

Maintaining Consistency in Project Planning Reuse

Ke Xu & Héctor Muñoz-Avila

Department of Computer Science and Engineering
19 Memorial Drive West
Lehigh University
Bethlehem, PA 18015, USA
{kex2,hem4}@lehigh.edu

Abstract. In this paper we describe an application of JTMS technology for maintaining consistency of pieces of a project plan obtained by case reuse. In our approach project plans are constructed interactively with the support of a CBR module. The user can either make edits to a project plan, or call a case reuse module for completing parts of it. As the user is making modifications on the project plan, conditions about applicability of the cases may change. We present an implementation of JTMS technology on a commercial tool for project planning to detect possible inconsistencies in reusing cases as a result of these changes.

1 Introduction

Project planning is a business process for successfully delivering one-of-a kind products and services under real-world time and resource constraints. One-of-a kind means that the product or service differs in some distinguishing way from similar products or services (Anderson *et al.*, 2000). Several software packages for project management are commercially available. These include *MS Project*TM (Microsoft) and *SureTrak*TM (Primavera Systems Inc). These interactive systems help users elicit a work-breakdown structure (WBS), which indicates how high-level tasks can be decomposed into simple work units. These packages also contain a suite of tools to control the scheduling of the tasks and the management of resources.

In previous work (Muñoz-Avila *et al.*, 2002), a knowledge-layer for existing tools supporting project planning was proposed. This approach, called knowledge-based project planning (KBPP) advocates the use of CBR technology to reduce the time required to generate WBSs. The main motivation for using CBR in this context is that knowledge about how to formulate project plans is mostly episodic, even though general guidelines have been formulated to help with the project plan elicitation process (PMI, 1999; Liebowitz, 1999).

We implemented KBPP on top of *MS Project*TM (Mukkamalla & Muñoz-Avila, 2002). During trials with this implementation we identified a problem that is due to the interactive nature of the KBPP process. When the user requests the KBPP system to complete parts of a project plan, the system responds by determining applicable cases and reusing them. Case applicability is determined based on two factors: the task being completed and the available resources. The problem may arise if the user

later changes the available resources and/or the task being solved. An inconsistency occurs when cases previously reused are no longer applicable. We refer to these inconsistencies as *case reuse inconsistencies*. These can be seen as semantic inconsistencies and are complementary to the syntactical inconsistencies that most commercial project planning tools can detect. A typical syntactical inconsistency is the over allocation of resources.

To deal with case reuse inconsistencies, we implemented a new component, the Goal Graph System (GGS), in our KBPP system. GGS is based on the Redux architecture, which is a justification truth-maintenance system (JTMS) for dealing with planning contingencies (Petrie, 1992). GGS keeps track of all modifications being performed to the current project plan in a data structure called the Goal Graph (GG). These modifications include user edits and case reuse episodes. Edits that may result in case reuse inconsistencies will trigger a JTMS propagation process in GG. Any inconsistencies detected are displayed to the user in a non-intrusive manner, allowing him to decide at what point he wants to deal with them. GGS has two crucial properties: first, GGS can propagate the effects of user edits rapidly. Second, GGS has a sound JTMS propagation mechanism that ensures the detection of all inconsistent pieces of the project plan.

In this paper we are going to explain in detail the different kinds of case reuse inconsistencies that may occur in project planning, how these are detected by GGS and discuss details of the Goal Graph.

2 Related Work

Our studies are closely related to replanning. In replanning, a plan is modified when some changes occur in the problem situation making pieces of the original plan invalid (Petrie, 1991). In the CAPlan/CbC system, replanning techniques are used to implement an adaptation procedure (Muñoz-Avila et al, 1996). CAPlan/CbC also uses a variation of the Redux architecture. The main difference with our KBPP system is that CAPlan/CbC assumes that a complete domain theory is available indicating all possible planning steps. In KBPP such an assumption is not possible since most of the domain knowledge is episodic and no complete domain theory exists. A second major difference is that the KBPP must be as non-intrusive as possible. When inconsistencies are detected, they are pointed out in the interface but there is no automatic replanning process. A third difference is that our KBPP system uses a hierarchical task network (HTN) representation compared to the STRIPS plan representation in CAPlan. HTNs have been shown to be more expressive than STRIPS (Erol et al, 1994). As a result, our use of the Redux architecture had to be adjusted accordingly.

A complementary problem to WBS elicitation is the problem of resource allocation. Initial research has been done to address resource allocation in the context of project management (Srivastava, Kambhampati & Minh, 2000).

