The Epistemology of Scheduling Problems

Enrico Motta[1], Dnyanesh Rajpathak[1], Zdenek Zdrahal[1] and Rajkumar Roy[2]

Abstract. Scheduling is a knowledge-intensive task spanning over many activities in day-to-day life. It deals with the temporally-bound assignment of jobs to resources. Although scheduling has been extensively researched in the AI community for the past 30 years, efforts have primarily focused on specific applications, algorithms, or 'scheduling shells' and no comprehensive analysis exists on the nature of scheduling problems, which provides a formal account of what scheduling is, independently of the way scheduling problems can be approached. Research on KBS development by reuse makes use of *ontologies,* to provide knowledge-level specifications of reusable KBS components. In this paper we describe a *task ontology*, which formally characterises the nature of scheduling problems, independently of particular application domains and independently of how the problems can be solved. Our results provide a comprehensive, domain-independent and formally specified *reference model* for scheduling applications. This can be used as the basis for further analyses of the class of scheduling problems and also as a concrete reusable resource to support knowledge acquisition and system development in scheduling applications.

1 INTRODUCTION

Scheduling is a knowledge-intensive task spanning over many activities in day-to-day life. We talk about train schedules, job-shop scheduling problems, personnel schedules, television schedules, etc. As a first approximation we can say that scheduling deals with the *temporally bound assignment of jobs to resources and times ranges*. This time-centric dimension is essentially what distinguishes scheduling from other design problems, in particular from *parametric design* problems [1], where the assignment of values to parameters is not temporally bound. In most real-world problems, the space of admissible solutions is also restricted by the applicable constraints, often of an organisational and technological nature - e.g., in an airport-gate scheduling scenario limitations may be enforced on the type of planes that a particular gate can handle. In addition, cost criteria may also play a role, as multiple solutions can be admissible for a particular problem, and some of them can be deemed to be 'better' than others. For instance, in a manufacturing scenario, we may privilege solutions which minimise the demands on expensive machinery, or, in the opposite case, sometimes we may want to minimise 'idle time'.

Although scheduling has been extensively researched in the AI community for the past 30 years [2, 3, 4 and 11], efforts have primarily focused on specific applications, algorithms, and 'scheduling shells'. No comprehensive analysis exists on the nature of scheduling problems that provides a formal account of what scheduling is, independently of the way scheduling problems can be approached.

An *ontology* [5] can be seen as an information model that explicitly describes the various entities and abstractions that exist in a universe of discourse, along with their properties. Much work on reusable components for knowledge-based systems (KBS) [1, 6, 7] relies on ontologies to specify formally generic classes of knowledge-intensive tasks, such as parametric design or classification, as well as the ontological commitments associated with knowledge-intensive problem solvers. In the former case we use the term *task ontologies*, in the latter *method ontologies*. In this paper we turn our attention to scheduling and we describe a task ontology, which formally characterises the nature of scheduling problems. This task ontology is method, application and domain independent. In other words it describes scheduling problems, independently of particular application domains and independently of how the problems can be solved. In our view this approach provides two main benefits. From an analytical point of view it provides a formal account of a class of problems, which can be used to clarify the nature of scheduling problems and to understand the domain of applicability of existing scheduling systems. From an engineering point of view it provides a generic model which can be instantiated with application and domain specific knowledge to represent concrete application problems, without actually committing to a particular problem solving approach.

Although, there have been some attempts at developing task ontologies for scheduling [8, 9, 10], these attempts are incomplete, as they are either committed to specific domains or, when they provide the right level of domain-independence, they seem to lack the required level of detail and formalisation. Moreover, as we will discuss in detail in section 5, important ontological distinctions are typically missing in the conceptualisations proposed so far. Thus, the primary aim of our research is to put the scheduling task on firmer ontological foundations to provide an adequate theoretical as well as engineering leverage for the various classes of scheduling applications.

The rest of the paper is organised as follows. Section 2 formally specifies a generic scheduling task. In section 3, we provide an overview of the ontology and we illustrate some of the definitions. In section 4, we discuss how we have validated the ontology by using it to model two real-world problems. In section 5, we compare our work with other alternative approaches and finally, in section 6, we conclude the paper by reiterating the main contributions of this work and propose the future directions of our research.

