Describing Problem Solving Methods using Anytime Performance Profiles

Annette ten Teije[1] and **Frank van Harmelen**[2]

Abstract. We propose the use of anytime performance profiles to describe the computational behaviour of problem solving methods. A performance profile describes how the quality of the output of an algorithm gradually increases as a function of the computation time. Such anytime descriptions of problem solving methods are attractive because they allow a trade-off to be made between available computation time and output-quality. It turns out that many problem solving methods found in the literature have a natural anytime behaviour, which has remained largely unexploited until now.

In this paper we propose an axiomatic description of performance profiles. Furthermore, we give a fixed schematic form for these axiomatic descriptions. Finally, we apply our proposal to a number of realistic problem-solving methods, namely hierarchical classification (used in MDX), and parametric design methods from XCON and VT.

1 Motivation

A dominant theme in the Knowledge Engineering literature on Problem Solving Methods (PSMs) in the last decade has been the description of the "competence" of PSMs, ie. the functional I/O relation of methods. Typically, a method is regarded as a functional I/O relation between the given domain knowledge and the required goals of the method. Much of the more recent work on characterising PSMs still focuses exclusively on the "competence" of the PSM [12].

This leaves an entire dimension of PSMs uncovered in the literature: *how should the **computational performance** of a PSM be described?*

In this paper we propose the use of so-called *anytime performance profiles* [3] to describe the computational behaviour of PSMs (see section 2). Traditionally, such performance profiles are given in the form of graphs which are obtained empirically by executing the PSM. We propose an axiomatic description of performance profiles (section 3). Our descriptions always consist of the same four elements, each of which must be provided when characterising method performance. Finally (section 4), we apply our proposal to a number of realistic problem-solving methods, namely for hierarchical classification, and parametric design (methods from XCON and VT).

2 What is anytime reasoning?

Anytime algorithms are defined as algorithms that return some answer for any allocation of computation time, and are expected to return better answers when given more time [1]. This is in contrast with traditional algorithms which guarantee a correct output only after termination, and no guarantees are given for any intermediate results. The behaviour of an anytime algorithm is described by a *performance profile*. A performance profile describes how the quality of the output of the algorithm varies as a function of the computation time. The quality measure of the output may be any characteristic of the result of an algorithm that we find significant. One anytime algorithm could have several performance profiles tracking different attributes of the results it returns. A performance profile is typically given in the form of a graph that plots output quality against runtime.

It is clear that anytime behaviour of PSMs is desirable. Such PSMs are usable even when there is insufficient time to compute complete solutions. This is often the case given the intractable nature of typical tasks for PSMs. Also, such PSMs are applicable in real-time situations when the available computation time is often short and not known in advance. Thirdly, they offer the user the possibility to trade solution-quality against computation time, making the PSMs more widely applicable when selected from a library.

Perhaps surprisingly, anytime PSMs occur frequently. Many PSMs in the literature turn out to have an anytime nature, even when they were not developed with this purpose in mind. We have analysed the PSMs from a modern textbook on knowledge-based systems [9], and have found that many of the methods discussed there have anytime behaviour. This will be illustrated in section 4, where we discuss examples which are all taken from this textbook, many of which are used in realistic KBS applications.

3 A schema for anytime descriptions

The motivating question for this paper as given in the introduction was: *how should the **computational performance** of a PSM be described?*

Instead of the empirically obtained quality-performance graphs used for this purpose in the literature, we aim for an analytic treatment of the performance profiles in the form of an axiomatic description. This has the advantages of not needing expensive and unreliable empirical performance observations, and of giving more insight in the actual behaviour of the PSM.

We have found that this formulation can always be structured in the same way. To describe the anytime functionality, four axioms are needed, each of which describes a different aspect of the anytime behaviour, as follows:

Initial behaviour: The initial period during which the behaviour of the method is constant. Many anytime algorithms start producing some output immediately, but the example in section 4.1.2 shows that some methods need an initial "startup period" before they start producing intermediate output.

