems
PO Box 513
NL-5600 MB Eindhoven
The Netherlands

Pareto Analysis with Uncertainty

Martijn Hendriks
Radboud University Nijmegen
Email: M.Hendriks@cs.ru.nl

Marc Geilen
Eindhoven University of Technology
Email: M.C.W.Geilen@tue.nl

Twan Basten
Eindhoven University of Technology
and Embedded Systems Institute
Email: A.A.Basten@tue.nl

***Abstract*—Pareto analysis is a broadly applicable method to model and analyze tradeoffs in multi-objective optimization problems. The set of Pareto optimal solutions is guaranteed to contain the best solution for any arbitrary cost function or selection procedure. This work introduces a method to explicitly take uncertainty into account during Pareto analysis. A solution is not modeled by a single point in the solution space, but rather by a set of such points. This is useful in settings with much uncertainty, such as during model-based design space exploration for embedded systems. A bounding-box abstraction is introduced as a finite representation of Pareto optimal solutions under uncertainty. It is shown that the set of Pareto optimal solutions in the proposed approach still captures exactly the potentially best solutions for any cost function *as well as* any way of reducing the amount of uncertainty. During model-based design space exploration, for instance, design and implementation choices that are made during the development process reduce the amount of uncertainty. Steps in such a refinement trajectory can render previously Pareto optimal solutions suboptimal. The presented results provide a way to ensure that early selections in the refinement process remain valid.**

***Keywords*-Pareto analysis, uncertainty, model-based design space exploration, refinement.**

I. Introduction

Pareto analysis is a well-known concept to model and analyze tradeoffs in multi-objective optimisation problems [1]. It originates from the economist Vilfredo Pareto who introduced a notion of optimality that states that a solution is optimal if it is not possible to find another solution which improves some of the objectives without worsening any of the others [2]. The Pareto optimal solutions constitute the Pareto frontier of the problem space and they accurately capture any tradeoff among the objectives. To ultimately select a solution, a cost function, such as a weighted sum of the various objective values, may be used to assign a cost to every solution. It is guaranteed that the cheapest solution for an arbitrary monotone cost function is always found on the Pareto frontier.

In much of the existing work on Pareto analysis, the solutions are represented by points in the solution space. Sometimes, however, *uncertainty* is present in the values of solutions. The work in the present paper investigates this situation and models solutions as arbitrary sets of points in the solution space. This naturally applies to model-based design space exploration for embedded systems. Early in the design phase the models will contain a lot of uncertainty because of, e.g., unresolved design choices and missing information. Objectives such as system latency and energy consumption have therefore not single values but rather can take their value from a range or set of values. The main question investigated in this work is under which conditions decisions that are based on information with uncertainty remain valid when (some of) the uncertainty is resolved.

Contribution. The main contribution of this paper is a general method to take uncertainty into account during Pareto analysis which fits in the framework of [1]. Furthermore, it is shown how this method naturally fits the area of model-based design space exploration of embedded systems. The development process – or refinement process – reduces the amount of uncertainty in the models. The proposed method provides a way to ensure that early choices in the refinement process remain valid. Furthermore, it is shown by example that Pareto analysis with uncertainty provides a way to trade design space size against model size. This can be very effective when the underlying analysis tools do not suffer (much) from the additional state space load. This work connects Pareto analysis, abstraction/refinement theory and model-based design-space exploration.

Related Work. There is a plethora of work on optimization under uncertainty, see e.g., [3] and [4]. Pareto analysis and uncertainty is explored in [5], [6] and [7] which all consider intervals on objectives for Pareto dominance. In [5] and [6] probability distributions are used to distinguish designs which have overlapping valuations. That approach is different from the approach of the present paper which is centered around *non-deterministic* domination between sets. Probably most closely related is [7] which uses a dominance relation on intervals that is almost equivalent to our definition. Our work in addition gives a justification for such a definition and investigates the impact of the reduction of uncertainty. Traditional refinement theories ([8], [9], [10], [11]) focus on behavioural refinement relations whereas our abstraction/refinement approach focuses on performance.

Outline. In Sec. II it is argued that uncertainty is needed to build faithful models. Sec. III recalls Pareto analysis theory and introduces a small example of a typical DSE problem which serves as a running example throughout the paper. Sec. IV formally defines the model-based design space exploration problem. Sec. V proposes a method to fit uncertainty in the framework of [1]. It shows a natural way of modeling uncertainty – by a set of possible values (for a property such as system latency) instead of by a single value. Such a set can be

This work was carried out as part of the Octopus project with Océ Technologies B.V. under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Bsik program.

accurately represented by a finite representation. Furthermore, it is shown that the resulting Pareto frontier is both sufficient and necessary when the amount of uncertainty is reduced. Sec. VI describes an industrial case study from the digital copying domain. It demonstrates how the framework presented in the present paper can be used. Finally, conclusions are presented in Sec. VII.

II. Uncertainty in Model-Based Design Space Exploration

One of the most important properties of model-based design space exploration is that exploration of the design space before the realization of a system is based on formal models rather than on ad-hoc calculations or physical prototypes. This should be more precise when compared to ad-hoc calculations and faster and cheaper when compared to the usage of physical prototypes. The models that are used are thus always models of non-existing systems. A clear prerequisite is that the models are *faithful*. This means that any implementation of a model should preserve the properties that have been discovered by analysis of the model. Unfortunately, this is problematic as even the faithful modeling of existing systems is far from trivial. In [12], for instance, it is observed for verification that "model hacking precedes model checking". The authors mean by this that the first few attempts to verify a formally modeled system often are only used to get the model right. How can a non-existing system be modeled faithfully if modeling an existing system already is very difficult? How can model-based design space exploration be applied if the models cannot really be trusted? The approach taken in this paper is to explicitly take uncertainty into account during both the modeling and during the interpretation of the analysis results. There are various ways to achieve this:

- *Non-deterministic behavior in the model.* Under-specification of system behaviour can be used to (i) keep various implementation routes open (e.g., priorities of tasks are left unspecified), (ii) to model partial knowledge (a task takes at least 10 ms and at most 20 ms), or (iii) to abstract from interaction (e.g., harddisk speed between 50 and 150 MB/s depending on the load). Furthermore, there can be inherent non-determinism in the system or environment (such as jitter in the arrival times of a task).
- *Probabilistic behavior in the model.* System behavior (execution time of a task, the system workload, etc.) may have a known probability distribution associated with it. Note that non-determinism can be used for this, but then some useful information may be lost.
- *Post processing of analysis results.* Analysis results may not be faithful because of (i) incomplete analysis and (ii) errors in the model. The state space explosion often prevents exhaustive analysis of formal models. This is very visible in design space exploration where in general a large number of models is analyzed. Even an analysis time of a minute per model is too much if there is a large number of models. As a result, only approximations of the bounds of system properties may be known. Furthermore, invalid abstractions such as faulty timing or ignoring memory fragmentation can have a significant effect on system properties. The analysis results can be adjusted afterwards in a post-processing step to counter these effects.

