A Model of Planning and Enactment Support in Software Development Projects

Sigrid Goldmann
University of Kaiserslautern
sigig@informatik.uni-kl.de

Abstract

In recent years, software development has become increasingly complex as requirements multiplied, and geographically distributed development became necessary and/or desirable. This complicates not only the enactment of software development projects, but also makes project planning and management much more difficult. Especially keeping the plan up to date, and distributing information about changes during project planning as well as plan enactment to everybody concerned, become more difficult with increasing project complexity while at the same time growing in importance.

In this paper, we propose an approach to facilitating not only project enactment but also project planning and management, by tracking all decisions made during project planning and enactment, and managing dependencies between these decisions. This allows us to feed enactment data back into the plan, either by automatically reacting to enactment events and plan changes, or by notifying the appropriate person(s).

We introduce an extendable model of planning and plan enactment that formalizes the dependencies between activities likely to occur during project planning and enactment, and provides the means to specify appropriate actions to be taken during change management.

1. Introduction

In complex software development projects, several problems frequently arise:

- Since software development is a highly creative process, unexpected changes occur on a regular basis. This means that initial plans are out-of-date almost immediately after project enactment has started, and coordinating the project, especially keeping everybody involved informed about information relevant to them, is a significantly important, as well as significantly difficult, responsibility that software project managers are confronted with.
- Decisions made by individuals, project planners as well as software developers, are often made and implemented informally, their rationales being known to the person making the decision and maybe a few others, but never documented in a way that makes them available for later reference. This means that reasons for a decision might be lost, complicating later changes.
- These problems become even more complex when geographical dispersion of project participants comes into play. In that case, informal means of communication, e.g. the casual exchange of information during coffee break, are no longer available, placing an even greater emphasis on the need to formalize and, where possible, automate the distribution of relevant project information.

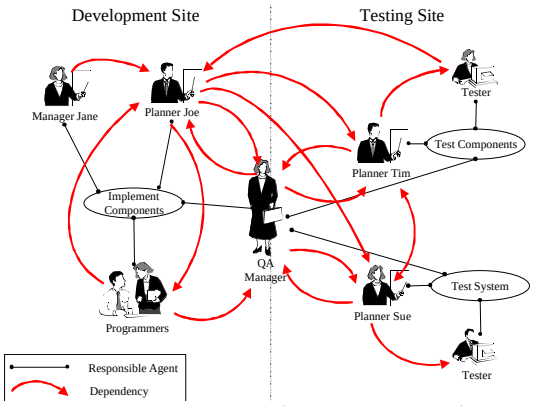

Figure 1: Planning Scenario for a Distributed Software Development Project

Suppose for example that a large software company with several development sites takes on an EJB-based software development project for an accounting system (see Figure 1). Implementation planner Joe needs to create a detailed plan for the component implementation phase. As inputs for his planning activity, he needs to have the system design, as well as information about what predefined Java classes will be available to facilitate implementation. The latter depends on which development tool is to be used, since some tools include EJB basic classes, and others do not. The decision for the development tool will be made by the project's central manager, Jane. Testing will be done at some other development site, so that Joe also needs to coordinate with Tim, who is responsible for planning the component tests, and Sue, the planner of the system integration tests.

Thus, Joe's planning decisions depend not only on Jane's management decision about what tool to use, but also on design decisions made during software development. On the other hand, the plan that Joe creates has an impact on Tim's and Sue's planning decisions, as well as any quality assurance planning concerned with component implementation. Figure 1 gives an overview over the wealth of dependencies occurring even in this small example scenario. Given the fact that changes are prone to occur frequently as design enhancement are released and errors are discovered in system components, these dependencies will quickly become difficult to manage without automatic support.