[1] Dept. of CS, Univ. of Utrecht, annette@cs.uu.nl
[2] Dept. of AI, Vrije Univ. Amsterdam, frankh@cs.vu.nl

Growth direction: This is the direction in which the quality of the intermediate output changes with increasing runtime.

Growth rate: The amount of increase in quality at each step during the computation. This increase in quality can be constant at each step, but may also vary during the computation

End condition: The amount of runtime needed for the method to achieve its full (ie. traditional) functionality. After this point, the quality of the output no longer increases, since the maximum quality has been achieved.

In [4] we have put forward the hypothesis that this scheme of four axioms would be suitable to describe a wide class of anytime behaviours. One of the results of this paper is the confirmation of this hypothesis, by showing that this scheme can be used to describe the anytime behaviour of a number of different and realistic PSMs from the literature.

4 Example anytime descriptions of PSMs

In this section we apply the above scheme for describing performance profiles of PSMs to a number of concrete PSMs. These methods are all described in a modern KBS textbook [9, Part III].

4.1 The classification task

In a classification task, we are given a set of candidate classes and a set of observed properties of a particular individual, and we must compute which candidate classes satisfy the classification criterion on the given properties. The details of the classification criterion can vary, and are not relevant to our discussion (see [9, Ch. 7] for the definition of various classification criteria such as candidates which explain, match or cover the given observations).

Slightly more formally, the classification task has as inputs a set of candidate classes *Cs* and a set of observations *Obs*, and must compute all classes from *Cs* that satisfy the classification criterion on *Obs*: $\{C_i | C_i \in Cs \wedge criterion(C_i, Obs)\}$.

4.1.1 Linear candidate confirmation (MC1)

A trivial PSM for the classification task is to iterate over all candidate classes, and add them to the output if they satisfy the classification criterion[3]:

```
MC1($\boxed{\text{n,}}$Cs,Obs):
  output = $\emptyset$
  candidates = {C$_i$ | C$_i$ $\in$ Cs $\boxed{\wedge\ i \leq n}$}
  for C$_i$ $\in$ candidates
  do if criterion(C$_i$,Obs)
     then output = output+C$_i$
  done
  return output
```

The algorithm MC1 is as given without the additional boxed text. If the boxed text is added to the code, we obtain an anytime version of MC1 that we will indicate with $\widetilde{MC}1$. The additional parameter n indicates that the algorithm will terminate after n steps (clearly larger values of n require more runtime). This implies that we model our PSMs as *contract-algorithms* with the amount of available runtime specified in advance. We can do this without loss of generality, since [8] shows that such contract-algorithms can be converted to interruptable algorithms at the cost of a constant factor of 4.

[3] This method is called MC1 in [9, Ch. 7].

Following the guidelines from the previous section, the gradual functionality of $\widetilde{MC}1$ can now be specified as follows:

MC1-initial: Initially no solutions are computed:
$\widetilde{MC}1(0, Cs, Obs) = \emptyset$

MC1-direction: The solution set only grows (and never decreases):
$\widetilde{MC}1(n, Cs, Obs) \subseteq \widetilde{MC}1(n+1, Cs, Obs)$

MC1-rate: Each additional step adds at most one solution:
$|\widetilde{MC}1(n+1, Cs, Obs)| - |\widetilde{MC}1(n, Cs, Obs)| \leq 1$

MC1-end: After considering all candidates, we have obtained the full functionality:
$n \geq |Cs| \rightarrow \widetilde{MC}1(n, Cs, Obs) = MC1(Cs, Obs)$

Assuming a uniform distribution of the solutions over the candidate set, the theoretically derived performance profile determined by these axioms is shown in figure 1a[4].