2 A SCHEDULING TASK SPECIFICATION

A scheduling task can be formally represented as a mapping from a nine-dimensional space {J, A, R, C, Req, Tr, P, Cf, Cr} to a set of schedules S = {S_1,…., S_n}. These parameters are described as follows:

- *Jobs,* J = {j_1, …., j_m}. A set of jobs to be assigned to a set of resources for their execution.
- *Activities,* A = {a_1, …., a_n}. A set of activities. Each activity must be associated to a job, and no activity can be associated to more than one job.
- *Resources,* R = {r_1, …., r_i}. A set of resources to which the jobs can be assigned for their execution.
- *Constraints*, C = {c_1, …., c_j}. A set of constraints that must not be violated by a solution schedule.
- *Requirements,* Req = {req_1, …., req_k}. A set of requirements that describe the desired properties of a solution schedule.
- *Schedule time range,* Tr. The time horizon in which the schedule takes place.

[1] Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom.
email: {e.motta, d.g.rajpathak, z.zdrahal}@open.ac.uk
[2] Cranfield University, School of Industrial & Manufacturing Science, Cranfield, Bedford, MK43 0AL, United Kingdom.

- *Preferences,* P = {p_1, …., p_t}. A set of criteria for choosing among competing solution schedules. Each preference defines a partial order over the set of solution schedules.
- *Cost function,* Cf. A function, which computes the cost of a solution schedule.
- *Solution criterion,* Cr. A mapping from S to {True, False}, which determines whether a candidate schedule is a solution. The minimal set of conditions imposed by a solution criterion on a solution schedule, say S_i, normally requires S_i to be *correct*, *complete*, *admissible,* and *feasible*- see below for the definitions of these properties. More restrictive solution criteria may introduce optimality conditions based on the applicable preferences and cost function.
- *Schedule*, S = {…<j_i, a_{ij}, r_k, tr_{ijk}>…}. A schedule is a set of quadruples <j_i, a_{ij}, r_k, tr_{ijk} >, where j_i is a job, a_{ij} is an activity associated with j_i, r_k is a resource, and tr_{ijk} is the *job time range* associated with the assignment of activity a_{ij} of j_i to resource r_k. A job time range specifies the earliest and latest start and end time and the duration of a_{ij}. We assume that no additional control information is required to interpret a schedule. In other words, we assume that the execution of a schedule only requires assigning resources to jobs and activities, in accordance with the given time ranges.

We can now define various criteria to check the validity of a schedule.

- A schedule, say S_i, is *correct*, if no pair <j_i, a_{ij}> appears in more than one quadruple in S_i.
- A schedule, say S_i, is *complete*, if for each activity a_{ij} in A, associated with job j_i, there exists a quadruple q in S_i, such that q = <j_i, a_{ij}, r_k, tr_{ijk} >.
- A schedule, say S_i, is *admissible*, if it does not violate any of the applicable constraints.
- A schedule, say S_i, is *feasible*, if it satisfies all the requirements imposed on a solution schedule.
- A solution schedule, say S_{opt}, is *optimal*, if there is no other solution schedule that has a lower cost than that of S_{opt}.

Our task ontology subscribes to a job-centred viewpoint [11], i.e., we take the point of view that scheduling is an assignment of jobs and associated activities to resources and time ranges. Clearly, it would be possible to get rid of the notion of job altogether and limit ourselves to modelling and assigning activities. However, the notion of job is so ubiquitous throughout the scheduling literature that we believe it is worth maintaining this form of redundancy in the model. Essentially, jobs provide a way of clustering the activities together: they are not important computationally, but they provide a useful abstraction to support knowledge acquisition.

Having completed our conceptual framework, we can now describe how this has been represented in our task ontology.

3 A SCHEDULING TASK ONTOLOGY

Our scheduling ontology contains about 60 definitions, and in addition it relies on two underlying ontologies, Base Ontology and Simple Time. Base Ontology provides the definitions for basic modelling concepts such as, tasks, relations, functions, roles, numbers, sets etc. Simple Time is based on Allen's [12] standard time relations and defines notions such as, time point, time interval, time range, etc. The scheduling ontology has been developed using the OCML modelling language [1], which provides support for executing the definitions in the ontology as well as export mechanisms to other representations, including Ontolingua. Both OCML and Ontolingua versions of the ontology are publicly available and can be found on URL: http://kmi.open.ac.uk/~enrico/scheduling.html. The OCML version of the ontology can also be browsed using the WebOnto environment at http://webonto.open.ac.uk. Given the obvious space limitations, here we only discuss the main building blocks of the ontology and the main modelling decisions taken while developing it. A reader interested in a more detailed analysis of the model is encouraged to look at the ontology in detail by accessing the above URLs. The reader should also note that we will focus the discussion on the definitions included in the scheduling ontology and we will take the underlying base and time ontologies for granted. Again, the reader interested in finding out more about these ontologies is encouraged to access the above URLs.

