CaBMA: A Case-Based Reasoning System for Capturing, Refining, and Reusing Project Plans

Ke Xu & Héctor Muñoz-Avila

Department of Computer Science and Engineering
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015, USA
{kex2, munoz}@cse.lehigh.edu

Keywords: Case-Based Reasoning, Project Planning

Abstract

In this paper, we present CaBMA, a prototype of a knowledge-based system designed to assist with project planning tasks using case-based reasoning. CaBMA introduces a novel approach to project planning in that, for the first time, a knowledge layer is added on top of traditional project management software. Project management software provides editing and bookkeeping capabilities. CaBMA enhances these capabilities by automatically capturing project plans in the form of cases, refining these cases over time to avoid potential inconsistency between them, reusing these cases to generate plans for new projects, and indicating possible repairs for project plans when they derive away from existing knowledge. We will give an overview of the system, provide a detailed explanation on each component, and present an empirical study based on synthetic data.

1 Introduction

Project management is a business process for successfully delivering one-of-a kind products and services under real-world time and resource constraints (PMI, 1999). Project management software helps users manage large projects by providing edits to project content and control to resource allocation. Commercial software for project management includes *Microsoft Project*™ (Microsoft), *SureTrak*™ (Primavera Systems Inc.), and *Autoplan*™ (Digital Tools Inc.).

A crucial activity of a project is creating a work breakdown structure (WBS), which decomposes the project's tasks into manageable work units. This process requires significant domain knowledge and experience. For example, a software project manager who needs to deliver a real-time chemical process control system must employ significant knowledge of real-time software development, combined with chemical process control experience. The complex interdependencies between tasks and domain knowledge make creating the WBS a difficult work. A common drawback of current commercial project management software is that they do not assist users in creating new WBS.

We present CaBMA (for: Case-Based Project Management Assistant), a prototype for a knowledge-based system that implements case-based reasoning on top of project management software (i.e., *Microsoft Project*™) to assist project planners in creating new WBSs. CaBMA's crucial contribution is that it demonstrates for the first time how to add a knowledge layer top of commercial project management software, going beyond the editing and bookkeeping capabilities that this software is traditionally limited to. CaBMA extends current project management software by adding a knowledge layer with following functionalities[1]:

- Capturing reusable cases from previous project plans

[1] This paper expands and extends the work presented in (Xu & Munoz-Avila, 2004).

- Refining cases to avoid inconsistencies resulting from contradictory cases captured over time
- Reusing cases to complete partial project plans and generate new plans from the scratch
- Maintaining consistency of pieces of a project plan that are obtained by case reuse

We conducted an experiment with synthetic data to measure the correctness of project plans generated by CaBMA. The result indicated that CaBMA has a high degree of precision, measured as the percentage of correct plans obtained albeit with a decrease in recall, measured as the ratio of solutions generated over the number of solvable problems given.

The next section illustrates the general idea of project management, using *Microsoft Project*™ as an example. Then in Section 3, we explain the integrated architecture and each component of CaBMA. The empirical results are discussed in Section 4. The related work is in Section 5, and the summary is in Section 6.

2 Project Management and Motivation

Project management has a large variety of applications, including research proposal development, public events organization, and civil construction management. According to the project management institute, which is a pioneering organization in the field, project management typically consists of two parts (1) project planning and (2) project execution sub-processes. Project planning can comprise the following knowledge/work activities and decisions (PMI 1999):

1. *Creating a work breakdown structure (WBS):* The (human) planner identifies and establishes a hierarchically organized collection of tasks that enables the delivery of required products and services.
2. *Identifying/incorporating task dependencies*: The planner identifies task dependencies and schedules tasks accordingly.
3. *Estimating task and project durations:* The planner estimates the time required to accomplish each task, and uses the task dependencies in the WBS to estimate overall project duration.
4. *Identifying, estimating, and allocating resources:* The planner identifies the types of resources required by each task, allocates the resources to each task in the WBS, and estimates the rates of resource consumption.

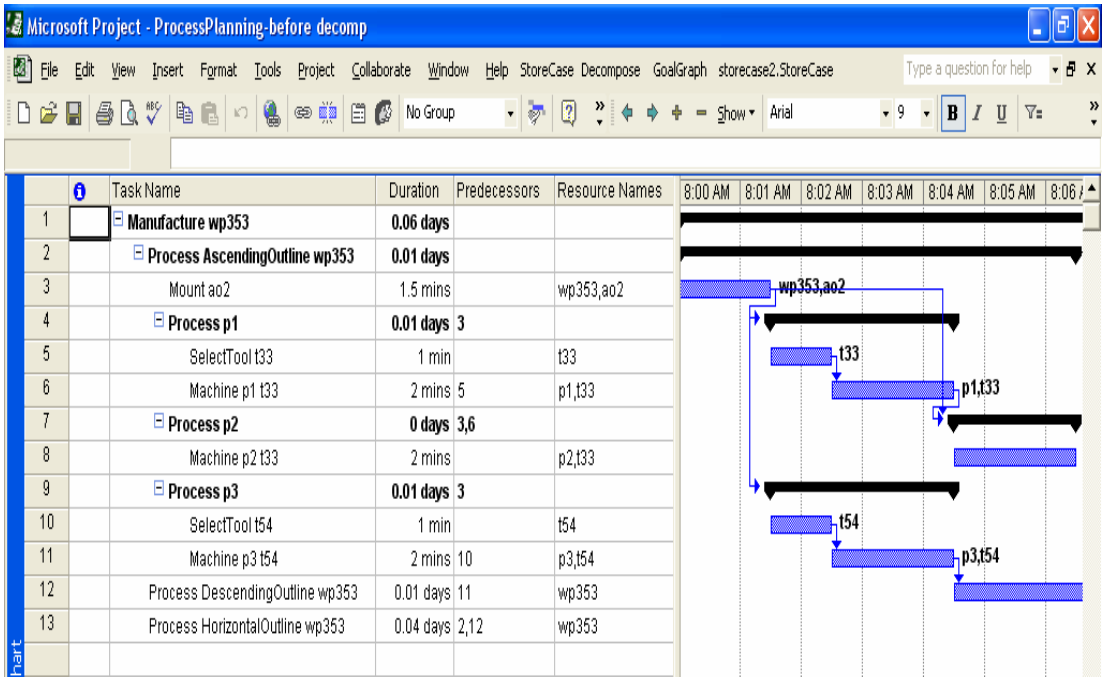


Figure 1: A WBS in Microsoft Project™

5. *Estimating overall project costs/budget:* The planner estimates the cost of resources consumed, compiles an overall project cost, and often derives a scheduled cash flow.
6. *Estimating uncertainties and risks:* The planner estimates uncertainties and risks associated with tasks, resources, and schedules.