4.1.2 MC1 with forward filtering (MC2)

This method is equal to MC1, but first applies an initial forward reasoning step, in which some of the observations are used to filter the set of possible candidates. The method MC1 is applied to the resulting candidate set. Instead of a single set of observations, MC2 receives two sets of observations as input, one to be used in the forward filtering step, the other to be used in the candidate confirmation step:

```
MC2($\boxed{\text{n,}}$Cs,Obs$_1$,Obs$_2$):
  output = $\emptyset$
  candidates = {C$_i$|C$_i$ $\in$filter(Cs,Obs$_1$) $\boxed{\wedge\ i \leq n}$}
  for C$_i$ $\in$ candidates
  do if criterion(C$_i$,Obs$_2$)
     then output = output+C$_i$
  done
  return output
```

Following the guidelines from the previous section, the gradual functionality of $\widetilde{MC}2$ can now be given informally as follows:

The performance profile of MC2 is shown in figure 1b. It shows a constant increase in quality (similar to MC1), but only after an initial period needed for the filtering step.

Following the guidelines from the previous section, the gradual functionality of $\widetilde{MC}2$ can now be specified as follows:

MC2-initial: Unlike MC1, MC2 does not immediate produce intermediate solutions, since it first completes the forward filtering step. If n_f indicates the duration of the filtering step, then:
$n < n_f \rightarrow \widetilde{MC}2(n, Cs, Obs_1, Obs_2) = \emptyset$

MC2-direction: similar to [MC1-direction].

MC2-rate: similar to [MC1-rate].

MC2-end: The total required runtime of MC2 is the sum of the two stages. The duration of the filtering step is n_f, and the duration of the confirmation stage is determined by the number of candidates that remain after the filtering step:
$n \geq |filter(Cs, Obs_1)| + n_f \rightarrow$
$\widetilde{MC}2(n, Cs, Obs_1, Obs_2) = MC2(Cs, Obs_1, Obs_2)$

The performance profile determined by these axioms is shown in figure 1b. It shows a constant increase in quality (similar to MC1), but only after an initial period needed for the filtering step.

[4] Our graphs are plotted as continuous functions, but in reality the increases in output quality are stepwise.

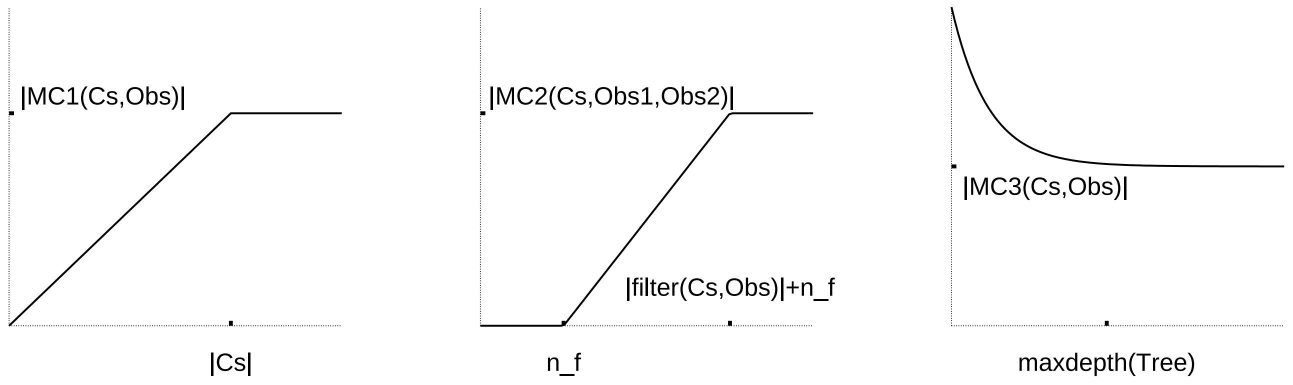

Fig. (a) Fig. (b) Fig. (c)

Figure 1. Performance profile of MC1, MC2 and MC3.