It often is tempting to reduce the uncertainty in high-level models to make them more amenable for analysis tools (simulation, model checking, etc.). The next example shows that this can have a significant effect on the analysis result and thus on the faithfulness of the model. Consider a system with 5 tasks, their precedences and their execution times (between parenthesis): $D1(9) \rightarrow A(1) \rightarrow B(1000)$ and $D2(10) \rightarrow C(1000)$. Tasks A and C share the same non-preemptive resource. Assume that the supervisory control system starts tasks as soon as they are enabled. If the execution times of $D1$ and $D2$ are indeed exactly 9 and 10 time units respectively then there is only a single execution: task $D1$ and $D2$ run in parallel, then A runs, and finally B and C run in parallel. The system latency thus equals 1010 time units. Suppose, however, that the execution time of $D2$ is not exactly 10 time units but it might sometimes be a bit faster: potentially completing in 9 time units. The following schedule then also becomes possible: $D1$ and $D2$ run in parallel, $D2$ finishes first, then C starts which delays A and B. The latency of this behavior equals 2010 – almost twice as long – because task $D2$ is a bit faster than expected.

The example demonstrates that there may be a thin line between a faithful and an unreliable model. A similar effect of small disturbances with significant consequences in the results is described in [13]. This work analyzes the effect of clock drifts on the upper bound of message latency. It is the case that a relatively small drift in clocks of components has a large impact on the message latency. This may be due to the same kind of "inversion" effect that occurs in the example above.

In model-based design space exploration, decisions are made based on system properties derived from high-level models of non-existent systems. This section argued that it is unavoidable that the models and therefore the analysis results contain uncertainty in order to be faithful. The present paper investigates the impact of this uncertainty on the high-level decisions that are made during model-based design space exploration.

III. Pareto Analysis Preliminaries

Pareto analysis is a broadly applicable method to model and analyze tradeoffs in multi-objective optimization problems. This section recalls some definitions and results from [1]. First, however, a small design-space exploration problem is introduced that serves as a running example.

Fig. 1 shows a small synthetic system which poses a typical design space exploration problem. The system consists of an FPGA with 3 logic blocks: two functional blocks and 1 memory block. The application that runs on this FPGA consists of 10 objects that need to be processed by the four functional steps A through D in order. The workload per step per object is given in the step, e.g., A has a workload of 2. Each

object also needs 10 units of memory during the processing. Note that the mapping of the functional steps to either F1 or F2 is not yet determined.

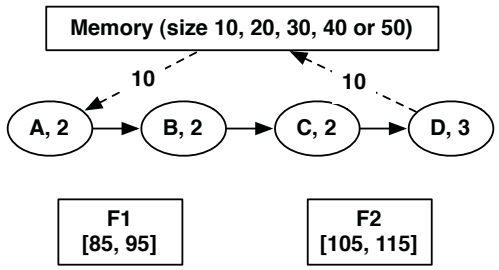

Fig. 1. A small design-space exploration example. There are 16 ways to map the functional steps to the two functional blocks and there are 24 totally ordered priority assignments of the tasks. Combined with a memory with a size in $\{10, 20, 30, 40, 50\}$ this gives $16 \times 24 \times 5 = 1,920$ design alternatives.

The time that each processing step takes is the product of the load of the step and the unit processing time of the block to which it is mapped. For instance, if step A is mapped to F1 then it takes between 2×85 and 2×95 time units. The uncertainty in the execution times of the functional blocks reflects uncertainty about the ultimate implementation of the functionality. Taking this into account improves the faithfulness of the model. The two system properties of interest are system latency and memory size. There are three unresolved design choices: (i) how much memory does the system need, (ii) how should the steps be mapped, and (iii) what should the task priorities be?

A *quantity* is a set Q with a partial order $\preceq_Q$. The memory size and the system latency of the running example can both be modeled as a natural number and can be compared with the natural ordering which makes them quantities. A *configuration space* is the Cartesian product $Q_1 \times Q_2 \times \cdots \times Q_n$ of a finite number of quantities, and a *configuration* is an element of a configuration space. The space $\mathbb{N} \times \mathbb{N}$ which contains tuples of memory and latency values is the configuration space of the running example. A concrete configuration is, for instance, $(30, 2098)$: a memory size of 30 and a latency of 2098. If $c_1 = (c_1^1, \ldots, c_1^n)$ and $c_2 = (c_2^1, \ldots, c_2^n)$ are two configurations from the same configuration space S then c_1 *dominates* c_2, denoted by $c_1 \preceq_S c_2$ if and only if $c_1^i \preceq_{Q_i} c_2^i$ for all dimensions i. The irreflexive variant – *strict dominance* – is denoted by $\prec_S$. The notation $c_1[k]$ is used to denote c_1^k. If S is clear from the context then it is omitted from the $\preceq_S$ notation. Consider two concrete configurations from the running example: $a = (20, 5068)$ and $b = (30, 7000)$. Then $a \preceq b$ because $20 \leq 30$ and $5068 \leq 7000$.

A set C of configurations is said to be *Pareto minimal* if and only if for every $c_1, c_2 \in C$ holds that $c_1 \not\prec c_2$. In general, configurations that are not strictly dominated by other configurations are called *Pareto points*, and are *Pareto optimal*. A configuration set C_1 dominates a configuration set C_2 from the same space, denoted by $C_1 \preceq C_2$ if and only if for every $c_2 \in C_2$ there is some $c_1 \in C_1$ such that $c_1 \preceq c_2$. Two sets of configurations C_1, C_2 from the same space are *equivalent*, denoted by $C_1 \equiv C_2$ if and only if $C_1 \preceq C_2$ and $C_2 \preceq C_1$. Every finite set C of configurations has a unique Pareto minimal subset D such that $D \equiv C$. This is the *Pareto frontier* of the set of configurations, and consists of all Pareto points of C.

Let $(Q, \preceq_Q)$ be a poset. A *cost function* on a configuration space S is a function $f : S \to Q$ that is montone: for all $c_1, c_2 \in S$ it holds that if $c_1 \preceq c_2$, then $f(c_1) \preceq_Q f(c_2)$. Let C be a set of configurations. A configuration $c \in C$ is *cost optimal for* C if $f(c)$ is minimal w.r.t. $\preceq_Q$ in the set $\{f(c) \mid c \in C\}$. The function that projects configurations of the running example to the system latency dimension is a cost function (one that favours system latency over memory size). An important result is that the Pareto frontier is necessary and sufficient to represent all possible tradeoffs w.r.t. cost functions.