3.1 Representing a scheduling task

Our task modelling framework characterises a generic task in terms of input and output roles, precondition and a goal expression [1, 7]. The input and output roles for a generic scheduling task have already been given, so here we focus only on precondition and goal expression. The precondition imposed on the task specifies that jobs, resources and the schedule time range are required for a meaningful specification. If no solution criterion is provided, then a default one applies, so it is not necessary to provide one. The goal expression simply states that the solution criterion must hold for the output schedule. In Ontolingua, the goal expression is represented as follows (note that, for the sake of brevity, the definition below is a heavily-abridged version of the one in the ontology, showing only the specification of the goal expression):

```
(Define-frame SCHEDULING-TASK
:own-slots ((subclass-of goal-specification-task))
:template-slots ((Has-Goal-Expression
      (slot-value (kappa (?task ?schedule)
                         (holds
                         (role-value
                          ?task 'has-solution-criterion)
                         ?task
                         ?schedule)))
(slot-value-type binary-relation)
```

As shown above, the goal of a task is represented as a binary relation that takes as argument a task and a schedule and is satisfied if a solution criterion associated with the task holds for the given task and schedule. The default solution criterion, which ignores optimality considerations, is represented as a binary relation, defined over a schedule and a task, as follows:

```
(Define-relation DEFAULT-SOLUTION-CRITERION (?s ?t)
:iff-def (and (schedule-is-correct ?s)
              (schedule-is-complete ?s
                (role-value ?t 'has-jobs))
              (schedule-is-admissible ?s
                (role-value ?t 'has-constraints))
              (schedule-is-feasible ?s
                (role-value ?t 'has-requirements)))
:constraints (and (schedule ?s) (scheduling-task ?t))))
```

3.2 Jobs and activities

A job is an entity that involves a list of activities and can be assigned to the available resources within a specific time window for its execution. It is defined in terms of the following attributes:

Has-activity. The activities associated with a job that needs to be executed in order to accomplish the job. For example, if a job is a drilling one, then its activities can be a machine set-up, the actual drilling operation, the unloading of the job, etc.

Requires-resource. The specific set of resources that is suitable to carry out the job.

Requires-resource-type. In some cases we do not need to specify concrete resources for a job, but we simply want to constrain the type of resources which are needed to carry out the job - e.g., a type of drilling machine, a machinist etc.

Has-time-range. The time range assigned to each job specified in terms of earliest and latest start and end time windows.

Has-load. The designated *quantity* of resources that are required by each job for its execution. The default is 1, but some jobs may require more than one resource.

Has-due-date. The calendar date by which a job must be send.

Activities indicate the number of operations within each job that need to be performed in order to accomplish that particular job. The attributes of activities are basically the same as those for jobs, except that activities are not further refined into sub-activities.

3.3 Resource

A resource is a finite capacitated entity to which the jobs can be assigned for their execution. The class resource has the following attributes.

Handles-job. The specific jobs a resource can handle.

Handles-job-type. The type of jobs a resource can handle.

Has-availability. The time interval in which a resource is available. For instance, a transmitter may have to be switched off periodically for maintenance purposes.

Has-capacity. The capacity of a resource is modelled as an *integer* indicating the maximum number of jobs a resource can handle concurrently. The default is 1.

A *resource-capacity* axiom literally states that for a given resource with capacity $?n_i$ and schedule ?s, there should not exists a job, $?j_i$, assigned to $?r_i$ in time range ?jtr, such that, if we construct the set of all jobs assigned to $?r_i$, which overlap with $?j_i$, and we can add $?j_i$ to this set, we end up with a set of cardinality higher than ?ni. This axiom is expressed in the ontology as follows:

```
(Def-axiom RESOURCE-CAPACITY
(forall (?ri ?s)
 (=> (and (resource ?ri) (has-capacity ?ri ?ni))
     (not (exists ?j (and (element-of
                           (?j ?a ?ri ?jtr) ?s)
     (= ?all (setofall ?j2 (and (element-of
                           (?j2 ?a2 ?ri ?jtr2) ?s)
          (job-time-ranges-overlap (?jtr ?jtr2)
           (not (= ?j2 ?j))))
 (> (set-cardinality (union (setof ?j) ?all2)) ?ni)))))))
```