In this paper, we propose an approach that addresses the problems described above by tracking decisions made during project planning and plan enactment. In order to identify those dependencies relevant for a decision, we established a *Model of Planning and Plan Enactment*, which explicitly describes the activities likely to occur while planning and enacting a software development project. In order to formalize this model, we will adapt the *Redux Model of Design* [14] to record planning and management decision dependencies, as well as project enactment decisions. Furthermore, we are currently identifying heuristics to automatically capture typical dependencies, and defining rules to provide automatic planning support where possible.

In the remainder of this paper, we outline the current state of research regarding the problems described above (see section 2), summarize our proposed solution (section 3), and give a summary and short overview over remaining questions (section 4).

2. State of the Art

SW Process Modeling (PM) approaches [2], [5], [9], [12] provide mechanisms for modeling the software development process itself, i.e. the technical activities necessary to create the software project deliverables (e.g. system documentation, executable code). Some of these approaches also provide some support for modeling planning and management activities. However, none of them provide a *systematic* approach to supporting planning and management activities: while such PM approaches support plan enactment by actively providing change notifications as well as passively giving access to relevant information whenever necessary, they do not do so for planning activities.

Artificial Intelligence (AI) planning research has yielded some interesting results in the field of *Reactive* or *Mixed Initiative Planning* [13], [16], [18]. These approaches are based on a systematic model of planning and management and provide some dependency management between planning decisions. However, AI planning approaches put their emphasis on automatic planning and plan enactment. User interaction is supported, but seen as an exception rather than the rule. Thus, AI planning approaches are difficult to adapt to complex and creative processes like software engineering, where decisions depend on too many chaotic factors to be effectively automated. This is especially true for recent research concerning Agile Methods (e.g. [3]), which tend to discard rigidly defined processes in favor of entirely reactive support of software development. Furthermore, AI planning approaches concentrate on structural planning, i.e., deciding what tasks to do in what order. Some also allow task scheduling, but other planning or management tasks are not taken into account.

Conventional business planning and workflow management tools are available commercially. These tools tend to concentrate either solely on project planning, [1], [10] and do not provide enactment support, or they concentrate almost entirely on project enactment [8], [17], and provide little or no support for project planning and management. None of these tools support feeding enactment information back into the plan.
Recently, tools have been presented that provide both planning support and workflow enactment, allowing for some feedback between enactment and planning, e.g.[11]. However, none of these COTS tools provide systematic planning support, i.e. no explicit model of planning exists, and no dependency management between planning and enactment decisions is provided. What planning support commercial business tools do support (e.g. planning wizards, critical path calculation, resource management) concentrates on individual technical details rather than an overall support for the process of planning as a whole.

None of the approaches mentioned above explicitly store dependencies between decisions in a comprehensive model. This issue is covered by design rationale and decision management research [3], [14], [16]. Design rationale approaches provide representation mechanisms to store decisions and their rationales, thus providing dependency and change management. However, existing design rationale approaches track arbitrary, untyped decisions, with no notion about how these decisions interact. Modeling decisions and dependencies is entirely left to the user, and little or no support exists for automatically capturing dependencies.

To summarize, we find that existing approaches and tools lack support for project planning and management in the following areas:

1. Planning and management activities are not supported *systematically*. We maintain that an *explicit model of planning* is needed, enabling an automatic planning support system to recognize the necessity of planning and replanning activities.
2. Existing approaches do not explicitly track the process of creating a plan. We believe that *explicitly capturing* planning and management *decisions*, as well as the *dependencies* between them, is necessary for effective planning support.
3. Without an explicit planning model, existing approaches cannot *automatically* capture decision

rationales, since that would require knowledge about planning, and typical planning dependencies and rationales. *Heuristics* are needed to automatically determine rationales for frequently occurring classes of decisions and dependencies, therefore only requiring the user to state extraordinary decision rationales and dependencies.

Below, we address these issues in more detail.

3 Approach

3.1 A Model of Planning and Enactment

In modeling the activities that typically occur in a software development project, we found that there are two classes of activities, creating two interleaved, but basically separate process levels distinguished by the results the corresponding activities produce:

1. The object-level process is the domain-specific process that produces the project deliverables. In other words, the object-level process corresponds to the project plan. It models the activities done during software development as (sub-) processes, with the project deliverables as their outputs.
2. The meta-level process is the process that describes the planning activities necessary to create and maintain the object-level process. It models these activities as meta tasks, which create parts of the object-level process as their outputs (see Figure 2).