The running example immediately shows that point values in the $\mathbb{N} \times \mathbb{N}$ configuration space may not be suitable to model the problem because the system has more than a single value for its latency due to the uncertainty w.r.t. task execution times. Observant readers may have noticed that [1] also allows a partially ordered domain of *intervals* to model uncertainty. However, instead of going directly to such a partially ordered set of intervals, the present paper takes a more structured approach by modeling uncertainty as arbitrary sets of configurations and by showing that such arbitrary sets have a finite interval representation which fits in the framework of [1].

IV. Model-Based Design Space Exploration

Universes $\mathbf{M}$ of models and $\mathbf{P}$ of parameters are assumed, and $\mathbf{Q}$ denotes the set of all quantities. Each parameter can take values from some quantity: The $qnt : \mathbf{P} \to \mathbf{Q}$ function gives the quantity of a parameter. Let $P = \{p_1, \ldots, p_n\} \subseteq \mathbf{P}$ be a set of parameters. A function $v : P \to qnt(p_1) \cup \cdots \cup qnt(p_n)$ with $v(p_i) \in qnt(p_i)$ is called a *parameter valuation*. The set of all parameter valuations of P is denoted by $val(P)$. A model $M \in \mathbf{M}$ captures the behavior of a system and is parameterized by a set of parameters, denoted by $param(M)$.

Definition 1 (Design Space): The design space of a model M is the set of design alternatives $\{(M, v) \mid v \in val(param(M))\}$ and is denoted by $dspace(M)$.

The DSE problem of the running example can be described by a model with parameters that specify the memory size, the mapping and the task priorities. The task priorities are left unspecified in the model. This is a technique to trade design space size for model size. The design space is a factor 24 smaller, but the state space of single design alternatives might be larger as additional non-determinism due to unspecified priorities is present. The model in this case has 5 parameters: $\{mem\_size, mapA, mapB, mapC, mapD\}$. The quantities are the following: $qnt(mem\_size) = \{10, 20, 30, 40, 50\}$, and

qnt(*mapA*) = *qnt*(*mapB*) = *qnt*(*mapC*) = *qnt*(*mapD*) = $\{F1, F2\}$. This gives $5 \times 2^4 = 80$ design alternatives.

The size of the design space of a model clearly is exponential in the number of parameters. A *dynamic quantity* for M is a quantity Q whose value is obtained by evaluation of system behavior. Typical examples include latency and energy consumption. The evaluation usually consists of model analysis by e.g., a simulator or analytic performance model.

Configurations are formed by a combination of *relevant* parameter valuations and values for dynamic quantities. A parameter is relevant if its value could potentially directly distinguish quality of design alternatives. For instance, the *mem_size* parameter of the running example is relevant because a system with less memory is cheaper. Furthermore, the running example has a single dynamic quantity: the system latency.

Definition 2 (Configuration Space): Let M be a model, let $\{p_1, \ldots, p_m\} \subseteq$ *param*(M) be the set of relevant parameters and let $Q_1, \ldots, Q_n$ be a set of dynamic quantities for M. The configuration space of M, denoted by *cspace*(M), is the Cartesian product *qnt*$(p_1) \times \ldots \times$ *qnt*$(p_m) \times Q_1 \times \ldots \times Q_n$.

The running example thus has a 5-dimensional design space with 80 design alternatives and a 2-dimensional configuration space which shows the latency-memory tradeoff.

In most existing work on Pareto analysis a design alternative is mapped to a single configuration, which allows, for instance, the application of the framework of [1] (Sec. III). In order to be able to express uncertainty, however, a design alternative is mapped to an arbitrary, non-empty, set of configurations in the present paper. Such a set can express lower and upper bounds on properties such as latency and energy consumption, and is also general enough to express correlations between such dynamic properties within a single design alternative.

$$\textit{config} : \textit{dspace}(M) \rightarrow 2^{\textit{cspace}(M)} \setminus \emptyset \tag{1}$$

A refinement is a relation on the set of models that relates compatible and faithful models. Two models are compatible if they have the same configuration space, and if the parameters of the refined model are a subset of the parameters of the refinement. A refinement is faithful if the behavior of the refinement is contained in the behavior of the refined model [8], [9], [10], [11], in our case, w.r.t. the configuration space.

Definition 3 (Refinement): Let M and M' be models. It is said that M' refines M, denoted by $M' \sqsubseteq M$, if and only if :

- (compatibility) *param*$(M) \subseteq$ *param*(M') and *cspace*$(M) =$ *cspace*(M')
- (faithfulness) *config*$((M', v')) \subseteq$ *config*$((M, v))$ for all $v \in$ *val*(*param*(M)) and $v' \in$ *val*(*param*(M')) such that $v'(p) = v(p)$ for all $p \in$ *param*(M).

The task priorities have been left unspecified in the current model of the running example as a way to treat design space size against model size. A model which explicitly adds task priorities with a new set of parameters refines the current model. Faitfulness follows from the fact that such a model exhibits a subset of the behavior of the current model.

Refinement is defined on the level of models rather than on the level of design alternatives. Design alternatives of related models with the same high-level parameter valuation are, however, connected by the faithfulness requirement. Let $M' \sqsubseteq M$ and let $v \in$ *val*(*param*(M)), and let $v' \in$ *val*(*param*(M')) such that $v'(p) = v(p)$ for all $p \in$ *param*(M). It is said that the design alternative (M', v') *refines* the design alternative (M, v), denoted by $(M', v') \sqsubseteq (M, v)$. Thus, if A' refines A then these design alternatives have the same high-level parameter valuation by definition.

Consider the running example and let $A = (M,$ *mem_size* $= 10,$ *mapA* $=$ *F1*, *mapB* $=$ *F2*, *mapC* $=$ *F1*, *mapD* $=$ *F2*$)$. Let M' be a refinement of M that adds priorities: a number from 1 to 24 that specifies a total order between the four tasks. Let $A' = (M',$ *mem_size* $= 10,$ *mapA* $=$ *F1*, *mapB* $=$ *F2*, *mapC* $=$ *F1*, *mapD* $=$ *F2*, *prio* $= 2)$. Then $A' \sqsubseteq A$ because A and A' have the same valuation for the high level parameters.

The design space exploration problem is to find a minimal set of *optimal* design alternatives. There are two complicating factors which prevent direct application of the definitions summarized in Sec. III:

- Design alternatives are mapped to sets of configurations in order to be able to express uncertainty and the correlations between quantities within a single design alternative.
- All possible refinements of a design alternative must be taken into account, i.e., decisions on the high level must remain valid.

V. Pareto Analysis and Uncertainty

A. Non-deterministic configurations

A natural way to express uncertainty w.r.t. the quantity values of a certain design alternative is to model this uncertainty by a set of configurations instead of by a single configuration, as proposed by Eq. 1.

Definition 4 (Non-deterministic configuration): Let S be a configuration space. A non-deterministic configuration is a non-empty subset of S.

To establish dominance at a higher level model requires that it is always preserved by refinement. Refinement – or reduction of uncertainty – corresponds to the act of reducing the set of configurations, as proposed by Def. 3. This leads to the following definition of dominance.