Project execution can include the following activities:

1. acquiring and organizing the resources,
2. performing the task,
3. monitoring the task status and comparing it with expected execution status to identify deviations, and
4. re-planning or adjusting the plan as needed to meet the project objectives.

Software packages for project management are commercially available, including *Microsoft Project*™ (Microsoft), SureTrak™ (Primavera Systems Inc.), and Autoplan™ (Digital Tools Inc.). They help a planner with manually recording his/her plan with the activities above involved. These packages do provide support to ensure that resources are not over-allocated (activity (4) from the project planning activities), to estimate costs (activity (5)) by adding up costs for tasks, and to estimate global times (activity (3)) by adding up times from leaf tasks. However, they do not assist a planner in the complicated and knowledge intensive part of project planning: creating a WBS and identifying dependencies between the tasks. Our primary focus on CaBMA is to generate the WBSs and task dependencies automatically, using case-based reasoning. These WBSs can be edited using the standard functionalities provided by the software packages afterwards. Thus, by helping the user perform activities (1) and (2), CaBMA complements existing commercial off-the-shelf (COTS) project management systems.

3. Hierarchical Plan and Work-Breakdown Structure Representations

CABMA follows the principles of Hierarchical Task Network (HTN). HTN planning accomplishes complex tasks by decomposing them into simpler subtasks (Erol *et al*. 1994). Planning continues by decomposing the simpler tasks recursively until tasks representing concrete actions are generated. These actions form a plan achieving the high-level tasks. Decomposable tasks are called ***compound tasks***, while non-decomposable tasks (i.e., tasks representing concrete actions) are called ***primitive tasks***. We now describe the HTNs, and later in this section we also describe the Work breakdown structures (WBSs) and a mapping from HTNs to WBSs.

A hierarchical task network can be defined recursively as follows[2]:

- An expression of the form $(t^h,\{t_1,\ldots,t_m\},<)$ is a HTN, where t^h, $t_1,\ldots,t_m$ with $m \geq 0$, are tasks (a task is represented as a logical atom), and < is a binary relation expressing temporal constraints between $t_1,\ldots,t_m$. A temporal constraint between task t and t' is defined as follows: if $t < t'$, then t' cannot start until t finishes. The task network indicates that t^h is decomposed into $t_1,\ldots,t_m$.
- An expression of the form $(t^h,\{T_1,\ldots,T_m\},<)$ is a HTN, where t^h is a task and $T_1,\ldots,T_m$ are HTNs, and < is a binary relation between tasks in $T_1,\ldots,T_m$. The task network indicates that t^h is decomposed into $T_1,\ldots,T_m$.

Hierarchical planning involves recursive decomposition of tasks in a HTN into their respective sub-tasks. The HTN that is given initially to decompose has the form: form $(\_,\{t_1,\ldots,t_m\},<)$, which indicate that $t_1,\ldots,t_m$ must be decomposed and that the ordering relation < must be preserved.

A WBS is a hierarchically organized set of ***elements*** that need to be performed to deliver the required goods and/or services. Elements in a WBS can be of two kinds: tasks and activities. An ***activity*** is a terminal node (i.e., additional elements cannot be attached). ***Tasks*** can contain activities and other tasks (i.e., subtasks). Elements in the WBS can be ordered using the following types of precedent constraints:

- *end-start*: An element cannot start before another one finishes.
- *start-start*: An element cannot start before another one starts.
- *end-end*: An element cannot finish before another one has finished.
- *concurrent-start:* Two elements must start at the same time.

[2] We omit binding constraints and interval preserving constraints as they play no role in the mapping to WBSs.

- *concurrent-end*: Two elements must finish at the same time.

The ordering constraints in HTNs are of type end-start. However, it is easy to use end-start constraints to represent start-start and end-end constraints. For example if we want to represent that a task t cannot start before a task t' starts (start-start relation), we introduce the task network: (_,{(t,{t_1,t_2},{ t_1< t_2}) ,(t', {t_1',t_2'},{ t_1'< t_2'})},{ t_1' < t_1}). The pairs of tasks t_1,t_2 and t_1',t_2' are added to represent the start and end of that tasks t and t'. The relation t_1' < t_1 formally indicates the start-start relation. Concurrent-start and concurrent-end constraints cannot be represented in HTNs as originally proposed. However, other approaches for hierarchical planning do contain ways to express task concurrency (Myers & Wilkins, 1999). In the remainder of this paper, we limit our discussion to end-start constraints. We are currently extending our representation to include these kinds of constraints. Another difference between HTN and WBS representations is that WBS contains additional information such as durations of tasks (Activity 3 of Project planning process in Section 2), and allocated resources (Activity 4). As stated in Section 2, CaBMA supports Activities 1 (creation of WBS) and 2 (identifying task dependencies).

The mapping of WBS and hierarchical plans is straightforward: WBS tasks are the same as compound tasks in a HTN, WBS activities are primitive tasks, and precedence constraints of type 1 (end-start) are the ordering constraints. Table 1 summarizes this mapping.

Table 1. Relation between WBSs and hierarchical plans.

WBS	Hierarchical Plans
Task	Compound task
Activity	Primitive task
End-start precedent constraint	Ordering constraint
Start-start precedent constraint	Ordering constraint
End-end precedent constraint	Ordering constraint

Figure 1 shows a work breakdown structure of a manufacturing project represented in *Microsoft Project*™. The compound tasks, for example, "Manufacturing Workpiece wp353", are decomposed into simpler subtasks. The subtasks, such as "Processing Ascending Outline ao2 on Workpiece wp353", are eventually decomposed into activities, which are basic executable work units (i.e., primitive tasks). For instance, "Machining Processing Area p3 with Tool t54" is an activity. In general, there are three kinds of relations in a WBS:

- task–subtask/activity relations (e.g., "Mounting Ascending Outline ao2" is a subtask of "Processing Ascending Outline ao2 on Workpiece wp353")
- task/activity–resource assignment relations (e.g., "Machining Processing Area p1 with Tool t33" requires processing area "p1" and tool "t33" as resources)
- task/activity–task/activity ordering relations (e.g., "Processing Ascending Outline ao2" must be executed prior to "Processing Descending Outline do2")

4 Integrated Architecture of CaBMA

We present CaBMA (for: Case-Based Project Management Assistant), a knowledge-based system that implements knowledge-based project planning (KBPP). Knowledge-based project planning advocates assisting project managers in developing a WBS by using case-based hierarchical task network (HTN) planning techniques (Muñoz-Avila et al, 2002). Our approach follows from the observation that a WBS has a one-to-one correspondence with an HTN, which will detail later on. CaBMA is built on top of *Microsoft Project*™, and serves as a knowledge layer to construct project plans automatically. The goal of CaBMA is to assist the planner during the generation of a WBS. Microsoft Project is a popular commercial project management tool. It helps users to store WBS, allocate resources and follow deadlines. It incorporates a Visual Basic editor that allows users to extend its functionality. The WBS is edited in a spreadsheet like window called the Task View. All the resources are edited in the Resource Sheet. The user can allocate resources to each task separately.