3 Knowledge-Based Project Planning

In (Muñoz-Avila et al, 2002), KBPP is proposed to assist planners in the development of WBS by using a hierarchical case decomposition algorithm. This approach is based on the observation that so-called Work-breakdown structures (WBS), as the main project planning representation paradigm is called, have a one-to-one correspondence with the hierarchical task networks (HTNs).

Figure 1 shows a snapshot of a work-breakdown structure in Microsoft Project. The task *Distribute-packages* is decomposed into four subtasks (first column): *Distribute package100 from Allentown to NYC*, *Distribute package200 from Bethlehem to Beijing*, *Distribute package300 from Easton to Newark*, and *Distribute package400 from LU to UIC*. Some tasks are called activities and represent concrete actions to be performed. For example, *drive truck47 to Allentown* is considered an activity. Tasks have assigned resources (second column). For example, the task *Drive truck47 to Allentown* has two assigned resources: truck47 and Allentown. Finally, tasks have ordering relations among them (third column). For example, the task *Drive truck47 to Allentown* is ordered before the task *Load truck47 with package100*. More generally, there are 3 kinds of relations in a WBS:

- Task - subtask relations
- Task-resource assignments
- Task- task ordering relations

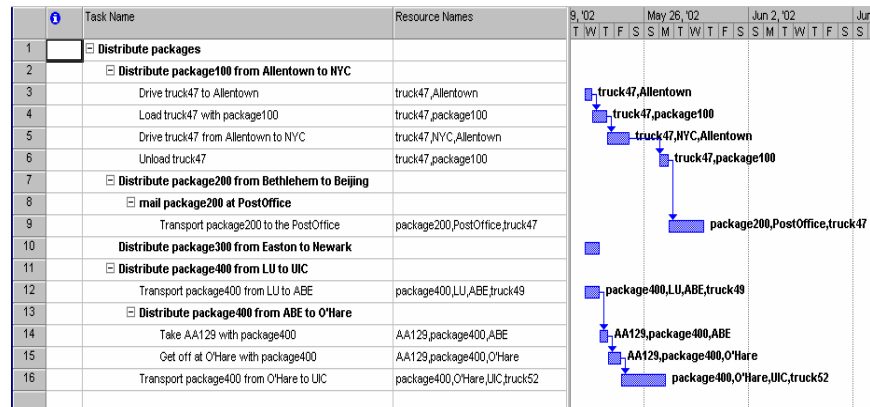


Fig. 1. Snapshot of a work-breakdown structure

Cases contain H-level decompositions in the WBS and consist of the following elements:

- Task h to be decomposed.
- Subtasks ST decomposing h if the case is applicable
- Conditions C indicating when the case is applicable
- Ordering relations between the subtasks

Cases represent generalizations of WBSs. That is, cases contain variables instead of the original elements mentioned in the WBS. For example, in place of the element `package100`, cases use the variable `?package100` (variables are denoted with a question mark). Table 1 shows the case for the decomposition of the task *Distribute package100 from Allentown to NYC* from Figure 1. This task is represented as (distribute `?package100` from `?Allentown` to `?NYC`). The four subtasks are represented using the same convention. For example, the subtask *Drive truck47 to Allentown* in Figure 1 is represented in the case as (drive `?truck47` to `?Allentown`). The four resources associated with the subtasks are used to define the conditions of the case. The conditions indicate the type of the resource. For example, the resource Allentown is used to define the condition (city `?Allentown`).

Table 1. A generalized case for a WBS decomposition

Case 1
Task: (distribute <code>?package100</code> from <code>?Allentown</code> to <code>?NYC</code>) Condition: (city <code>?Allentown</code>) (package <code>?package100</code>) (city <code>?NYC</code>) (truck <code>?truck47</code>) Subtask: (Drive <code>?truck47</code> to <code>?Allentown</code>) (Load <code>?truck47</code> with <code>?package100</code>) (Drive <code>?truck47</code> from <code>?Allentown</code> to <code>?NYC</code>) (Unload <code>?truck47</code>)

In (Mukammalla & Muñoz-Avila, 2002) a procedure is presented to automatically capture cases from WBSs. WBSs are generalized to improve the coverage of the case base. Since the case capture and reuse procedures are automatic, end-users are not expected to see these cases. There are some issues regarding soundness of the generalized WBS but we omit discussing them because they are beyond the purpose of this paper.