$\widetilde{MC}2$ is only attractive if the costs of the filter step (n_f) are outweighed by its savings (which equal the reduction in the candidate set: $|Cs| - |filter(Cs, Obs)|$), ie. when $n_f < |Cs| - |filter(Cs, Obs)|$, or equivalently $n_f + |filter(Cs, Obs)| < |Cs|$. Now notice that this states precisely that the end time of MC2 is less then the end time of MC1 (see axioms [MC1-end] and [MC2-end]).

4.1.3 *Hierarchical classification (MC3)*

The first realistic PSM that we will discuss is hierarchical classification (used among others in the MDX system [2]). This method no longer does a linear traversal of the candidate set. Instead, the candidate set is organised as the leaves of a tree. The nodes in this tree are "abstract classes", representing abstractions of sets of candidates. The PSM recursively descends down the tree, at each level deciding if the abstract classes still satisfy the observations. If yes, the method continues to descend down that part of the tree, if no, the entire tree below the abstract class is pruned. At each step of the PSM, the intermediate solution is the set of all candidates (= all leaves) that can be found under the currently considered abstract classes.

```
MC3($\boxed{n,}$Tree,Obs):
   output = leaves(Tree)
   current = {Tree}
   next = []
   $\boxed{d=1}$
   while current $\neq \emptyset$ $\boxed{\wedge\ d \leq n}$
   do for c $\in$ current
      do if   not criterion(c,Obs)
         then output = output - leaves(c)
         else next = next + children(c)
      done
      current = next - c
      next = $\emptyset$
      $\boxed{d = d+1}$
   done
   return output
```

The gradual functionality of MC3 can be characterised as follows:

MC3-initial: Initially, all candidates are still potential solutions:
$\widetilde{MC}3(0, Tree, Obs) = leaves(Tree)$

MC3-direction: an extra computation step can only decrease the set of potential solutions:
$\widetilde{MC}3(n+1, Tree, Obs) \subseteq \widetilde{MC}3(n, Tree, Obs)$

MC3-rate: Taking b as the branching factor of the tree, and assuming that at each level of the tree, at least 1 of the abstract classes satisfies the criterion, and assuming a balanced tree, then each step reduces the candidate set by a factor b (assuming a uniform reduction rate while descending the tree):
$|\widetilde{MC}3(n, Tree, Obs)| \geq |\widetilde{MC}3(n+1, Tree, Obs)| \geq \frac{|\widetilde{MC}3(n,Tree,Obs)|}{b}$

MC3-end: MC3 needs as many steps as there are levels in the tree
$n \geq maxdepth(Tree) \rightarrow$
$\widetilde{MC}3(n, Tree, Obs) = MC3(Tree, Obs)$

Using the performance profiles. We have now defined anytime performance profiles for three different classification methods. We give three examples how these profiles help us with choosing which method to apply. First, the profiles tell us how we can trade computation time for solution quality. For example, fig 1c shows that the increase in quality of MC3 (ie the decrease in the candidate set) is exponential while for MC1 and MC2 this linear. If it is important to quickly obtain a good approximation of the final solution, then MC3 is more attractive than MC1 or MC2.

Secondly, MC1 and MC2 are incomplete (but sound) approximations of the final solution. The intermediate solutions of MC3 on the other hand are unsound (but complete) approximations of the final solution, since MC3 approaches the solutions from above, rather than from below.

Thirdly, we see that not all methods start to produce approximate solutions immediately. If such a property is important (e.g. in a setting where some solution is always required but no guarantees can be given on the available runtime), then MC2 is unattractive.

4.2 The parametric design task

We now turn to a very different type of task, namely the task of parametric design. In this task we are given a set of parameters and a set of constraints, and have to compute an assignment for each parameter such that these assignments are consistent with the given constraints.

Slightly more formally, given a set of parameters Ps and a set of constraints Cs, we have to compute a set of parameter assignments S with $S = \{(P_i, V_i) | P_i \in Ps\}$ such that $consistent(Cs, S)$.

In the next two sections, we give two problem solving methods for performing this parametric design task.