3.4 Constraints and requirements

In our task ontology we clearly distinguish between constraints and requirements, even if some approaches blur such distinction [13]. A constraint defines a property, which must not be violated by a solution, while requirements specify properties, which a solution has to satisfy. In general, not all defined constraints are necessarily applicable to a schedule, so a solution may be admissible even if some constraints are not satisfied. They simply may not be relevant. In contrast, a feasible solution must satisfy all requirements defined for the problem. They specify the desired properties over a solution.

Many approaches in the literature distinguish between *soft* and *hard* constraints. While hard constraints must not be violated, soft constraints can be relaxed if necessary to reach a solution. We find this distinction ambiguous, as it is not clear what does it mean to have constraints which can be violated if necessary. In our model constraints define *proscriptive* properties, while requirements describe *prescriptive* ones. Soft constraints are neither proscriptive nor prescriptive. Normally what happens is that soft constraints are used as a quality measure for the different solution schedules. A solution schedule that satisfies a maximum number of soft constraints is treated as a better schedule than other competing solutions. Hence, soft constraints do not concur to define the space of admissible solutions, but they instead can be used to rank solutions. For this reason we prefer to use the notion of *preference*, which we believe more clearly express the ontological role of these constructs.

3.5 Cost, cost function and preferences

Most scheduling tasks can be seen as a combinatorial optimisation problem [3], i.e., normally we do not simply want to find an admissible and feasible solution, but we want to optimise over some evaluation function, e.g., we may want to minimise cost or maximise resource utilisation. Our ontology provides two constructs, which allow us to capture the knowledge, needed to rank solutions: *preferences* and *cost function*. Preferences allow us to describe task knowledge which can be used to assess whether a solution can be regarded as better than another. For instance, in some cases we may prefer to use one resource rather than another, even when both are suitable for a particular job. The role of preferences allow us to capture important task knowledge, which is clearly of a different nature from requirements and constraints and which is often sloppily characterised as "soft constraints" in the literature. Once acquired the relevant preferences we use the notion of cost function to develop an optimisation criterion for a given scheduling problem. In general, this is a non-trivial effort, as preferences tend to be heterogeneous and they have different cost - e.g., it may be acceptable to violate any number of 'cheap' preferences, but not to violate even one 'expensive' preference. Therefore it is important to emphasise that cost functions may not necessarily be numeric and often, some *non-Archimedean criterion* may be applied [1].

Our ontology models preferences as binary relations, which define a partial order over schedules. A cost function is defined (not surprisingly) as a mapping from schedules to costs, where a cost can be modelled either as a real number or as an n-dimensional vector.

We have pointed out that the role of a cost function is to define a single optimisation criterion, which is both consistent with and subsumes the various criteria expressed by the various preferences. This requirement is formalised by means of two axioms:

```
(Def-axiom COST-PREFERENCE-CONSISTENCY
(forall (?task ?s1 ?s2)
   (=> (and (scheduling-task ?task)
            (has-preferences ?task ?prs)
            (has-cost-function ?task ?cf)
            (has-cost-order-relation ?cf ?rel)
            (cheaper-schedule ?rel ?s1 ?s2))
       (not (exists ?pr (member ?pr ?prs)
            (holds ?pr ?s2 ?s1))))))

(Def-axiom COST-SUBSUMES-PREFERENCES
 (forall (?task ?s1 ?s2)
   (=> (and (scheduling-task ?task)
            (has-preferences ?task ?prs)
            (member ?pr ?prs)
            (holds ?pr ?s1 ?s2)
            (has-cost-function ?task ?cf)
            (has-cost-order-relation ?cf ?rel))
       (cheaper-schedule ?rel ?s1 ?s2))))
```

The first axiom enforces the requirement that the cost function should not be contradicted by any partial order expressed by any relevant preference. The second axiom states that any solution ranking introduced by any preference associated with a task should also be enforced by the cost function. The above definitions make use of the association between a cost function and a *cost-order relation* that expresses the partial order defined by the cost function.