Figure 2 depicts the integrated architecture of CaBMA. There are four major components in CaBMA, shown as the green boxes in the figure. First, the user creates a project plan in *Microsoft Project*™. This means that the user adds tasks, resources, and orderings in a WBS. The tasks provided by the user need to

be recognized by CaBMA. The Task Dictionary Component (TDC) allows the user to choose alternatives from existing tasks or define new tasks. The user then selects to store parts of or the entire WBS as cases using the Gen+ component. The Gen+ component stores the cases in the case base and may refine cases to avoid conflicts if necessary. At any time, the user can select a task in a WBS and ask CaBMA to generate a plan (decomposition). SHOP/CCBR is the component that reuses cases to generate the decomposition. As the user changes the current project plan, some parts of the WBS may become inconsistent relative to the reused cases. The Goal Graph Component (GGC) keeps track of all edits, including those performed by the user and those made by case reuse. GGC alerts the user of potential inconsistencies in an unobtrusive manner.

Before explaining each component in detail, we briefly discuss some implementation details. In our first prototype, Gen+ and SHOP/CCBR were implemented in Java as separate tools from MS Project. We used COM (Component Object Model) as the software design model to facilitate the communication between Microsoft Project and these two components. This resulted in very slow running time. For our second prototype we re-implemented directly these two components in Microsoft Project's native programming environment, which uses Visual Basic. This resulted in much faster execution times for the components. The TDC component was build form the outset in Microsoft Project.

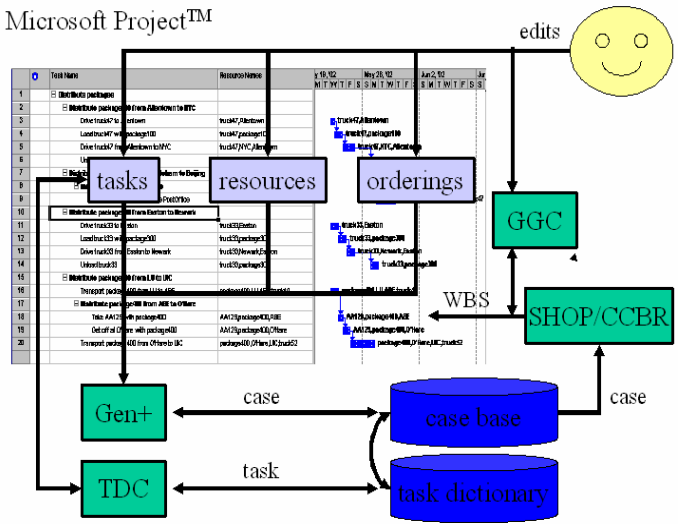


Figure 2: The integrated architecture of CaBMA. The green color refers to the four components, the blue to the two knowledge bases, and the purple refers to the input data

3.1 Task Dictionary Component (TDC)

Current project management tools, such as *Microsoft Project™*, do not constraint the syntax of the task descriptions. This means different users may represent the same task in different ways. For example, one user may input a task as "deliver file X from office Y to office Z", while another user may put the same task as "get file X at office Y, and then deliver it to office Z". Since CaBMA applies case-based reasoning techniques, which requires task matching to compute similarity between tasks and cases, arbitrary task description will make it difficult to identify or match the tasks. CaBMA sidesteps this problem by using a task dictionary component (TDC) to maintain the list of all tasks that can be decomposed by existing cases. When the user inputs a task and decides to use CaBMA to decompose the task, TDC will list all tasks in the record that are syntactically similar to the input task. CaBMA then reuses the case associated with the selected task to generate a decomposition automatically. The user can also define new tasks that are not in the dictionary.

Tasks in the dictionary are kept as generalized tasks. When a user adds a new task, TDC prompts the user to identify the resources in the task. The generalized task t^G of a task t is obtained by replacing each resource ψ in t with a unique variable $?\psi$, if the type of the resource is known. For example, in Figure 1, if the input task is *Machine p1 t33*, the generalized task will be *Machine ?p1 ?t33*, assuming that the types of *p1 and t33* are known. When the tasks in the dictionary are displayed upon user selection, they are shown

with the resources highlighted. The user needs to specify how the resources should be filled when selecting a task.

3.2 Gen+ Component

As with any knowledge-based system, a crucial issue is how to fill the knowledge base. For CaBMA, the issue is how to populate the case base. The Gen+ component of CaBMA is designed for capturing cases from project plans, and refining existing cases when new cases are added. CaBMA extracts cases automatically from previous project management episodes. By doing so, the knowledge acquisition effort becomes transparent to the user. We now explain the three steps followed by the Gen+ component.

Step 1: Case Acquisition

In a project management process, a work breakdown structure (WBS) is a hierarchically organized set of elements that need to be performed to deliver the required products and services (e.g., creating a marketing strategy). Elements in a WBS can be of two kinds: tasks and activities. Tasks can contain activities and other tasks. Activities are executable work units (e.g., posting an ad in the local newspaper). Elements in the WBS can be ordered using precedent constraints.

The mapping from a WBS to a hierarchical task network (HTN) is straightforward: the WBS tasks map the HTN compound tasks. The WBS activities map the HTN primitive tasks. Finally, the WBS precedence constraints map the HTN ordering constraints. This mapping means that the AI techniques used for hierarchical plan generation can be used for WBS generation. Therefore, WBSs from previous project plans are captured as cases in an HTN representation, and these cases are reused to generate WBSs for new project plans afterwards.

A WBS can be seen as a collection of one-level decompositions. Each one-level decomposition indicates how a single task is divided into (sub)tasks and activities. In our approach, each one-level decomposition is stored as a single case. An alternative could be storing cases that contain several decomposition levels or even a complete WBS. There are trade-offs that we consider when choosing between these alternatives:

- Cases containing several decomposition levels provide a better picture of the overall project plan. Their main drawback is that the case reusability is limited. As more levels are added, more constraints should be considered to assess the applicability of a case in new situations.
- Cases containing one-level decomposition give no overall picture of a project plan. Their main advantage is that they can be reused in several situations, and cases from different WBSs can be combined. For example, one can decompose a task with a case captured from one WBS and decompose one of the subtasks with a case captured from another WBS.