4.2.1 *Param. design by constraint clustering (XCON)*

XCON [6] is a method for parametric design based on dividing the constraints in clusters. The constraints within a cluster have many mutual dependencies, but no dependencies exist with constraints in other clusters. This makes it possible to solve the problem in separate steps, with backtracking occurring only within each step (namely per cluster), and not between steps.

The XCON method simply iterates over the constraints clusters Cs_i, solves each cluster separately (no details for this step are given in the definition below), and adds the assignments found for each step to the current output:

```
XCON($\boxed{\texttt{n,}}$Ps,[Cs$_1$,...,Cs$_k$]):
   output = $\emptyset$
   for i=1 to $\boxed{\texttt{min(n,}}$k$\boxed{\texttt{)}}$
   do CurrentPs = relevant(Ps,Cs$_i$)
      S={(P$_i$,V$_i$) | P$_i$$\in$CurrentPs}
        such that consistent(Cs$_i$,S)
      output = output $\cup$ S
   done
   return output
```

The gradual functionality of XCON can be characterised as follows:

XCON-initial: Initially no assignments have been computed.
$\widetilde{XCON}(0, Ps, [Cs_1, ...]) = \emptyset$

XCON-direction: The set of assignments that is computed grows monotonically.
$\widetilde{XCON}(n, Ps, [Cs_1, ...]) \subset \widetilde{XCON}(n+1, Ps, [Cs_1, ..])$

XCON-rate: The maximal number of new assignments for a cluster is the number of variables of the constraints in that cluster. This is a maximum, since some parameters might have been computed in earlier steps (clusters).
$|\widetilde{XCON}(n+1, Ps, [Cs_1, ..])| - |\widetilde{XCON}(n, Ps, [Cs_1, ..])|$
$\leq |\text{vars}(Ps, Cs_{n+1})|$

XCON-end: The complete functionality is obtained after k steps, with k the number of constraint clusters in the input.
$n \geq k \rightarrow$
$\widetilde{XCON}(n, Ps, [Cs_1, .., Cs_k]) = XCON(Ps, [Cs_1, .., Cs_k])$

These axioms determine the performance profile, as shown in figure 2a. The graph shows a stepwise increase of the output quality in k steps.

4.2.2 *Parametric design by propose and revise (P&R)*

Another well known method for parametric design is Propose & Revise (P&R). The P&R method iterates over the set of parameters. In each step, P&R takes a new parameter and proposes a likely value for that parameter. This new assignment becomes part of the current partial design. If this partial design is consistent with the constraints, a next step can be taken. If the partial design violates the constraints then it will be revised such that it becomes consistent with the constraints. After fixing the partial design the process continues with the next parameter.

For XCON we used the set of assignments as the quality measure for the computation. This same measure cannot be used for P&R. Unlike XCON, P&R assignments in earlier steps might have to be revised in later steps. As a result, the set of assignments does not grow monotonically. For this reason, we use another quality measure for P&R, namely the set of assigned parameters instead of the set of assignments (ie parameters plus their values). This set does grow monotonically, since parameters might be revised, but once assigned, a parameter is never left without a value in later stages of the computation.

```
P&R($\boxed{\texttt{n,}}$Ps,Cs):
   output = $\emptyset$
   for i=1 to $\boxed{\texttt{min(n,}}$|Ps|$\boxed{\texttt{)}}$
   do V$_i$ = propose(output,P$_i$)
      output = output = (P$_i$,V$_i$)
      if $\neg$consistent(Cs,output)
      then output = revise(Cs,output)
   done
   return output
```

P&R-initial: Initially no assignments have been computed.
$\widetilde{P\&R}(0, Ps, Cs) = \emptyset$