4 Case Retrieval and Reuse

Given a task t in the WBS, the user can request the KBPP system to automatically decompose t into subtasks. The KBPP system selects applicable cases by performing the following two tests:

1. The task of the case matches t .
2. The conditions of the case match the existing resources.

For Example, the case in Table 1 is applicable to decompose the task *Distribute package300 from Easton to Newark* from Figure 1. First, the task of the case matches

this task with the substitution {?package100 → package300, ?Allentown → Easton, ?NYC → Newark}. Second, the conditions match existing resources. A truck, truck33, is available. The list of available resources in MS Project can be viewed under the so-called Resources Sheet. For determining the applicability of a case, the KBPP system collects all available resources that are not used by any task whose scheduled times overlaps with the task being decomposed. In this way, the KBPP system ensures that only free resources will be assigned.

Once an applicable case is selected, its subtasks are used to decompose the target task. Figure 2 shows the resulting decomposition when the case in Table 1 is used to decompose the task *Distribute package300 from Easton to Newark*. Notice that in addition to the decomposition, the resources used to match the conditions of the case have been assigned. This assignment is necessary to maintain consistency in the usage of resources.

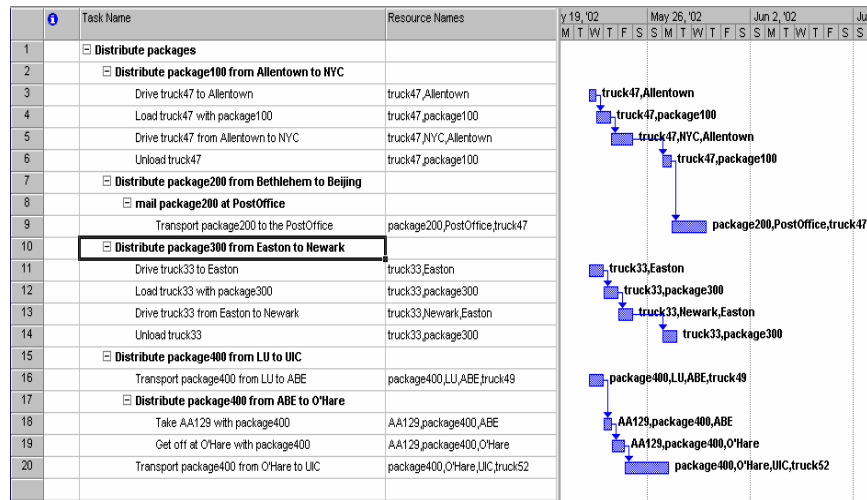


Fig. 2. Snapshot of the refined WBS after case reuse

5 Case Reuse Inconsistencies

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

5.1 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 if the user removes truck33 from the subtasks of the task *Distribute package300 from Easton to Newark* from Figure 2. In this situation, the condition (truck ?truck47) of the case in Table 1 is no longer valid. Thus, the decomposition of the tasks into the subtasks is invalid. Another example of an inconsistency occurs if the user replaces the resource truck33 with truck10 in the activity *Load truck33 with package300*. The inconsistency occurs because all dependent tasks using truck33 need to be changed as well.

5.2 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 *Distribute package300 from Easton to Newark* from Figure 2 is renamed as *Distribute package300 from Easton to Reading*, then the decomposition is no longer valid.

5.3 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. In Figure 2, assume that the decomposition of the task *mail package200 at post office* into the task *Transport package200 to the PostOffice* was performed by case reuse. If the ordering link from the task *Unload truck47* to the task *Transport package200 to the PostOffice* is removed, the case may not be applicable. The reason for this is that truck47 is used by both tasks and eliminating the ordering link will make both tasks be performed at the same time, while the resource can only be assigned to one of them.

The reader who is 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 at most once during any period 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. The user has several alternatives to solve these inconsistencies. Solving the semantic inconsistency will ensure that the syntactic inconsistency is also solved. However, the opposite is not necessarily true. If the user decides to remove the resource truck47 from the task *Unload truck47*, the syntactic inconsistency will be solved but the semantic inconsistency still remains since the case is inapplicable.

6 The Goal Graph System

We have seen how edits in a project plan may result in case reuse inconsistencies. The simple examples of case reuse inconsistencies discussed previously show only one task decomposition being affected. However, edits may have a domino effect in which several pieces of the plan will become inconsistent. The Goal Graph System was created for two reasons: first, we wanted a mechanism to propagate the effects of user edits rapidly. Second, we wanted a sound mechanism to ensure detection of all inconsistent pieces.

At the core of the Goal Graph System is the Goal Graph (GG), which is based on the Redux architecture. Redux combines the theory of justification-based truth maintenance system (JTMS) and constrained decision revision (CDR) (Petrie, 1992). In a truth maintenance system (TMS), assertions (called nodes) are connected via a tree-like network of dependencies. The combination of JTMS and CDR provides the ability to performed dependency -directed backtracking, which is adopted in GG to propagate changes.