We decided to store only one-level decompositions in cases to maximize their reusability. Nevertheless, more than one case can be composed to represent multiple decomposition levels. Using an HTN representation, a case is defined as a 5-tuple (h, Q, ST, <, ID), where:

- h is the decomposed task
- Q are the required conditions to apply the case
- ST are the subtasks decomposing h
- < is the set of ordering relations between the subtasks
- ID is the index of a case, which indicates the WBS from which the case is captured. Cases obtained from the same WBS have the same ID, and will be more likely considered together when they are reused during the task decomposition process.

Step 2: Case Generalization

A possible case applicability criterion for cases could require the current task being decomposed to be identical to the head of the case. This implies that if a planning problem contains n compound tasks, the average number of arguments in each task is m, and the average number of possible instantiations for each argument is i, then the number of cases required to generate a plan will be $n*i^m$. This is only the minimum number since it is desirable to have alternative cases for some tasks.

To reduce the number of cases required during planning, there are two alternatives. The first alternative is to relax the case applicability criterion by defining similarity metrics between non-identical ground tasks. Similarity metrics that use taxonomical representations for cases have been proposed (e.g., (Bergmann and Stahl, 1998)). This alternative also requires creating a case reuse mechanism for transforming the ground

subtasks of the case into other ground tasks. This alternative is typical of case-based planning systems such as CHEF (e.g., (Hammond 1986)) that rely on cases as the main source of domain knowledge. Such systems perform an implicit generalization when computing similarities between non-identical ground tasks. The second alternative is to generalize cases, use a task matching mechanism during case retrieval, and use HTN task decomposition for case reuse. These two alternatives are related in that they both have to deal with the issue of the correctness of any plan found, because cases are generalized (explicitly or implicitly) and reusing them may yield incorrect plans. In our approach, we selected the second alternative because it avoids the knowledge engineering effort of obtaining domain-specific adaptation knowledge.

To generalize a concrete case, CaBMA replaces each parameter in the case with a unique variable ?ψ, if the type of the parameter is known. Otherwise, the parameter is not replaced and is kept as a constant. In addition, we add the following constraints as conditions to the case:

- For two different variables ?x and ?y of the same type, we add the constraint: *different ?x ?y*
- For each constant c and each variable ?x, we add the constraint: *different ?x c*

As an example, Table 2 shows a concrete case (taken from the decomposition of manufacturing workpiece *wp353* in Figure 1) and the corresponding generalized case.

Table 2. A concrete project plan as a case and the corresponding generalized case

Concrete Case	Generalized Case
Task: Manufacture wp353 **Condition**: WorkPiece wp353 AscendingOutline ao2 DescendingOutline do2 HorizontalOutline ho2 **Subtask**: Process ao2 wp353 Process do2 wp353 Process ho2 wp353	**Task**: Manufacture ?wp353 **Condition**: WorkPiece ?wp353 AscendingOutline ?ao2 DescendingOutline ?do2 HorizontalOutline ?ho2 **Subtask**: Process ?ao2 ?wp353 Process ?do2 ?wp353 Process ?ho2 ?wp353

By capturing generalized cases, Gen+ increases the coverage, which is defined as the number of problems that can be solved using a case base (Smyth & Keane, 1995). Generalized cases may be redundant to the case base, but this can be easily solved by ensuring that a new case will not be added if it matches an existing case. (A generalized case C' matches a generalized case C if there is a substitution θ such that C'θ = C.)

Table 3: Two generalized cases requiring revision

Case 1	Case 2
Task: Manufacture ?wp48 **Conditions**: WorkPiece ?wp48 AscendingOutline ?ao9 DescendingOutline ?do9 **Subtask**: Process ?ao9 ?wp48 Process ?do9 ?wp48	**Task**: Manufacture ?wp353 **Conditions**: WorkPiece ?wp353 AscendingOutline ?ao2 DescendingOutline ?do2 HorizontalOutline ?ho2 **Subtask**: Process ?ao2 ?wp353 Process ?do2 ?wp353 Process ?ho2 ?wp353

Step 3: Case Revision

A problem of using generalized cases is that they might not model the target domain correctly. There are situations in which more than one generalized case can be applicable to the same task. Depending on the case reused, an incorrect plan may be obtained. To address this limitation, our bias is to select the case that

has conditions that are more specific. To implement this bias, Gen+ adds conditions to the cases. As a result, refined cases are more specific, and will be selected during retrieval. As we will explain later, our experiments show the adequacy of this bias. The revision process of Gen+ is performed when potential conflicts are identified between the newly captured cases and the existing ones.

We illustrate the idea of the case revision process with an example in a supply-chain domain. For a detailed discussion of the case revision algorithm, please see (Xu & Muñoz-Avila, 2003). Table 3 shows two generalized cases. Case 1 is a generalized case for manufacturing workpiece ?wp48, which has two outlines: ascending outline ?ao9 and descending outline ?do9. Case 2 is the same generalized case from Table 2. Case 2 has more conditions and a difference sequence of subtasks. Thus, even though the two cases have the same form of task, they should be reused to solve different projects.

Suppose that case 1 is added first in the case base and that case 2 is added afterwards. Because the conditions in case 1 are a subset of the conditions in case 2, any situation satisfying case 2 must also satisfies case 1. Thus, it will be an incorrect plan if the task used to generate case 2 is given again but case 1 is retrieved instead. Gen+ modifies case 1 and case 2 to ensure that such problem will not happen. The result of the modification is illustrated in Table 4. A new case, case 3, is added, having the same task as the original case 1 and case 2. Case 3 has a new, unique subtask vT ?wp3 ?ao3 ?do3. This subtask is used as the task for both case 1 and case 2. It links the two new cases to case 3. The negated conditions in new case 1 ensure that it will not be selected when the new case 2 is applicable. Whenever the new case 2 is applicable, the new case 1 is not applicable. Thus, if we give the same task and conditions as in the second column of Table 3, the new case 2 will be applicable, but not the new case 1.

Table 4: Case revision of Gen+

Case 3	New Case 1	New Case 2
Task: Manufacture ?wp3 **Condition**: WorkPiece ?wp3 AscendingOutline ?ao3 DescendingOutline ?do3 **Subtask**: vT ?wp3 ?ao3 ?do3	**Task**: vT ?wp3 ?ao3 ?do3 **Condition**: ¬HorizontalOutline ?ho2 **Subtask**: Process ?ao3 ?wp3 Process ?do3 ?wp3	**Task**: vT ?wp3 ?ao3 ?do3 **Condition**: HorizontalOutline ?ho2 **Subtask**: Process ?ao3 ?wp3 Process ?do3 ?wp3 Process ?ho3 ?wp3

Although the case revision process is cumbersome, it is transparent to the user. The user never sees the cases directly, but only the resulting decompositions. In addition, new tasks such as "vT" in Table 4 are not shown. The system simply skips the display of any intermediate level. The reason is that these tasks are added artificially for the revision process, but have no meaning in the actual domain.