Definition 5 (Non-deterministic dominance): Let C and C' be two non-deterministic configurations on the same configuration space. Then C dominates C', denoted by $C \preceq^* C'$, if and only if for all $c \in C$ and $c' \in C'$ holds that $c \preceq c'$.

Non-deterministic domination can be lifted to the design space of a model as follows. Let M be a model and let $A_1, A_2 \in$ *dspace*(M). Then $A_1 \preceq^* A_2$ if and only if $A_1 = A_2$ or *config*$(A_1) \preceq^*$ *config*(A_2).

Proposition 1: $\preceq^*$ is a pre-order on the design space[1].

The relation is not a partial order because two different design alternatives can be mapped to the same singleton configuration: this invalidates anti-symmetry. Two design alternatives A_1 and A_2 are *equivalent* if and only if $A_1 \preceq^* A_2$ and $A_2 \preceq^* A_1$, and is denoted by $A_1 \equiv A_2$. This is an equivalence relation, and the equivalence class of a design alternative A is denoted by $\overline{A}$. The *quotient* of a design space of a model M is defined as follows:

$$\mathit{qspace}(M) = \{\overline{A} \mid A \in \mathit{dspace}(M)\}$$

Proposition 2: If $A_1 \equiv A_2$, then $\mathit{config}(A_1) = \mathit{config}(A_2) = \{c\}$ for some configuration c.

Equivalent design alternatives thus have the same singleton configuration and hence there is no reason to prefer one over the other. The pre-order induces a partial order on the quotient design space. This enables the re-use of the definitions introduced in [1]: $\mathit{qspace}(M)$ together with the induced partial order on it is a quantity. Note, however, that the multi-dimensional configuration space of Def. 2 is collapsed into a single partially ordered quantity. This disables much of the possibilities of the framework. The next subsection presents two finite representations of which one addresses this issue.

B. Finite representations of non-deterministic configurations

The following definition represents an arbitrary non-deterministic configuration by two configurations which describe its *bounding box*. To this end, from now on, we assume that all quantities are complete lattices, so that infimum and supremum are guaranteed to exist.

Definition 6 (Bounding Box Representation): Let S be a configuration space, and let $C \subseteq S$ be a non-deterministic configuration. Then $\mathit{inf}(C)$ is the configuration $(v_1, \ldots, v_n)$ where $v_i = \mathit{inf}(\{c[i] \mid c \in C\})$ and $\mathit{sup}(C)$ is the configuration $(v_1, \ldots, v_n)$ where $v_i = \mathit{sup}(\{c[i] \mid c \in C\})$. The set $\{\mathit{inf}(C), \mathit{sup}(C)\}$ is the bounding box of C, and is denoted by $\mathit{BB}(C)$.

Note that the infimum and supremum points of a non-deterministic configuration do not need to be elements themselves of that non-deterministic configuration. Fig. 2 shows a non-deterministic configuration and its bounding box representation.

The finite bounding box representation, which itself is a non-deterministic configuration, is an exact abstraction w.r.t. the non-deterministic dominance relation. Note that $\mathit{inf}(C) \preceq \mathit{sup}(C)$. The following theorem show that correlations between quantity values in a non-deterministic configuration are not important for Pareto analysis and we are allowed to combine the individual extreme values per quantity to obtain the bounding box.

[1]Proofs are published in the appendix.

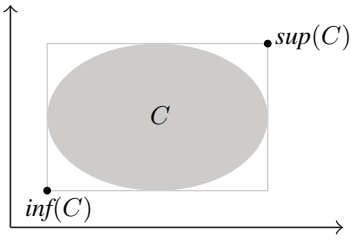

Fig. 2. A non-deterministic configuration C in a two-dimensional configuration space and its bounding box.

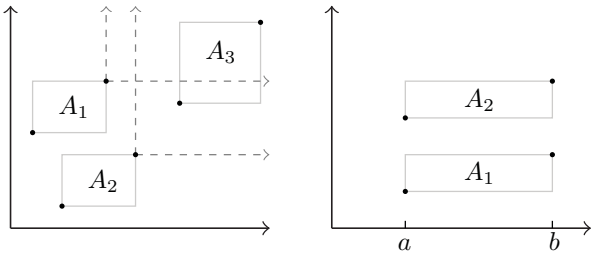

Fig. 3. Fig. (a) shows bounding boxes of three design alternatives in a 2-dimensional configuration space. Only $A_2 \prec^* A_3$; the other pairs of design alternatives are not related. Fig. (b) shows two bounding boxes with the same range in the horizontal dimension but which are unrelated nevertheless.

Theorem 1: Let M be a model and let $A_1, A_2 \in \mathit{dspace}(M)$. Then $A_1 \preceq^* A_2$ if and only if $\mathit{BB}(\mathit{config}(A_1)) \preceq^* \mathit{BB}(\mathit{config}(A_2))$.

Fig. 3 shows examples of how the dominance relation $\preceq^*$ can be represented by bounding boxes.

The quotient design space together with the induced partial order on it form a quantity. Together with the finite bounding box representation this can be used for modeling and analysis using the framework of [1]. Note, however, that there is only a single quantity: the quotient design space, and *not* the various dimensions of the configuration space. This is the result of the approach to model uncertainty as sets of configurations. The remainder of this section presents an alternative finite representation, interval representation, based on the original quantities in the configuration space. This gives rise to a multi-dimensional interval configuration space that fits the framework of [1] and in which the original quantities are retained.

A bounding box represents intervals on each quantity. The set of *intervals* of quantity Q is the following:

$$\mathbb{V}_Q = \{(x, y) \in Q \times Q \mid x \preceq_Q y\}$$

A partial order can be defined on the set of intervals, which makes the pair $(\mathbb{V}_Q, \leq)$ a quantity in the sense of [1].

Definition 7 (Interval order): Let $(x, y), (x', y') \in \mathbb{V}_Q$. Then $(x, y) \leq (x', y')$ if and only if $(x, y) = (x', y') \vee y \preceq_Q x'$.

Proposition 3: The $\leq$ relation is a partial order on $\mathbb{V}_Q$.

The next definition gives the interval representation of a non-deterministic configuration. Since intervals of quantities are quantities themselves, the interval representation gives a configuration in the sense of [1].

Definition 8 (Interval Representation): Let $C \subseteq Q_1 \times \cdots \times Q_n$ be a non-deterministic configuration. The interval representation, denoted by *interval*(C), is the tuple $((x_1, y_1), \ldots, (x_n, y_n)) \in \mathbb{V}_{Q_1} \times \cdots \times \mathbb{V}_{Q_n}$, where $x_j = \mathit{inf}(\{c[j] \mid c \in C\})$, and $y_j = \mathit{sup}(\{c[j] \mid c \in C\})$.