4 VALIDATION STUDY

In order to validate our claim about the generality of the task ontology as well as to verify its expressiveness we have tested it on two different scheduling applications: one (provided by a colleague) in the domain of satellite scheduling, and a real-life resource allocation problem in the context of a large research project.

4.1 The satellite scheduling problem

The main reason for choosing the satellite-scheduling problem is its dynamic nature and the varying degrees of constraints and requirements.

The satellite-scheduling problem can be characterised in terms of an assignment of a number of satellites to available antennas in order to ensure communication between the satellites and antennas at

different times during a 24-hour period. The satellites are treated as jobs and the antennas are treated as limited supply resources.

The problem specifies 5 satellites, nimbus-1, nimbus-2, chandra-1, meteorological-1 and meteorological-2. In order to communicate with the satellites we have 3 antennas available, low-range-antenna, wide-range-antenna and meteorological-antenna. The satellite-scheduling problem is formalised by eliciting the following constraints and requirements from the problem description:

- *Antenna visibility Constraints*: this set of constraints states that each antenna has a limited visibility period for communicating with the assigned satellites. All the communication activities within each satellite must be completed within the visibility period of the antennas.
- *Number of communications:* this requirement specifies that every satellite must have at least 4 communication slots per day.
- *Communication duration:* this requirement states that each communication slot must have 15 minutes duration.
- *Communication gap:* this requirement says that the gap between any two communication slots for a given satellite should not be greater than 5 hours.

No optimality criteria were defined for this problem, so we used the default solution criterion requiring a schedule to be correct, complete, admissible and feasible (cf. section 2).

We did not encounter any particular problem in applying our task ontology to this problem. Each daily communication slot was modelled as an activity, associated with the relevant job (i.e., satellite). Ten constraint instances were created in order to specify the restrictions on the availability of specific antennas to specific satellites. One such constraint is shown below that states that no communication activity involving satellite chandra-1 and the wide-range antenna can take place during the time range 'no-chandra-1-to-wide-range-antenna-visibility', i.e., between 13:01 and 23:59.

```
(Def-Instance    CHANDRA-1-VISIBILITY-TO-WIDE-RANGE-ANTENNA
satellite-antenna-visibility-constraint
((Has-Expression (kappa (?schedule)
  (forall (?a ?jtr)
   (=>
     (member (chandra-1 ?a wide-range-antenna ?jtr)
               ?schedule)
     (not (time-ranges-intersect ?jtr
     no-chandra-1-to-wide-range-antenna-visibility))))))))
```

In sum, the ontology appeared to provide the modelling support required to represent this problem. Only a few additional relations to reason about time were added as a result of this validation exercise.

4.2 CIPHER: a resource allocation problem

CIPHER is a real-life collaborative project among 6 academic and industrial partners. To maintain anonymity we will denote them as co-ordinator-1, contractor-2, contractor-3, contractor-4, contractor-5 and contractor-6. The project comprises 12 work packages to be delivered by a certain deadline. Each work-package includes a number of tasks, which need to be carried out in order to achieve the objectives of the work-package. Partners have naturally only a limited number of persons available, who can carry out the work prescribed by the various work-packages. The goal of the scheduling task is to allocate the available persons to the various work-packages and activities constructing a complete project schedule.

In accordance with the task ontology the project staffs are treated as resources, and work packages as jobs. The tasks within each work package are treated as activities. In all there were 32 activities involved within 12 work packages. The schedule is constructed for a period of 30 months. Each resource is assumed to have capacity 1, i.e., no researcher is assumed to be able to work on two tasks at the same time throughout the project duration. The following preferences were elicited:

- *End time compliance preference:* this preference says that each work package is preferred to end on its specified end time and not prior to it.
- *Coverage preference:* this preference states that 'idle time' ought to be minimised and ideally every month in every work package must to be covered by at least one resource.
- *Competence matching preference.* Some people are better at certain tasks than others, so any schedule should ensure optimal competence matching.

This example showed the value of our ontology with respect to knowledge acquisition and modelling. In particular we took advantage of the distinction between constraints, requirements and preferences provided by the task ontology to correctly characterise the problem specification. The elicited preferences specify alternative partial orders on the solution space and we adopted a non-Archimedean cost function to reconcile these conflicting preferences to produce a single optimisation criterion. Conceptually this was achieved by ranking the three preferences: competence matching was given priority over 'idle time' minimisation, and the latter was given priority over end time compliance. In addition, the formal representation of the model made it possible for us to quickly implement prototypes and test the behaviour of problem solvers on alternative problem specifications.