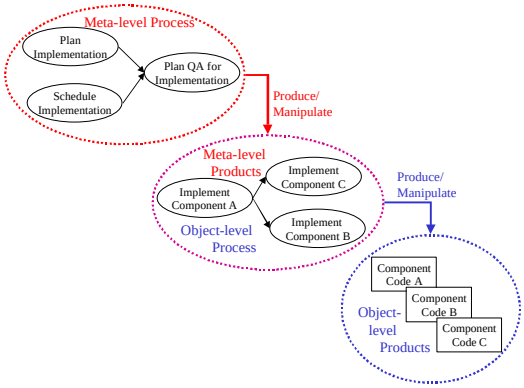

Figure 2: Meta-Level and Object-Level Entities

Accordingly, our model of planning and enactment distinguishes between *object-level entities* and *meta-level entities*, which can be manipulated by *operations*. *Decisions* represent the performance of operations, and connect the two levels, as decision *rationales* can refer to entities on either level.

Object-level entities represent data structures which can occur in a software-development process or project plan, e.g. the software development processes occurring in the project plan, development methods that can be selected to solve a process, and process parameters determining the product flow between processes.

Meta-level entities represent data structures occurring in the meta-level process, i.e. during project planning and management. Examples for such entities are tasks to schedule an object-level process, select a solution method for it, or to plan necessary quality assurance activities for a process. Outputs of such meta tasks are object level data structures (e.g. processes, method selections, product flow definitions, etc.).

Operations performed during project planning or enactment are formally represented by decisions. A decision depends on facts and other decisions. When such a decision prerequisite is invalidated, dependent decisions need to be reconsidered.

The above model is extendable both on the object level and the meta level, and new decision types and dependencies can be defined. This is necessary in order to allow the user to model additional and unforeseen tasks and dependencies to provide as extensive a support for project planning and plan enactment as possible.

In the scenario delineated above, object-level entities are for example the process of implementing the accounting system's components, and the component code produced by the implementation process. Joe's task of planning and scheduling the implementation process is a meta task, which can be solved by the operation of dividing the object-level process into subprocesses, and scheduling them. This operation will be represented as a planning decision that depends on other decisions, namely Jane's selection of a development tool, and the (object-level) decisions made by the software designers. In turn, any scheduling decisions made by Tim and Sue for the component and system integration test processes depend on Joe's schedule for the implementation process, as well as on events occurring during component implementation.

The user must be allowed to model meta-tasks manually, as the necessity to plan or re-plan certain parts of the project becomes evident. On the other hand, the burden of having to model every single plan step cannot be charged solely on the user. Therefore, we need to automatically generate meta-tasks in certain situations. For example, if the plan is inconsistent in some way (e.g. a process has not been planned or scheduled yet, or a process input is not produced anywhere in the plan) a meta-task can be generated for the user to correct this situation. Other meta-tasks can be created by generation rules, either system-inherent (e.g. a replanning task should be generated if a process is started too late to meet a deadline), or user-defined (e.g. an additional quality assurance planning task should be generated if the system design exceeds a certain complexity).

3.2 Formal Representation

As mentioned above, various representation formalisms for decisions and their dependencies have been proposed in literature. Therefore, we did not develop a new

representation mechanism to formalize our model, but adapted an existing approach to our purposes:
Petrie's *Redux Model of Design* [14] is a model of decision dependencies in engineering design projects. It tracks decisions made for design goals, captures the rationales that led to these decisions, and stores their results, i.e. assignments made to design parts, and subgoals that are still open after a decision was made for a particular goal. Decision rationales can be arbitrary facts stored in a database, or assignments supported by other decisions. Redux uses a Truth Maintenance System (TMS) [5] to represent goals and decision in a network of dependencies. Based on the state of the TMS, Redux sends notifications to the concerned agents whenever a change elsewhere in the project necessitates that they rethink their own decisions.