The order on intervals is almost equivalent to the non-deterministic dominance relation. For one of the directions of the equivalence the assumption is needed that for every dimension there are no two design alternatives with an equal non-point interval in that dimension. This avoids that using the interval representation design alternatives are dominated that would not be dominated in the original definition of non-deterministic dominance (Def. 5). For instance, Fig. 3(b) shows the bounding boxes of design alternatives A_1 and A_2. Because of reflexivity of $\leq$ it holds that *interval*(*config*$(A_1)) \leq$ *interval*(*config*(A_2)), whereas $A_1 \not\preceq^* A_2$. Despite this problem, the interval representation is very useful since it gives rise to a *multi-dimensional* interval configuration space whereas the bounding box representation collapses all dimensions into a single dimension. Multiple dimensions enable the full power of the framework of [1].

The assumption can be verified by a tool that uses the interval representation. Moreover, it can be justified by the observation that the non-point intervals usually originate from dynamic quantities (also see Def. 2). Since there is uncertainty, it can be assumed that no two design alternatives have the *exact* same bounds for a dynamic quantity: if the analysis tools tell us this nevertheless, then this is an artifact from the modeling process which can be corrected automatically.

Theorem 2: Let M be a model and let $A_1, A_2 \in$ *dspace*(M). If for every dimension holds that no two design alternatives have the same non-point interval representation for that dimension, then $A_1 \preceq^* A_2$ if and only if *interval*(*config*$(A_1)) \leq$ *interval*(*config*(A_2)).

Let M be a model. The interval space of M, denoted by *ispace*(M), is the set $\{$*interval*(*config*$(A)) \mid A \in$ *dspace*$(M)\}$. From now on it is assumed that the interval space of a model satisfies the condition stated in Thm. 2.

The work presented in this paper has been added to the OCTOPUS toolset [15]. This toolset can be used to model and analyze the DSE problem of the running example. The UPPAAL [16] backend has been used to analyze the latency bounds of all 80 design alternatives (a matter of seconds on a regular personal computer). The Pareto frontier consists of 23 design alternatives, and many design alternatives have overlap w.r.t. their latency values.

C. Pareto frontiers and refinements

Refinement reduces the amount of uncertainty. This subsection studies the effects of refinement on the *sufficiency* and *necessity* of the Pareto frontier. Informally, sufficiency means that the Pareto frontier contains enough design alternatives, and necessity means that it does not contain too many design alternatives. The following theorem states that the Pareto frontier shows all design alternatives that can be optimal after refinement. I.e., refinement does not add Pareto optimal design alternatives that were not Pareto optimal previously.

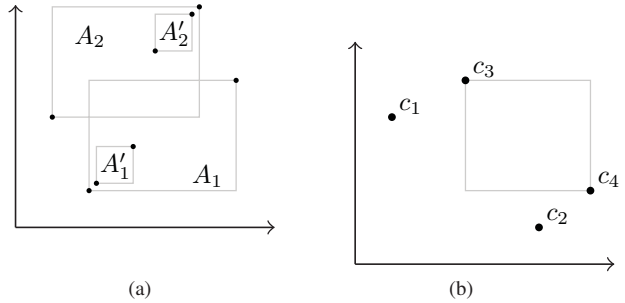

Fig. 4. Fig. (a) shows the bounding box representations of four design alternatives. Fig. (b) shows a problematic setting for refinement and dominance.

Theorem 3 (Sufficiency): Let M and M' be models such that $M' \sqsubseteq M$, let $A \in$ *dspace*(M), let $A' \in$ *dspace*(M'), and let $A' \sqsubseteq A$. If $\overline{A'}$ is Pareto optimal in *qspace*(M'), then $\overline{A}$ is Pareto optimal in *qspace*(M).

Note that the reverse implication of Thm. 3 does not hold. Fig. 4(a) shows such a situation. Note that $A_1' \sqsubseteq A_1$ and that $A_2' \sqsubseteq A_2$. The figure shows that A_1 and A_2 are both Pareto optimal in $\{A_1, A_2\}$. However, only A_1' is Pareto optimal in $\{A_1', A_2'\}$.

The necessity of the Pareto frontier depends on whether the exact non-deterministic configurations or only their finite representations are known. In practice, often no non-deterministic configurations are computed by analysis tools other than in the form of lower and upper bounds on individual quantities. This exactly coincides with the bounding box or interval representation. If it is assumed that any behavior within the finite representation can be achieved by a refinement, then the Pareto frontier is necessary.

Theorem 4 (Necessity): If $\overline{A}$ is Pareto optimal in *qspace*(M) and every behavior in the bounding boxes of the design alternatives are possible refinements, then a $A' \sqsubseteq A$ and a cost function exist such that $\overline{A'}$ is the cost optimum in *qspace*(M').

Necessity of the Pareto frontier in case when the exact shape of the non-deterministic configuration is taken into account does not hold in general. Consider Fig. 4(b) and let $A_1 = \{c_1\}$, $A_2 = \{c_2\}$, and $A_3 = \{c_3, c_4\}$. Then $A_1 \not\preceq^* A_3$ and $A_2 \not\preceq^* A_3$, yet any "strict" refinement of A_3 is dominated by either c_1 or c_2. A cost function that would make A_3 the optimum, cannot be safe under refinement: a strict refinement of A_3 clearly is not the cost optimum because a cost function must be monotonic.

Consider the running example and suppose that the execution time of F1 and F2 is understood better because

there is more clarity about the exact implementation of the processing units: F1 takes between 89 and 91 time units, and F2 takes between 109 and 111 time units per unit of load. The Pareto frontier now consists of 12 instead of 23 design alternatives and this shows that refinement (or the reduction of uncertainty) can easily remove design alternatives from the Pareto front. Before refinement there are 8 Pareto optimal design alternatives with a memory size of 10, and after refinement there is only a single Pareto optimal design alternative with a memory size of 10.

D. Refinements and cost functions

Tradeoff options based on information with uncertainty may cease to be optimal choices after the amount of uncertainty is reduced. This is to be expected. The example in Fig. 4(a) can be used to show this. Suppose we take as a cost function f the value in the horizontal dimension and let $F(A)$ be the worst-case value of $f(c)$ it may assume for any configuration $c \in \mathit{config}(A)$. Then $F(A_2) < F(A_1)$, but $F(A_1') < F(A_2')$. The refinement step has improved A_1 more than A_2. Thus, worst-case cost optimal non-deterministic configurations are in general not preserved by refinement. This means that in such cases, decision need to be postponed till after as much as possible refinement, as suggested by the fact that A_1 and A_2 were both Pareto optimal.

The problem of which Pareto optimal element to choose for implementation cannot be solved without knowing the exact cost function. The priority function as defined in [1], however, is safe under refinement: the set of cost optimal configurations does not grow when refinements are considered (although it can shrink as demonstrated by Fig. 4(a)).