5 COMPARISON WITH RELATED WORK

In this section we compare our work with three other existing task ontologies: the job-assignment task ontology [8], the MULTIS task ontology [9] and the OZONE ontology [10].

The job-assignment task ontology was developed in the context of the CAKE project by Hama. *et al*. The job assignment task is defined as “assigning all given jobs to the available resources within the time range, while satisfying various constraints”. The most important distinction between this ontology and ours is the *level of granularity*. In their framework the level of detail for describing the important concepts is very coarse-grained. For instance, let's consider the definition of a job from the job assignment task ontology.

```
(Define-class job (?source)
 :def (source ?job))
```

If we compare this representation of a job with ours it is clear that while the job assignment identifies the building blocks of a scheduling task, it does not provide the level of support necessary to model scheduling tasks in detail. Another crucial difference is that our task ontology provides a richer framework for representing task knowledge for scheduling problems, by distinguishing between constraints, requirements, preferences and cost function. As discussed in section 4, these distinctions are important to allow us to represent a problem description correctly. It is not clear how one would model the CIPHER application using the job-assignment ontology. In addition our task ontology also assumes that no fixed notion of 'schedule solution' exists, but simply provides a number of possible criteria, as well as the modelling support needed to characterise alternative notions. In contrast with our task ontology, the job-assignment ontology framework simply validates a schedule against the total mapping of jobs to resources while satisfying constraints, ignoring feasibility and optimality aspects.

The MULTIS task ontology is constructed through a task analysis interview for the general classes of scheduling tasks. There are a few important concepts that are missing in MULTIS. There is no explicit representation of a *‘resource capacity’* criterion, which is crucial in order to avoid the overlapping of jobs and is one of the focal issues, as far as scheduling is concerned [4, 11]. In MULTIS the definitions of important concepts do not provide any scope to express the problem specific knowledge by filling the slots of the definitions. For instance, a class `resource` is defined, but it fails to provide other attributes of resources such as, a capacity, an avail

ability period etc. If we consider the antenna visibility constraint in the satellite application (cf. section 4.1), it is difficult to represent a non-availability period of an antenna by using MULTIS as the notion of availability period is missing. In addition MULTIS also ignores the distinction between requirements and constraints and does not handle cost-related issues in association with preferences. Again, it is not clear how could one model a problem such as CIPHER, where cost-related issues play a crucial role.

The OZONE ontology also provides a model of scheduling tasks, which are defined in terms of five base concepts: *demand*, *activity*, *resource*, *product* and *constraint*. In terms of our framework the concept *product* does not directly contribute to specify the scheduling problem but it can be seen as an external environmental factor. We are mainly interested in investigating the core issues involved in a scheduling task. The concept demand and activity in OZONE have attributes such as time range and assigned-resource, but they do not talk explicitly about the *load* factor indicating the number of resources that are required by each demand or activity. In the CIPHER application the load distribution on each activity is an important aspect of the problem specification. Like the other two task ontologies we have examined in this section, the OZONE ontology does not explicitly talk about the cost and preference issues. However, their use of soft constraints resembles the role of preferences in our ontology, although it seems to us that such an approach is rather opaque. Moreover, the lack of a cost function means that no mechanism is provided to integrate different preferences in order to discuss their relative importance and this also makes it difficult to assess the impact of preference-specific decisions on the cost of a schedule. In addition, no notion of requirement is included either.

The OZONE framework is built to support a constraint-based scheduling 'shell'. Therefore most of the definitions are geared to support the constraint-based problem solving approach. In contrast with this approach we do not make any assumptions about the type of problem-solving approaches that can be used to solve the problem. The disadvantage of subscribing to a particular problem solving approach is that important conceptual distinctions are not considered, if they are not directly supported by the problem solving environment. We argue that this is bad, as it blurs the distinction between analysis and design in system development. Consistently with structured approaches to knowledge engineering, such as the KADS methodology [14], the analysis phase focuses on identifying the knowledge and the problem solving strategies relevant to a knowledge-intensive problem. During this phase, the goal is to identify all relevant conceptual distinctions and capture the relevant knowledge. It may well be that certain distinctions, for instance requirements and constraints, may end up being implemented using the same computational structures in the end system. Nevertheless, if a biased conceptual model is elicited, or if the conceptual model is only a partial characterisation of the problem, then the 'wrong' system will inevitably be built.