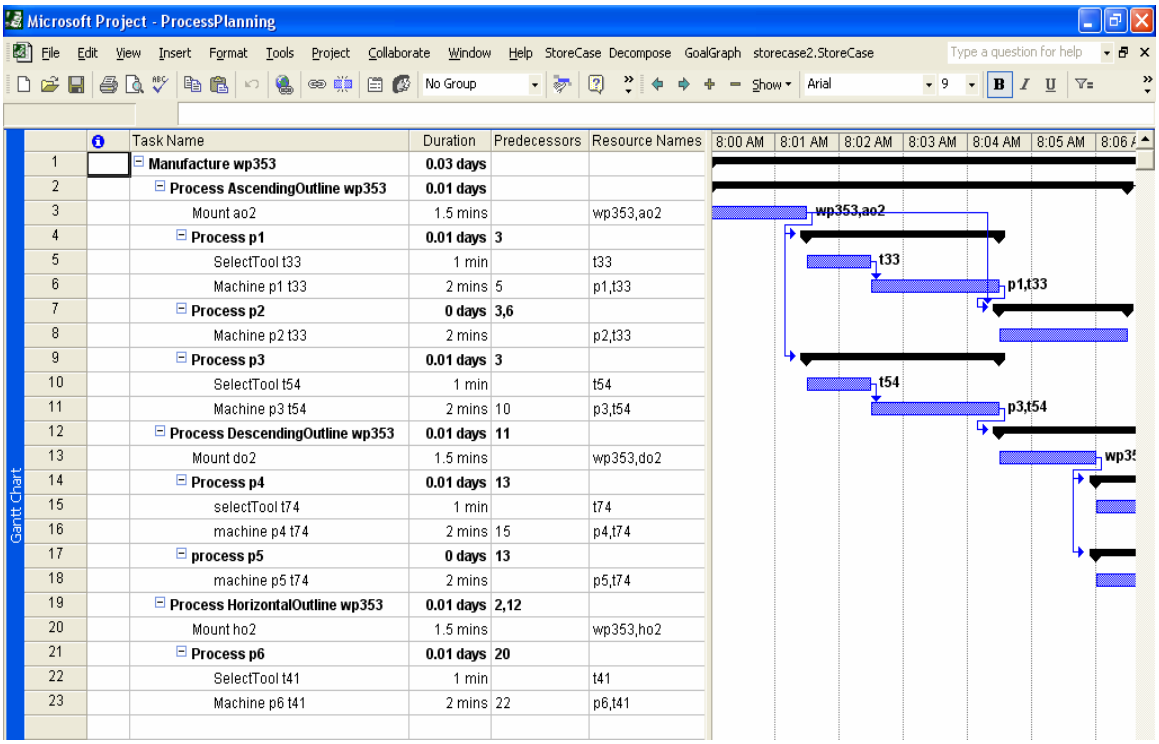
3.3 SHOP/CCBR

A generalized case C = ((h, Q, ST, <, ID*)* is reused to decompose a task *t* in a current project plan if *t* is an instance of h, and the conditions in Q can be instantiated by the resources in the current project plan. The case reuse component, SHOP/CCBR, is a variant of the HTN planner SHOP (Nau *et al*,, 1999). Once a generalized case C = (h, Q, ST, <, ID) is retrieved for a task-state pair (*t*,S), C is reused in standard HTN planning fashion. If Θ is a substitution fulfilling the applicability requirement of C, the task *t* is decomposed into the subtasks of STΘ. The task decomposition process continues recursively with the subtasks until primitive tasks are obtained. Figure 3 shows the completed decomposition of tasks #12 and #13 from Figure 1, obtained by SHOP/CCBR. Task #12 is decomposed into tasks #13 - # 18. Former task #13 is now task #19, and it has been decomposed into tasks #20 - #23. A major difference between SHOP and SHOP/CCBR is that the latter allows for partial order between the tasks. This is needed to represent the various relations between the tasks as summarized in Table 1.

3.4 The Goal Graph Component (GGC)

We identified a potential problem that results from the combination of reusing cases and the interactive nature of project management software. When CaBMA is used to complete parts of a project plan, the

system responds by finding applicable cases and reusing them. Case applicability is determined based on the matching between the case's elements and the elements of the current task. If parts of a project plan are obtained by case reuse, and the user edits some of these parts afterwards, the current plan may become inconsistent unbeknown to the user. A case reuse inconsistency occurs if a case C that was used to decompose a task t into subtasks in the WBS is no longer applicable as a result of edits made by the user in the project plan. This kind of semantic inconsistency is reflected by the fact that if t had been decomposed until after the edits were made, C would not have been selected. Instead, either a different case would have been selected or no applicable case would have been found.. These can be seen as semantic inconsistencies and are complementary to the syntactical inconsistencies that Microsoft Project™ can detect, which refer to the overlapped allocation of resources.



	Task Name	Duration	Predecessors	Resource Names
1	**Manufacture wp353**	**0.03 days**		
2	**Process AscendingOutline wp353**	**0.01 days**		
3	Mount ao2	1.5 mins		wp353,ao2
4	**Process p1**	**0.01 days**	**3**	
5	SelectTool t33	1 min		t33
6	Machine p1 t33	2 mins	5	p1,t33
7	**Process p2**	**0 days**	**3,6**	
8	Machine p2 t33	2 mins		p2,t33
9	**Process p3**	**0.01 days**	**3**	
10	SelectTool t54	1 min		t54
11	Machine p3 t54	2 mins	10	p3,t54
12	**Process DescendingOutline wp353**	**0.01 days**	**11**	
13	Mount do2	1.5 mins		wp353,do2
14	**Process p4**	**0.01 days**	**13**	
15	selectTool t74	1 min		t74
16	machine p4 t74	2 mins	15	p4,t74
17	**process p5**	**0 days**	**13**	
18	machine p5 t74	2 mins		p5,t74
19	**Process HorizontalOutline wp353**	**0.01 days**	**2,12**	
20	Mount ho2	1.5 mins		wp353,ho2
21	**Process p6**	**0.01 days**	**20**	
22	SelectTool t41	1 min		t41
23	Machine p6 t41	2 mins	22	p6,t41

Figure 3: A complete WBS after case reuse

Case Reuse Inconsistencies

We will now discuss the kinds of edits in a project plan that may result in case reuse inconsistencies.