The priority function assigns partial priorities to dimensions. Worse configuration values in low priority dimensions may be overruled by better values in high priority dimensions.

Definition 9 (Priority function [1]): Let $S = Q_1 \times \cdots \times Q_n$ be a configuration space and let $\trianglelefteq$ be a partial ordering on $\{1, 2, \ldots, n\}$. The priority function is the identity function on S, and the order is defined as $c \preceq_p c'$ if and only if for every $1 \leq k \leq n$ $c[k] \preceq c'[k]$ or some m exists such that $m \trianglelefteq k$ and $c[m] \prec c'[m]$.

The following theorem states that the priority function can be used for decisions on the high level. It implies that the solutions on the high-level which are not priority optimal can be discarded safely since these solutions will not become priority optimal after refinement. Note that this theorem applies to the interval representation, because this is the only representation with a multi-dimensional configuration space.

Theorem 5: Let M and M' be models, let $M' \sqsubseteq M$, let $A \in \mathit{dspace}(M)$, let $A' \in \mathit{dspace}(M')$ and let $A' \sqsubseteq A$. If $\mathit{interval}(\mathit{config}(A'))$ is priority optimal in $\mathit{ispace}(M')$, then $\mathit{interval}(\mathit{config}(A))$ is priority optimal in $\mathit{ispace}(M)$.

Consider the refinement of the running example. The priority function can be applied to make latency more important than memory. In that case, only one design alternative (the one with memory size 10) is filtered out because the others all overlap w.r.t. latency and thus 11 design alternatives remain, and none has a memory size of 10. On the other hand, if memory is more important than latency, then 11 of the 12 Pareto optimal design alternatives are filtered out because there was only a single Pareto optimal solution with a memory size of 10. Note that the task priorities have not yet been specified in the model used in the running example. This choice was motivated as a means to trade design space size for model size. The Pareto frontier now consists of 11 design alternatives. In order to find an optimal task priority assignment only these 11 design alternatives need to be considered further. This gives $11 \times 24 = 264$ refinements. Thus, in order to analyze the design space a total of $80 + 264 = 344$ dynamic analysis runs of the UPPAAL model checker are needed. This saves a factor of $\frac{1{,}920}{344} = 5.58$ compared to the situation in which the priority assignments are added directly as a model parameter. This is an effective method to attack the design space explosion problem, especially if the underlying analysis tools do not suffer from additional state space load. This is the case for simulation and sometimes even for model checking (if the state space is small or is represented symbolically).

VI. CASE STUDY

This section presents an industrial case study of the datapath of a professional printer under development from Océ technologies (*http://www.oce.nl*). The datapath is the digital image processing part of the machine. It contains image processing steps such as, for example, scaling and sharpening. The datapath is implemented on a general purpose PC which is connected to the scanner and printer hardware via dedicated PCIe boards. Fig. 5 shows an informal representation of the copy application and its mapping to the execution platform.

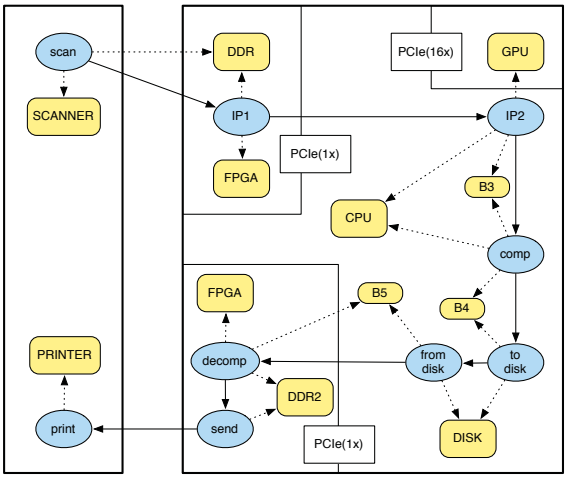

Fig. 5. The copy application (ellipses) and its mapping to the platform.

A copy job starts with scanning a page (left upper corner of the figure). Then, the image data is transferred to the controller and an image processing step is performed on the PCIe board. The second image processing step is performed on the GPU.

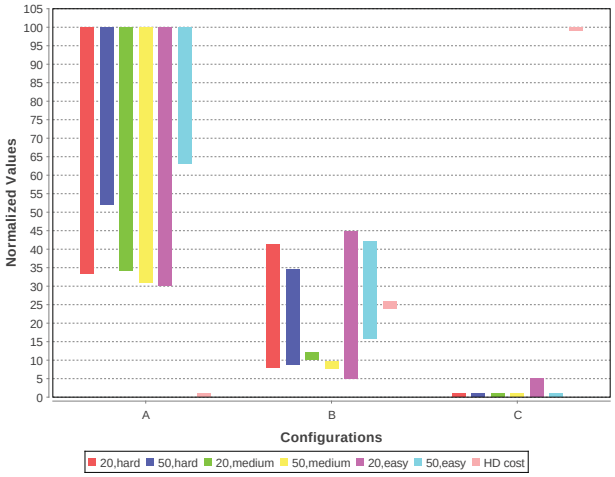

Fig. 6. The tradeoff between harddisk cost and latency in a "parallel coordinates" view.

The image data then is compressed and is written to a harddisk. It then is read again from the harddisk, decompressed and sent to the printer hardware. The various steps have fixed-size RAM buffers between them to transfer the image data (B3, B4 and B5). The tasks work on various levels of abstraction: scan, print, IP1, and IP2 work on the image data of whole pages, whereas the other tasks work on parts of the page's image data in order to benefit from pipelining.

The behavior of the system is influenced by the type of pages that are copied. Pages with high-resolution photographs are more difficult to compress than pages with only plain text. This means that the amount of data that is processed by the harddisk may vary significantly per page. Furthermore, the speed of the harddisk is influenced by the load on it. If data is read while at the same time data is written, then the speed drops. Thus, the input of the system (the pages to copy) can effect system latency.

A choice needs to be made between various harddisks, each with a different cost. I.e., the design space consists of 3 design alternatives, A, B and C. Alternative A has the slowest disk, and C has the fastest disk. An important performance metric is the system latency for copying a certain number of pages. As explained above, the latency may depend on the complexity of the individual pages. In order to get a representative overview, 180 typical copy jobs have been selected for analysis: 30 jobs with 20 easy to compress pages, 30 jobs with 20 hard to compress pages, 30 jobs with 20 pages that are neither easy nor hard to compress, and easy, medium and hard sets with 30 jobs for 50 pages. A configuration space with 7 dimensions is created. The minimal and maximal system latency in each of the 6 sets of 30 jobs is used to create latency intervals. (Note that the latency of the individual jobs also is uncertain.)