P&R-direction: Unlike XCON, the set of assignments does not grow monotonically in P&R, but the set of assigned parameters does. The set $\{P_i | (P_i, V_i) \in \widetilde{P\&R}(n, Ps, Cs)\}$ contains all parameters that have been assigned a value after n steps. The monotonic growth is then:
$\{P_i | (P_i, V_i) \in \widetilde{P\&R}(n, Ps, Cs)\} \subseteq$
$\{P_i | (P_i, V_i') \in \widetilde{P\&R}(n+1, Ps, Cs)\}$

P&R-rate: P&R iterates over the set of parameters, therefore each step yields exactly one additional assigned parameter:
$|\{P_i | (P_i, V_i) \in \widetilde{P\&R}(n+1, Ps, Cs)\}| =$
$|\{P_i | (P_i, V_i) \in \widetilde{P\&R}(n, Ps, Cs)\}| + 1$

P&R-end: The method needs as many steps as there are parameters:
$n \geq |Ps| \rightarrow \widetilde{P\&R}(n, Ps, Cs) = P\&R(Ps, Cs)$

These axioms determine the performance profile, as shown in figure 2b. The set of assigned parameters grows at a constant rate during the computation.

Concerning axiom [P&R-direction]: This axiom does not require that partial assignments are a subset of the final assignments (ie partial assignments are allowed to be unsound). However, in the VT application which used the P&R method, it turns out that the domain knowledge used in the propose step is so good that revision is almost never needed: on test-cases with 1000-2000 parameters (and therefore as many propose steps), there were only some 10-20 violations (and therefore as many revision steps), ie only 1% of the proposed values were wrong [5]. Thus, although the soundness of XCON's approximations (expressed by XCON's axiom [XCON-direction]) cannot be guaranteed for P&R, in practice P&R comes very close.

Using the performance profiles. Again, the performance profiles for the different methods can be used for selecting problem solving methods. It follows from axioms [XCON-end] and [P&R-end] that P&R divides the process in $|Ps|$ steps, and XCON in k steps (k the number of constraint clusters). Since every cluster will assign at least one additional parameter, we have $|Ps| \geq k$, so P&R divides the process in much smaller steps then XCON. If it is important that new solutions are produced at a constant rate during the computation process (e.g. because of interface requirements with users or other programs), then P&R is more attractive from an anytime perspective.

5 Conclusions & Future Work

Conclusions We have given axiomatic descriptions of anytime behaviour of PSMs. This is unlike existing work on anytime algorithms, which obtains performance profiles by simulation and measurement.

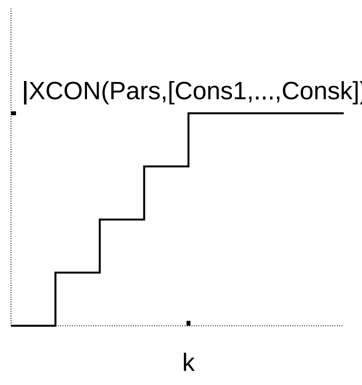

Fig (a)

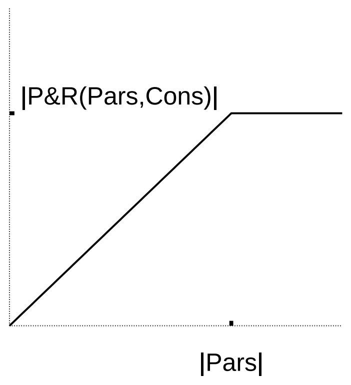

Fig (b)

Figure 2. Performance profile of XCON and P&R

Such empirically obtained profiles are dependent on the quality of the simulations, which are often expensive, and also not very reliable since they depend on the particular input distribution used for the simulations. Notice that the qualitative profiles given in the various graphs can be derived from our axiomatic descriptions (sometimes after making some plausible additional assumptions), while the converse is not true. On the other hand, our axiomatic descriptions are often limited to giving an upper- or lower-bound on the rate of the quality improvement, whereas empirical performance profiles do obtain values for the improvement rate.

We have given a scheme for constructing such an axiomatic descriptions: each description should consist of four statements, describing initial behaviour of the PSM, direction of quality change, rate of quality of change per time unit and the time at which the optimal output quality is obtained. This regularity in the description of the dynamic behaviour of PSMs confirms a hypothesis put forward in our earlier work [4].