- **Inconsistency by change in a resource**. These inconsistencies occur when the user removes a resource or replaces a resource with another one of different type. Since the resources are used to determine the applicability of a case, these kind of changes will usually make the case non applicable with the current instantiation of the variables. An example of such an inconsistency occurs in the following situation. Suppose that Task # 1 in Figure 3, Manufacture wp353, was decomposed by using the generalized case from Table 2. This results in the 3 subtasks: Tasks # 2, # 12, and #19. Suppose that the user removes the resource ascending outline a02. This means that the precondition *AscendingOutline ?ao2* of the generalized case cannot be satisfied. Therefore, the decomposition of Task # 1 into Tasks # 2, # 12, and #19 is no longer consistent with the cases.
- **Inconsistency by change in a task.** These inconsistencies occur when the user removes or renames a task. Since the task is used to determine the applicability of the cases, the case will be no longer applicable. For example, if the task Manufacture wp353 in Figure 3 is renamed as *Clean wp353*, then the decomposition is no longer consistent with the cases.
- **Inconsistency by change in an ordering link.** These inconsistencies occur when the ordering between tasks is removed. Since the applicability of a case is made based on the free resources that are available at a certain point of time removing an ordering link may cause some conditions not to be satisfied. For example, assume that the user removes the ordering restriction between Task # 6,

Machine p1 t33, and Task # 7, process p2. Since the later task is decomposed into Task # 8, Machine p2 t33, a conflict will occur because both Tasks # 6 and # 8 will require to use the tool t33 at the same time. The reader familiar with tools such as MS Project may recognize that MS Project will detect this kind of conflict. This syntactic inconsistency takes place because the resource can only be used once at any point of time. Thus, we have a situation in which a syntactic and a semantic inconsistency take place at the same time and for the same reason.

Case Repair

To solve case reuse inconsistencies, the Goal Graph Component (GGC) of CaBMA keeps track of all modifications to the current project plan in a data structure called Goal Graph (GG). These modifications include user edits and changes made by CaBMA. Any edit will trigger a propagation process in GGC. If there are any inconsistencies as results of the propagation, they will be pointed out for the user in a non-intrusive manner so that the user can decide when and how to deal with these inconsistencies. The main advantage of GGC is twofold. First, it propagates the effects of the edits immediately. Second, it is sound: all affected pieces of the plan will be detected. As a result, the affected pieces can be repaired without having to re-plan from the scratch.

Goal graphs are generated automatically during project planning sessions with Microsoft Project™. Every modification in the project plan causes an update in the GG. For example, suppose that the user removes tool t41 from the resources in the project plan shown in Figure 3. Recall that CaBMA obtained the decomposition of task #12. Since tool t41 is not available anymore, tasks #15, #16 and #18 will be affected. GGC will update the goal graph by propagating the effects and marking tasks #14 and #17 affected as well. No other tasks are affected as a result of this change.

GGC is built on top of the Redux architecture, which is a Justification Truth Maintenance System (JTMS) for planning contingencies (Petrie, 1992). Redux combines the theory of JTMS and the theory of Constrained Decision Revision (CDR). In JTMS, assertions (called nodes) are connected via a tree-like network of dependencies. The combination of JTMS and CDR enables dependency-directed backtracking and identification of pieces of the plan that are affected by the edits.

Redux represents relations between goals, operators and decisions. A goal g is decomposed into the subgoals G by applying an operator o. The assignments A indicate changes in the plan caused by applying o. A decision is a 4-tuple (g, G, o, A). As different operators may be applicable, the choice of an operator represents a backtracking point, and it is represented as a decision. Given a goal g, a conflict set is a pair of the form (g, (D1, .., Dn)), where (D1, .., Dn) is the collection of decisions for g. A decision is rejected if its associated operator is rejected. The reasons for rejections are explicitly represented as justifications, which are a list of assignments.

We mapped elements from a project plan into Redux. Since tasks are decomposed into subtasks, tasks and subtasks are mapped into Redux goals. Resources and orderings between tasks and activities are mapped into Redux assignments because they represent changes in the plan. Cases are mapped into Redux operators. With this mapping, GGC provides a complete representation of tasks, subtasks, orderings, and resources of a project plan. It also records alternative decompositions and their validities.

CaBMA uses special icons to mark any pieces of a project plan containing case reuse inconsistencies. Inconsistencies sometimes have a domino effect when other parts of the plan become inconsistent too. The propagation mechanism of GGC deals with this domino effect by identifying affected tasks and their associated justifications. GGC not only detects and propagates modifications, but also provides alternative solutions to the user. When a task is being decomposed, all possible decompositions will be generated and recorded, but only one will be displayed to the user. When the current decomposition is not applicable any more due to case reuse inconsistency, GGC selects one of these alternative decompositions based on user preferences. If the new decomposition contains non-primitive subtasks, the user can ask SHOP/CCBR to decompose these subtasks anew.

Figure 4 shows the case repair procedure in GGC. When an inconsistency occurs due to edits to a task T, the decomposition D for T will be labeled as invalid (line 1). Otherwise the algorithm terminates (line 2). The user can choose a valid alternative decomposition D_T, should one exist. GGC labels the new decomposition as valid and updates the GG to record the modifications (lines 3-6). If there are no alternative decompositions available, which means T cannot be repaired (line 8), then GGC calls the case repairing procedure recursively with the parent task of the current task to propagate the effect of T's loss of validity (lines 9, 13). Sometimes even being aware of inconsistencies, the user may still want to keep the decompositions. In such situation, GGC does not propagate the effect of the changes (lines 10-12).

```
caseRepair(T, res(T), GG)
 // Input: T is a task; res(T) are the resources associated with T; GG is the goal graph
 // containing T; D is the current decomposition for T
// Output: GG is updated with a valid decomposition for T, if any exists

  1.  if invalidDecomposition(T,D) then  label(D, invalid)
  2  else return GG
  3. if a valid decomposition, D_T of T with resources res(T) exists then
  4.          UserSelectdecompose(T, D_T, GG)
  5.          label(D_T, valid)
  6.          return GG
  7. else
  8.      label(T, invalid)
  9.     Let P be the parent task of T
 10.     if UserAllows(P) then
 11.         return GG          // GG is returned without change upon the user's choice
 12.     else
 13.          caseRepair(P, res(P), GG)
```

Figure 4: Case-repairing procedure

Figure 5 follows the example from Figure 3, illustrating the plan obtained once the repair procedure takes place when the user removes the tool t74. In the original decomposition (Figure 3), machining the areas p4 and p5 was done with the same tool, t74. These two areas are of different types but the tool t74's properties allow it to be used on both of the areas. Now that tool t74 is no longer available, CaBMA finds alternative tools but their properties do not allow them to be used in both areas. Thus, the previous plan needs to be repaired. CaBMA repairs the plan in two different places. First, the tool t88 is selected for tasks