The OCTOPUS toolset implements the approach for tradeoff analysis with uncertainty and has been used to model and analyze the system. Fig. 6 shows the results: all three design alternatives are on the Pareto frontier. Design alternative C clearly has the lowest latency. However, it does not dominate the other design alternatives because these have a cheaper hard disk. Application of the priority filter with the priority that any of the latency dimensions is more important than harddisk cost gives design alternative C as the unique best choice.

VII. CONCLUSIONS

Pareto analysis is a broadly applicable method to model and analyze tradeoffs in multi-objective optimization problems and it is used in many engineering disciplines. The present paper introduces a way to explicitly take uncertainty into account and has been applied to the domain of model based design space exploration of embedded systems. The proposed method has been implemented in the OCTOPUS toolset. Furthermore, the running example shows a technique to trade design space size for model size by leaving design parameters unspecified at first. The example shows that this can potentially be very effective if the underlying analysis tools do not suffer much from the additional state space load. Finally, an industrial case study showed the applicability of Pareto analysis with uncertainty.

REFERENCES

[1] M. Geilen et al., "An algebra of pareto points," *Fundamenta Informaticae*, vol. 78, no. 1, pp. 35–74, 2007.
[2] V. Pareto, *Manual of political economy (manuale di economia politica)*. New York: Kelley, 1971 (1906), translated by A. S. Schwier and A. N. Page.
[3] A. A. Giunta et al., "Perspectives on optimization under uncertainty: Algorithms and applications," in *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2004.
[4] S. Ferson et al., "Summary from the epistemic uncertainty workshop: consensus amid diversity," *Reliability Engineering & System Safety*, vol. 85, no. 1-3, pp. 355 – 369, 2004.
[5] C. Haubelt and J. Teich, "Accelerating design space exploration using pareto-front arithmetics," in *Proc. ASP-DAC'03*. ACM, 2003.
[6] J. Teich, "Pareto-front exploration with uncertain objectives," in *EMO'01*, ser. LNCS. Springer, 2001, pp. 314–328.
[7] P. Limbourg, "Multi-objective optimization of problems with epistemic uncertainty," in *EMO*, ser. Lecture Notes in Computer Science, C. A. C. Coello, A. H. Aguirre, and E. Zitzler, Eds., vol. 3410. Springer, 2005, pp. 413–427.
[8] L. de Alfaro and T. Henzinger, "Interface automata," in *Foundations of Software Engineering (FSE)*. ACM Press, 2001.
[9] M. Broy and K. Stølen, *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. Springer, 2001.
[10] A. David et al., "Timed I/O automata: a complete specification theory for real-time systems," in *Proc HSCC 2010*. ACM, 2010, pp. 91–100.
[11] S. Cattani and M. Kwiatkowska, "A refinement-based process algebra for timed automata," *Form. Asp. of Computing*, vol. 17, no. 2, pp. 138–159, 2005.
[12] E. Brinksma and A. Mader, "On verification modelling of embedded systems," 2004, university of Twente. CTIT technical report series 04-03, http://doc.utwente.nl/48688/.
[13] S. Mohalik et al., "Model checking based analysis of end-to-end latency in embedded, real-time systems with clock drifts," in *DAC '08: Proceedings of the 45th annual Design Automation Conference*. New York, NY, USA: ACM, 2008, pp. 296–299.
[14] T. Basten et al., "Model-driven design-space exploration for embedded systems: The Octopus toolset," in *ISoLA 2010*, ser. LNCS, vol. 6415. Springer, 2010, pp. 90–105.
[15] G. Behrmann et al., "Uppaal 4.0," in *QEST 2006*. IEEE CS, 2006, pp. 125–126.

APPENDIX: PROOFS

Proposition 1: The $\preceq^*$ relation is a pre-order on the design space.

Proof: The relation is reflexive by definition. In order to prove transitivity, suppose that $A_1 \preceq^* A_2$ and $A_2 \preceq^* A_3$. If $A_1 = A_2$ or $A_2 = A_3$ then transitivity follows trivially. Suppose that these equalities do not hold. By transitivity of the relation $\preceq$ on configurations it follows that $A_1 \preceq^* A_3$. ■

Proposition 2: If $A_1 \equiv A_2$, then $\mathit{config}(A_1) = \mathit{config}(A_2) = \{c\}$ for some configuration c.

Proof: Let $c_1 \in \mathit{config}(A_1)$ and let $c_2 \in \mathit{config}(A_2)$. By Def. 5 it holds that $c_1 \preceq c_2$ and $c_2 \preceq c_1$. The $\preceq$ relation is anti-symmetric because it is a partial order. Therefore, $c_1 = c_2$. Thus, for all $c_1 \in \mathit{config}(A_1)$ and for all $c_2 \in \mathit{config}(A_2)$ holds that $c_1 = c_2$. Therefore, $\mathit{config}(A_1) = \mathit{config}(A_2) = \{c\}$ for some c. ■

Theorem 1: Let M be a model and let $A_1, A_2 \in \mathit{dspace}(M)$. Then $A_1 \preceq^* A_2$ if and only if $\mathit{BB}(\mathit{config}(A_1)) \preceq^* \mathit{BB}(\mathit{config}(A_2))$.

Proof:

($\Rightarrow$) Suppose that $A_1 = A_2$ or for all $c \in \mathit{config}(A_1)$ and $c' \in \mathit{config}(A_2)$ holds that $c \preceq c'$. The first case proves the implication by definition. Now assume towards a contradiction that $\mathit{sup}(\mathit{config}(A_1)) \not\preceq \mathit{inf}(\mathit{config}(A_2))$. This means that $\mathit{inf}(\mathit{config}(A_2)) \prec \mathit{sup}(\mathit{config}(A_1))$. Thus, there exists a dimension k of the configuration space, a $c \in \mathit{config}(A_1)$ and a $c' \in \mathit{config}(A_2)$ such that $c'[k] \prec_{Q_k} c[k]$. Thus, $c \not\preceq c'$ which contradicts the second case.

($\Leftarrow$) Suppose that $\mathit{sup}(\mathit{config}(A_1)) \preceq \mathit{inf}(\mathit{config}(A_2))$. Let $c \in \mathit{config}(A_1)$ and let $c' \in \mathit{config}(A_2)$. By Def. 6 $c \preceq \mathit{sup}(\mathit{config}(A_1))$ and $\mathit{inf}(\mathit{config}(A_2)) \preceq c'$. From transitivity of the $\preceq$ relation follows $c \preceq c'$. ■

Proposition 3: The $\leq$ relation is a partial order on $\mathbb{V}_Q$.