6.1 Justification Truth Maintenance Systems

In JTMS, each assertion is associated with a justification (Doyle, 1986). A justification consists of two parts: an IN-list and an OUT-list. Both IN-list and OUT-list of a justification are sets of assertions. The assertions in the IN-list are connected to the justification by “+” links, while those in OUT-list are linked by “-” links. The validation of an assertion is supported by the justification that it is associated with, i.e., an assertion is believed when it has a valid justification. A justification is valid if every assertion in its IN-list is labeled “IN” and every assertion in its OUT-list is labeled “OUT”. If the IN- and OUT-lists of a justification are empty, it is called a *premise justification*, which is always valid. A believable assertion in JTMS is labeled “IN”, and an assertion that cannot be believed is labeled “OUT”. To label each assertion, 2 criteria about the dependency network structure need to be met: *consistency* and *well-foundness*. Consistency means that every node labeled IN is supported by at least one valid justification and all other nodes are labeled OUT. Well-foundness means that if the support for an assertion only depends on an unbroken chain of positive links (“+” links) linking back to itself, then the assertion must be labeled OUT.

In a consistent JTMS, each node is labeled either IN or OUT. A node is labeled IN when it has a valid justification, i.e., the assertions in the IN-list of the justification are all labeled IN, and the assertions in the OUT-list of the justification are all labeled OUT. A node is labeled OUT if either it has an invalid justification (which means that either some assertions in the IN-list are labeled OUT, or some in the OUT-list are labeled IN, or both situations occur), or it has no associated justification that supports it.

6.2 The Goal Graph

The goal graph represents relations between goals, operators and decisions. A goal is decomposed into subgoals by applying an operator. The applied operator is called a decision. The assignments represent conditions for applying the operator. Figure 3 shows the relationship between a decomposed goal, its subgoals, the operator list, the decision, and the assignments.

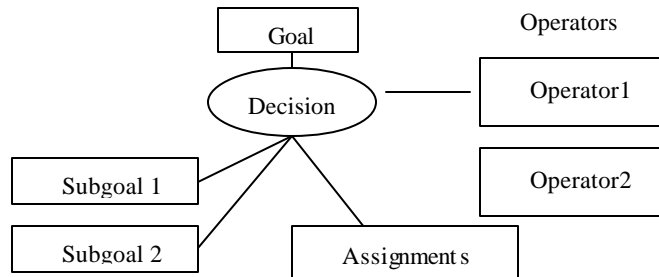


Fig. 3. A decision in the goal graph

Figure 4 shows a sketch of the goal graph. The first goal list from the top represents the main goals. A goal may have several decisions, one for each possible operator that can be chosen to achieve the goal. In GG, decisions decompose goals into the subgoals. A decision contains a goal list, storing all the subgoals of the goal. Assignments needed for applying the decision to the goal are collected in an assignment list, which is also contained in the decision.

A JTMS mechanism is built on GG. A decision is valid if all the assignments in its assignment list are valid, and all the subgoals in its goal list are valid. Valid decisions are labeled “N”. For a goal, all the decisions in its decision list labeled “IN” are applicable, which means the goal can be decomposed by those valid decisions.

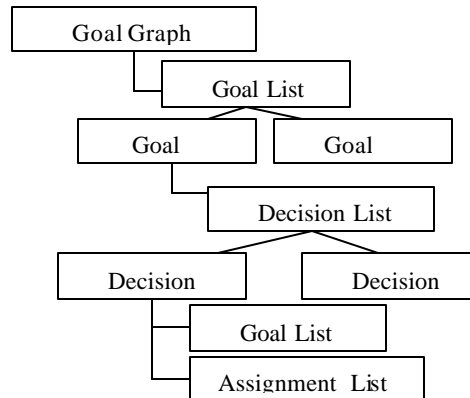


Fig. 4. Sketch of a goal graph

If for some reason, the validity of some assignments of a valid decision changes, then the decision may become invalid. GG incorporates a JTMS mechanism, so that the changes can be automatically propagated.

6.3 The Goal Graph in KBPP

For applying the Goal Graph System into KBPP, we mapped the elements of our KBPP approach into Goal Graphs. Since tasks are decomposed into subtasks, tasks were mapped into goals. A task may be associated with some resources. These resources are mapped as assignments in GG. Ordering relationships between tasks are also mapped into assignments.