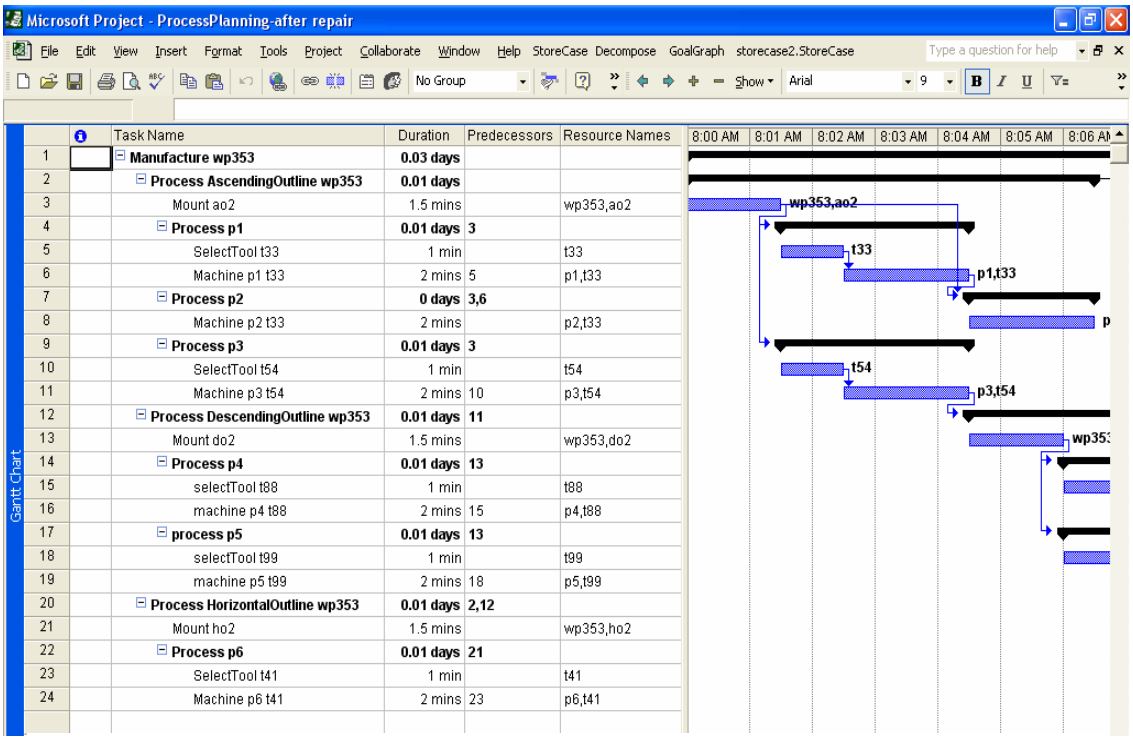


Figure 5: A complete WBS after case repair

#15 and #16. Second, task #17 is now decomposed into two subtasks. Subtask #18 selects a tool t99, and subtask #19 uses t99 to machine p5.

4 Empirical Results

We performed an experiment to measure the planning precision of CaBMA. As a baseline for comparison, we use data obtained from a straightforward generalization algorithm called ***Gen***, which uses the same case generalization procedure as CaBMA but does perform the case revision process (Step 3 of Section 3.2). That is, it performs the first two steps, case acquisition and case generalization, as the Gen+ component, but it does not perform the third step, case refine step. In the context of case-based planning, we define precision as A/B, where A is the number of times that correct plans were generated and B is the number of times that plans were generated. We also define recall as B/C, where C is the number of solvable problems given.

We used a subset of the UMTranslog domain (Erol et al, 1994). In this domain, objects need to be moved between different places. Trucks and planes are used to transport objects. UMTranslog defines objects that can be a combination of the various types such as *liquids* and *perishables*. For example, milk can be defined as a liquid and perishable object. The types of trucks and planes in UMTranslog can be a combination of various types such as *tanker* and *refrigerated*. Each type is specially suited to transport one object of the responding type (e.g., a refrigerated tanker truck is suitable to transport milk).

The UMTranslog is adequate for our experimental purposes for two reasons. First, even though the UMTranslog domain is a synthetic domain, in previous work we shown that it can naturally modeled by *Microsoft Project*™ (Mukammalla & Muñoz-Avila, 2002). We developed an automatic procedure utilizing the one-to-one mapping between HTN and WBS, which takes a plan generated by SHOP and represents it in *Microsoft Project*™. Second, this synthetic domain allows us to determine if a given plan is correct. Therefore, we can compute the precision of planning with CaBMA.

For our experiments, we used a simulated user that enters plans in *Microsoft Project*™. To simulate this user, a random problem generator feeds problems to the HTN planner SHOP to generate plans. These plans are then captured as cases by CaBMA, with the case revision step (Gen+) and without case revision (Gen). The random problem generator was also used to generate a test set which contains 50 problems. Two sets of plans were obtained by using Gen+ and Gen. Table 7 shows the results:

- Number of problems used to construct new solutions (column labeled Training set).
- Number of cases added to case bases (column labeled Cases).
- Number of problems solved with Gen+ and Gen (column labeled Solutions Found).
- Number of solutions that were incorrect (columns labeled Error).

Table 5: Results of the Experiment

Training Set	Cases	Algorithm	Solutions Found	Error
50	31	Gen	14	9
		Gen+	7	0
160	101	Gen	28	10
		Gen+	16	0
240	153	Gen	28	9
		Gen+	19	0

Of the 50 problems in the test set, 28 were solvable. Gen was capable of solving all 28 solvable problems but 9 of the solutions generated were incorrect. Therefore using Gen, we obtained a 100% recall and 68% precision scores. Using Gen+, we obtained 19 solutions, all of which were correct. Thus, for Gen+, we obtained 67% recall and 100% precision scores.

The results of the experiment illustrate a trade-off between solving more problems and solving them correctly. The Gen procedure generalizes the cases without any constraints. Thus it solves more problems but also makes more mistakes. Such mistakes may reduce the user trust in an interactive system such as project planning. On the other hand, Gen+ reduces substantially the likelihood of producing incorrect plans by constraining the cases, albeit by reducing the number of solutions generated.

5 Related Work

To the best of our knowledge, CaBMA is the first system to support the automatic acquisition, refinement and reuse of project plans. However, some other systems, which we now discuss, are related to CaBMA.

Case-based reasoning (CBR) has been used in a variety of applications including for learning customer preferences (Branting, 2004), for knowledge discovery (Patterson *et al*., 1999), and for active databases (Li & Yang, 2001). Our work with CaBMA showcases project planning as a new field of application for CBR.