Proof: Reflexivity follows by definition. Next, anti-symmetry must be proven: $(x, y) \leq (x', y')$ and $(x', y') \leq (x, y)$ implies that $(x, y) = (x', y')$. Assume towards a contradiction that $(x, y) \neq (x', y')$. Then $x \neq x'$ or $y \neq y'$. Expansion of $(x, y) \leq (x', y')$ and $(x', y') \leq (x, y)$ give that $y \preceq x'$ and $y' \preceq x$. Furthermore, from the definition of $\mathbb{V}_Q$ follows that $x \preceq y$ and $x' \preceq y'$. Thus, $x \preceq y \preceq x' \preceq y' \preceq x$ which gives us that $x = x'$ and $y = y'$ which contradicts the assumption. Finally, transitivity is proven: $(x, y) \leq (x', y')$ and $(x', y') \leq (x'', y'')$ implies that $(x, y) \leq (x'', y'')$. Distinguish two cases: (a) $(x, y) = (x', y')$ or $(x', y') = (x'', y'')$ and (b) $(x, y) \neq (x', y')$ and $(x', y') \neq (x'', y'')$. Transitivity trivially follows from (a). Expansion of $\leq$ of the premisses in case (b) gives that $y \preceq x'$ and $y' \preceq x''$. From the definition of $\mathbb{V}_Q$ follows that $x' \preceq y'$. Therefore, $y \preceq x''$ which proves transitivity. ■

Theorem 2: Let M be a model and let $A_1, A_2 \in \mathit{dspace}(M)$. If for every dimension holds that no two design alternatives have the same non-point interval representation for that dimension, then $A_1 \preceq^* A_2$ if and only if $\mathit{interval}(\mathit{config}(A_1)) \leq \mathit{interval}(\mathit{config}(A_2))$.

Proof:

($\Rightarrow$) Suppose that $A_1 = A_2$ or $\mathit{config}(A_1) \preceq^* \mathit{config}(A_2)$. From both cases follows that $\mathit{config}(A_1) \preceq^* \mathit{config}(A_2)$. Thm. 1 then gives that $\mathit{sup}(\mathit{config}(A_1)) \preceq \mathit{inf}(\mathit{config}(A_2))$. From Def. 8 follows $\mathit{interval}(\mathit{config}(A_1)) \leq \mathit{interval}(\mathit{config}(A_2))$.

($\Leftarrow$) Suppose that $\mathit{interval}(\mathit{config}(A_1)) \leq \mathit{interval}(\mathit{config}(A_2))$ which is to say that $((x_1, y_1), \ldots, (x_n, y_n)) \leq ((x'_1, y'_1), \ldots, (x'_n, y'_n))$. Thus, $(x_i, y_i) \leq (x'_i, y'_i)$ for all $1 \leq i \leq n$. This means by Def. 7 that (a) $(x_i, y_i) = (x'_i, y'_i)$ or (b) $y_i \preceq_{Q_i} x'_i$. The assumption of the theorem applies to (a) and gives that $x_i = y_i = x'_i = y'_i$ and thus $y_i \preceq_{Q_i} x'_i$. From case (b) follows directly that $y_i \preceq_{Q_i} x'_i$. This means that $\mathit{sup}(\mathit{config}(A_1)) \preceq \mathit{inf}(\mathit{config}(A_2))$. Thus, by Thm. 1 $A_1 \preceq^* A_2$. ■

Theorem 3 (Sufficiency): Let M and M' be models, let $A \in \mathit{dspace}(M)$, let $A' \in \mathit{dspace}(M')$, let $M' \sqsubseteq M$, and let $A' \sqsubseteq A$. If $\overline{A'}$ is Pareto optimal in $\mathit{qspace}(M')$, then $\overline{A}$ is Pareto optimal in $\mathit{qspace}(M)$.

Proof: Suppose that $\overline{A'}$ is Pareto optimal in $\mathit{qspace}(M')$, which means that for all $\overline{B'} \neq \overline{A'} \in \mathit{qspace}(M')$ holds that $\overline{B'} \not\preceq \overline{A'}$. Assume towards a contradiction that $\overline{A}$ is not Pareto optimal in $\mathit{qspace}(M)$. This means that it is strictly dominated: $\overline{B} \prec \overline{A}$ for some B. This on its turn means that for all $c \in \mathit{config}(B)$ and for all $c' \in \mathit{config}(A)$ holds that $c \preceq c'$. Let $B' \in \mathit{dspace}(M')$ such that $B' \sqsubseteq B$. By Def. 3 it holds that $\mathit{config}(B') \subseteq \mathit{config}(B)$ and also that $\mathit{config}(A') \subseteq \mathit{config}(A)$. Therefore, for all $c \in \mathit{config}(B')$ and for all $c' \in \mathit{config}(A')$ holds that $c \preceq c'$. Thus, $\overline{B'} \preceq \overline{A'}$ which contradicts Pareto optimality of $\overline{A'}$. ■

Theorem 4 (Necessity): If $\overline{A}$ is Pareto optimal in $\mathit{qspace}(M)$ and every behavior in the bounding boxes of the design alternatives are possible refinements, then a $A' \sqsubseteq A$ and a cost function exist such that $\overline{A'}$ is the unique cost optimum in $\mathit{qspace}(M')$.

Proof: Take the refinement such that $\mathit{config}(A') = \mathit{inf}(\mathit{config}(A))$ and $\mathit{config}(A'') = \mathit{sup}(\mathit{config}(A'''))$ for all $A'' \neq A' \in \mathit{qspace}(M')$. Then A' is Pareto optimal in $\mathit{qspace}(M')$. Assume towards a contradiction that it is not: $A'' \prec A'$ for some $A'' \in \mathit{qspace}(M')$. Then A is not Pareto optimal in $\mathit{qspace}(M)$ which is a contradiction. This refinement thus gives point configurations for each design alternative. The proof in [1] then constructs a cost function that makes $\overline{A'}$ the cost optimum. ■

Theorem 5: Let M and M' be models, let $M' \sqsubseteq M$, let $A \in \mathit{dspace}(M)$, let $A' \in \mathit{dspace}(M')$ and let $A' \sqsubseteq A$. If $\mathit{interval}(\mathit{config}(A'))$ is priority optimal in $\mathit{ispace}(M')$, then $\mathit{interval}(\mathit{config}(A))$ is priority optimal in $\mathit{ispace}(M)$.

Proof: Assume towards a contradiction that $\mathit{interval}(\mathit{config}(A))$ is not priority optimal in $\mathit{ispace}(M)$. Then some B exists such that $\mathit{interval}(\mathit{config}(B)) \prec_p$

interval(*config*(A)). Consider a refinement of B which is B'. Define the following abbreviations: $a =$ *interval*(*config*(A)), $a' =$ *interval*(*config*(A')), $b =$ *interval*(*config*(B)), and $b' =$ *interval*(*config*(B')). Then (1) $b[k] \leq a[k]$ or (2) some m exists such that $m \trianglelefteq k$ and $b[m] < a[m]$ by Def. 9. In the first case $b'[k] \leq a'[k]$ because the intervals become smaller by refinement. Therefore, *interval*(*config*(A')) is not priority optimal in *ispace*(M'), which is a contradiction. In the second case also $b'[m] < a'[m]$ by a similar argument, which again gives a contradiction. ■