Our axiomatic description of performance profiles is based on our earlier and more general proposal for describing gradual properties of PSMs [11]. It turns out that performance profiles can be described in our framework for describing gradual properties. This was not obvious beforehand because the framework must be used in a slightly non-standard way. It was designed to deal with functional properties (I/O-pre/post-conditions), while anytime behaviour is a non-functional property (concerning also the computation time, and not only the I/O relation).

We have applied our proposal to a number of well-known and realistic problem solving methods. To our surprise, many existing methods display a natureal anytime behaviour, although they were never designed for this purpose. All of these anytime behaviours fitted naturally in our proposed description schema.

Future Work In the axioms in this paper, we give only upper- or lowerbounds for the rate of quality improvement (the third axiom in our general scheme), Instead of such upper- and lowerbounds, we would like to give more precise *expected values* for the improvement rate.

We could investigate the use of temporal formalisms for specifying the behaviour axioms (see e.g. [10] for a survey), instead of "encoding" them in predicate logic via an additional parameter n that we had to add to each PSM in order to represent run-time.

Finally, in [4] we have developed some simple techniques for proving dynamic properties of KBS, and we used the interactive theorem prover KIV [7] to verify a simple PSM (in fact, MC1). All the formal definitions in this paper (both program code and axioms) have been given in a syntax already very close to that used in the KIV systems. We expect that our techniques can be applied to verify the axiomatic anytime descriptions given in this paper, yielding machine-assisted formal proofs of the anytime behaviour of realistic PSMs.

Acknowledgement We are grateful to Leon van der Torre for his comments on an earlier version of this paper.

REFERENCES

[1] M. Boddy and T. Dean, 'Solving time-dependent planning problems', in IJCAI'89.

[2] B. Chandrasekaran, S. Mittal, 'Deep versus compiled knowledge approaches to diagnostic problem solving', *Int. J. of Man-Machine Studies*, **19**, 425–436, (1983).

[3] T. Dean and M. Boddy, 'An analysis of time-dependent planning', in AAAI–88, pp. 49–54, 1988.

[4] P. Groot, A. ten Teije, and F. van Harmelen, 'Formally verifying dynamic properties of knowledge based systems', in EKAW '99, eds., D. Fensel and R. Studer, Springer LNAI 1621, pp. 157–172, 1999.

[5] S. Marcus, J. Stout, and J. McDermott, 'VT: An expert elevator designer that uses knowledge-based backtracking', *AI Magazine*, Spring issue, pp. 95–111, (1988).

[6] J. McDermott, 'R1: A rule-based configurer of computer systems', *Artificial Intelligence*, **19**, 39–88, (1982).

[7] W. Reif, 'The KIV approach to Software Verification', in *KORSO: Methods, Languages, and Tools for the Construction of Correct Software – Final Report*, eds., M. Broy and S. Jähnichen, Springer LNCS 1009, (1995).

[8] S. Russel and S. Zilberstein, 'Composing real-time systems', in *IJCAI'91*, pp. 212–217.

[9] M. Stefik, *Introduction to Knowledge-Based Systems*, Morgan Kaufmann, 1995.

[10] P. van Eck, J. Engelfriet, D. Fensel, F. van Harmelen, Y. Venema, and M. Willems, 'A survey of languages for specifying dynamics: A knowledge engineering perspective', *IEEE Transactions on Knowledge and Data Engineering*, (2000). to appear.

[11] F. van Harmelen and A. ten Teije, 'Characterising approximate problem-solving by partial pre- and postconditions', in ECAI'98, pp. 78–82, 1998.

[12] B.J. Wielinga, J.M. Akkermans, and A.Th. Schreiber, 'A competence theory approach to problem solving method construction', *Int. J. of Human-Computer Studies*, **49**(4), (1998).