Table 2. Map of KBPP elements into GG

KBPP	Goal Graph
Task	Goal
Subtask	Subgoal
Task-Resource assignment	Assignment
Ordering link	Assignment
System-made decompositions	Operator
User-made decompositions	Operator

One of the most challenging aspects of our work was to cope with the interleaving of user-made task decompositions and system-made decompositions (i.e., case reuse). We decided to map both kinds of decompositions as operators since they decomposed goals into subgoals. Decisions are labeled as user-made or system-made depending on the situation. The reason for this is that only system-made decompositions can be determined to be semantically inconsistent. Thus, GGS needs to know whether a decision is user-made or system-made during the JTMS propagation procedure.

In summary, the mapping of KBPP into GG results in the following dependencies:

- Subtasks depend on their parent tasks
- Subtasks depend on the decision (user-made, system-made) introducing them
- Decisions depend on the task they accomplish
- Task-resource assignments and ordering links depend on the decision that added them

These dependencies determine the next elements that are accessed in the JTMS propagation process.

6.4 Implementation and Example

We implemented the Goal Graph System in Java and established a communication module between GGS and Microsoft Project (called MSP from now on). When the user or the system makes some changes to the project plan, such as adding or deleting

invalid. Any subtasks of these subtasks will become invalid as well. In addition, the goal *Distribute package300 from Easton to Newark* will become invalid (However its parent task, *distribute-packages*, will not become invalid since the other 3 children are still valid). GG allows a systematic propagation of these changes by following the dependencies between the plan elements.

Once inconsistencies are detected by GG, a special icon is displayed in front of the affected tasks (Figure 6). This icon notifies the user about the inconsistency in an unobtrusive manner. Since the JTMS propagation is done after each edit, inconsistencies will be marked immediately after the infringing edit is made.

7 Conclusions and Future Work

Knowledge-based project planning is a promising application field of CBR technology. The knowledge in KBPP is mostly episodic and represented in a formalism that facilitates its automatic case capture and reuse. Early trials with an implementation of KBPP on top of a commercial tool made evident a consistency problem that is due to the interactive nature of the KBPP process. The problem arises when pieces of a project plan obtained with case reuse become invalid because of user edits. We presented a complete catalog of edits that may result in case reuse inconsistencies. These kinds of semantic inconsistencies are complementary to syntactical inconsistencies that most commercial project planning tools can detect. In this paper we presented GGS, the component of our KBPP system that was developed to address this problem. GGS maintains a Goal Graph representing dependencies between the pieces of a project plan. The Goal Graph offers a natural representation for project plans and facilitates the detection of case reuse inconsistencies. When edits are made to the project plan, a JTMS propagation procedure detects inconsistencies. These inconsistencies are then displayed to the user in a non intrusive manner.

In future work we plan to extend GGS to be able to automatically suggest repairs to the case reuse inconsistencies. In addition we are planning to deploy our KBPP system in an adequate environment to evaluate its impact in an organization (Davenport & Prusak, 1997).

References

1. Anderson, V.; Bradshaw, D.; Brandon, M.; Coleman, E.; Cowan, J.; Edwards, K.; Henderson, B.; Hodge, L.; & Rundles, S. (2000). Standards and Methodology of IT Project Management. *Technical Report*. Office of Information Technology. Georgia Institute of Technology.
2. Davenport, T; & Prusak, L. (1997). *Working Knowledge, How Organizations Manage What They Know*. Harvard Business School Publishing.
3. Erol, K; Hendler, J; & Nau D.S. (1994). HTN Planning: Complexity and Expressivity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-94)*. AAAI Press.
4. Liebowitz, J. (1999). *Knowledge Management Handbook*, CRC Press, Boca Raton, FL.

5. 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.
6. Muñoz-Avila, H. & Weberskirch F.: Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions In *Proceedings 3rd International Conference on AI Planning Systems (AIPS-96)*, AAAI-Press, 1996.
7. Muñoz-Avila, H., Gupta, K., Aha, D.W., Nau, D.S. Knowledge Based Project Planning. In *Knowledge Management and Organizational Memories*. 2002.
8. Petrie, C. (1991). *Planning and Replanning with Reason Maintenance*. PhD thesis, University of Texas at Austin, Computer Science Dept.
9. Petrie, C. (1992). Constrained decision revision. In *Proceedings of AAAI-92*. AAAI Press.
10. Project Management Institute (PMI). (1999). PMI's A Guide to the Project Management Body of Knowledge (PMBOK® Guide). *Technical Report*. Release No.: PMI 70-029-99. Project Management Institute.
11. Srivastava, S.; Kambhampati, R.; Minh, B. (2001). Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *ASU CSE Technical Report*.