6 CONCLUSION

In this paper we have described a task ontology which characterises scheduling tasks independently of particular domains, applications, or problem solving approaches. This work can be situated in the framework of ongoing work on KBS development by reuse, which aims to put KBS technology on firm ontological and engineering foundations. We see this work as being important for both analytical and engineering reasons. It helps us to understand the ontological nature of important classes of KB applications, and at the same time it enables us to develop resources, which can be used to acquire knowledge about a specific problem and build a detailed specification of it. The task ontology described here has been validated on two real-world applications. As argued earlier in the paper, our ontology includes and formally characterises a number of important conceptual distinctions that are missing from existing approaches. Because our task ontology does not subscribe to any specific problem solving approach it provides a sound ontological foundation for alternative problem solvers and can be used to support task modelling independently of any target shell or computational method.

An anonymous reviewer pointed out that our approach ignores the distinctions between different classes of scheduling problems, which have been identified in the literature. We do not see this as a limitation of our approach, as our main goal here is to provide a generic reference model for *all* classes of scheduling problems. And moreover the two lines of research are not inconsistent, as our reference model can of course be specialised for different classes of applications, if appropriate.

Our current work is using the task ontology as a starting point for building a library of *problem solving methods* for scheduling. By building on solid ontological foundations, we expect not only to provide a useful set of resources for developing scheduling applications, but we also expect to be able to develop insights on the space of scheduling behaviours, along the lines of our earlier work [1, 15] on parametric design problem solving.

ACKNOWLEDGEMENTS

We would like to thank anonymous referees of ECAI 2002 for their valuable comments.

REFERENCES

[1] E. Motta, Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design, IOS Press, Amsterdam, (1999).
[2] M. S. Fox, ISIS: A Retrospective, in M. Zweben and M. S. Fox (edited), Intelligent Scheduling, 3-28, Morgan Kaufmann, (1994).
[3] J. Saucer, Knowledge-Based Systems Techniques and Applications in Scheduling, Knowledge-Based Systems Techniques and Applications, San Diego, Academic Press, (1997).
[4] M. Zweben and M. S. Fox (eds.), Intelligent Scheduling, Morgan Kaufmann, Palo Alto, (1994).
[5] T. R. Gruber, A Translation Approach to Portable Ontology Specification, Knowledge Acquisition, **5** (2), 199-221, (1993).
[6] E. Motta, D. Fensel, M. Gaspari, and R. Benjamins, Specifications of Knowledge Components for Reuse, in the proceedings of the 11th Intl. Conf. on Software Engineering and Knowledge Engineering (SEKE '99), Kaiserslautern, Germany, 36-43, (1999).
[7] D. Fensel and E. Motta, Structured Development of Problem Solving Methods, IEEE Transactions on Knowledge and Data Engineering, **13** (6), 913-932, (2001).
[8] T. Hama, M. Hori and Y. Nakamura, Modelling job assignment task based on task ontology, in the proceedings of the 2nd Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop, Kobe and Hatoyama, 199-213, (1992).
[9] R. Mizoguchi, Y. Tijerino and M. Ikeda, Task Analysis Interview Based on Task Ontology, Expert Systems with Applications, **9** (1), 15-25, (1995).
[10] S. F. Smith and M. A. Becker, An Ontology for Constructing Scheduling Systems, in the proceedings of AAAI-97, Spring Symposium on Ontological Engineering, (1997).
[11] S. F. Smith, Reactive Scheduling Systems, Intelligent Scheduling Systems, D. E. Brown and W. T. Scherer (eds.), Kluwer Press, (1995).
[12] J. Allens, Towards a general theory of action and time, Artificial Intelligence, **23** (2), 123-154, (1984).
[13] J. Breuker and W. Van de Velde (edited), CommonKADS Library for Expertise Modelling, IOS Press, Amsterdam, The Netherlands, (1994).
[14] A. Th. Schreiber, B. J. Wielinga, and J. A. Breuker (eds.), KADS: A Principled Approach to Knowledge-Based System Development, vol. 11 of Knowledge-Based Systems Book Series. Academic Press, London, ISBN 0-12-629040-7, (1993).
[15] E. Motta and Z. Zdrahal, A library of problem-solving components based on the integration of the search paradigm with task and method ontologies. Intl. Journal of Human-Computer Studies, **49** (4), 437-470, (1998).