The RealPLAN system advocates the separation of planning from resource allocation (Srivastava & Kambhampati, 1999). RealPLAN is inspired by project planning. As opposed to CaBMA, RealPLAN uses standard planning techniques for plan generation, and, thus, requires a complete domain model to generate the plans.

Providing tools for knowledge acquisition has been a frequently studied research topic. For example, in the EXPECT project (Blythe *et al*., 2001), an integrated suit of intelligent interfaces is used to capture knowledge. The EXPECT project shows that by integrating these interfaces, it is possible to elicit the context of the user actions, which enhances the knowledge acquisition capabilities. The main difference is that CaBMA does not use ad-hoc interfaces to capture knowledge; instead, it captures the cases from the interactions of the user with the project management software.

SIPE-2 is a system for interactive planning and execution (Wilkins, 1988). SIPE-2 uses operators for action executions. It also has execution-monitoring techniques to accept new information and interact with the user. CaBMA automatically captures cases from previous project plans, and reuses them for planning. However, CaMBA does not require operators for generating WBSs.

NONLIN is a partial order planner, which generates hierarchical project network (Tate, 1977). The main difference is that NONLIN assumes a domain theory modeled using a task formalism representation. On the other hand, CaBMA uses cases as its source for domain knowledge.

6 Summary

We presented CaBMA, a case-based planning assistant built on top of *Microsoft Project*™. CaBMA is a prototype designed to assist the project manager by providing suggestions for refining tasks in the current work breakdown structure, and by reusing experience captured from previous project planning episodes. CaBMA consists of four components. The first component, Task Dictionary Component (TDC), uses task ontology to assist the user with task identification. The second component is Gen+, which automatically captures cases. To increase the coverage of the case base, cases are generalized from task decompositions in a WBS. Gen+ also provides a procedure to refine the cases over time to avoid inconsistency due to case generalization. The third component is SHOP/CCBR, which follows a HTN planning paradigm for case reuse. As user edits may result in case reuse inconsistencies, the last component, Goal Graph Component (GGC), detects such inconsistencies and indicates them to the user in an unobtrusive manner. CaBMA's crucial contribution is that it demonstrates for the first time how to add a knowledge layer top of commercial project management software, going beyond the editing and bookkeeping capabilities that this software is traditionally limited to.

References

Bergmann, R. and Stahl, A., Similarity Measures for Object-Oriented Case Representations. In: *Proceedings of the European Workshop on Case-Based Reasoning* (*EWCBR-98)*. Springer, 1998.

Blythe, J., Kim, J., Ramachandran, S., and Gil, Y. An Integrated Environment for Knowledge Acquisition. In: *Proceedings of the International Conference on Intelligent User Interfaces*, 2001.

Branting, L. K. Learning Feature Weights from Customer Return-Set Selections. *The Journal of Knowledge and Information Systems (KAIS)*. Springer, 2004.

Erol, K., Nau, D., & Hendler, J. HTN planning: Complexity and expressivity. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press, 1994.

Hammond, K. J., Chef: A model of case-based planning. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*. AAAI Press, 1986.

Li, S., & Yang, Q. ActiveCBR: An Agent System That Integrates Case-Based Reasoning and Active Database. *The Journal of Knowledge and Information Systems (KAIS)*. Springer, 2001.

Myers, K.L.; & Wilkins, D.E. *The Act Formalism*. Artificial Intelligence Center, SRI International, Menlo Park, CA, version 2.1 edition, 1997

Mukammalla, S. & Muñoz-Avila, H. Case Acquisition in a Project Planning Environment. In: *Proceedings of the Sixth European Conference on Case-Based Reasoning (ECCBR-02)*. Springer, 2002.

Muñoz-Avila, H., Gupta, K., Aha, D.W., Nau, D.S. Knowledge Based Project Planning. *Knowledge Management and Organizational Memories*. 2002.

Nau, D., Cao, Y., Lotem, A., & Muñoz-Avila, H. SHOP: Simple Hierarchical Ordered Planner. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*. Stockholm: AAAI Press, 1999.

Patterson, D. W., Anand, S. S., Dubitzky, W., Hughes, K. G. Towards Automated Case Knowledge Discovery in the M2 Case-Based Reasoning System. The *Journal of Knowledge and Information Systems (KAIS)*. Springer, 1999.

Petrie, C. Constrained decision revision. In: *Proceedings of the National Conference of Artificial Intelligence (AAAI-92).* AAAI Press, 1992.

Project Management Institute (PMI). *PMI's A Guide to the Project Management Body of Knowledge (PMBOK® Guide).* Technical Report. Release No.: PMI 70-029-99. Project Management Institute, 1999.

Smyth, B., and Keane, M.T., Remembering to forget: A Competence-Preserving Case Deletion Policy for case-based reasoning systems. In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-95)*. AAAI Press, 1995.

Srivastava, B., Kambhampati, S. *Scaling up Planning by teasing out Resource Scheduling*. *Tech*. Report ASU CSE TR 99-005. Arizona State University, 1999.

Tate, A., Generating Project Networks. In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77),* 888-893, 1977.

Wilkins, D. E. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc, 1988.

Xu, K. & Muñoz-Avila, H. CaBMA: Case-Based Project Management Assistant. In: *proceedings of the Innovative Applications of Artificial Intelligence Conference on Artificial Intelligence* (IAAI-04). Springer, 2004.

Xu, K. & Muñoz-Avila, H. CBM-Gen+: An Algorithm for Reducing Case Base Inconsistencies in Hierarchical and Incomplete Domains. In: *Proceedings of the International Conference on Case-Based reasoning (ICCBR-03)*. Springer, 2003.

Mr. Ke Xu is a PhD candidate at the Department of Computer Science and Engineering at Lehigh University, where he is finishing his dissertation on the topic of Case-Based Task Decomposition with Incomplete Domain Descriptions. Mr. Xu did his undergraduate studies at Tsinghua University in China. He also holds an M.S. in Computer Science from Lehigh University. His research interests include case-based reasoning and planning.

Dr. Héctor Muñoz-Avila is an assistant professor at the Department of Computer Science and Engineering at Lehigh University. Prior to joining Lehigh, Dr. Muñoz-Avila did post-doctoral studies at the Naval Research Laboratory and the University of Maryland at College Park. He received his PhD from the University of Kaiserslautern (Germany). Dr. Muñoz-Avila has done extensive research on case-based reasoning, planning, and machine learning, and in advancing game AI with AI techniques. He has written over 10 journal papers and over 30 refereed conference/workshop papers on the subject. Two of these papers received awards. He was also awarded an NSF CAREER award. Dr. Muñoz-Avila has been chair, program committee member and a reviewer for various international scientific meetings. He will be program co-chair of the Sixth International Conference on Case-Based Reasoning (ICCBR-05) to be held in Chicago, Il (USA).