It has been stated before that planning and design share many similarities [16], and the concepts Redux supports (goals, decisions, assignments, etc.) easily lend themselves to representing planning tasks as well as software development processes [15]. The fact that Redux provides active notification services as well as passive "bookkeeping" of decisions made, makes it especially suitable to our purposes.

In order to maintain two separate (but interleaved) Redux goal trees for the meta-level and object-level processes, some redundancy in the data structures is necessary, since decision results on the meta level also have to be represented as data structures on the object level (see Figure 3). In order to keep the Redux representations of the two process levels consistent with each other, we had to slightly extend Redux' underlying TMS, adding rules that allow data-structures on the object-level to depend on decisions and assignments on the meta level.

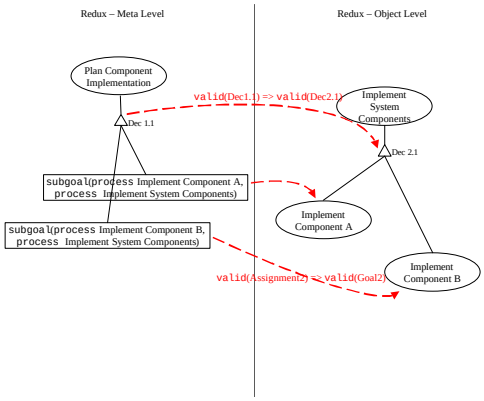

Figure 3: Interdependent Redux Goal Trees for Meta Level and Object Level

3.3 Tracing Project History

In capturing the project history, i.e. tracking the decisions made during project planning and enactment, two problems arise which need to be solved:

1. How to capture dependencies between decisions: We need to find heuristics what user behavior, i.e. which order of user actions, suggests which dependencies to define, respectively, which rationales to automatically record.
2. How to represent temporary relation between decisions, or rather, networks of decisions: A representation is needed which allows us to capture "snapshots" of decision networks at certain points in time.

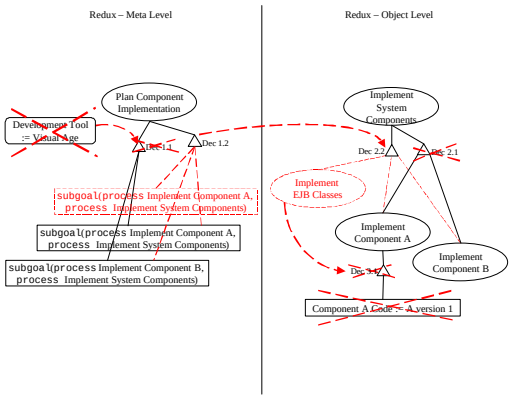

Figure 4: Temporal Dependencies between Decisions

Figure 4 shows an example for a temporal relationship between decisions and their rationales: If in the above scenario the selected development tool is unavailable for some reason, and some other tool is selected which does not contain EJB basic classes, several decisions need to be reconsidered. The meta-level decision made for the implementation plan will have to provide for the new need to implement EJB classes, and dependent object-level decisions need to be reconsidered. For example, components whose interface depends on VisualAge classes might need to be adjusted to the new requirements. Capturing "snapshots" of the situations before and after the change from one development tool to another will be useful for later analysis. Also, storing the causal relationship between the availability of the originally selected development and the decisions retracted and remade because of the tool's unavailability will allow the system to notify the planners and programmers in question if for some reason VisualAge should become available in the future.

3.4 Heuristics

Other typical dependencies between decisions made during project planning and plan enactment are for example:

- Any decision made for a meta task or object-level process depends on the currently valid assignment to the corresponding process' or task's input variables.
- A decision determining process precedence depends on that process's input variables, preconditions, and

possibly also on the resource assignments made for the process.

These are examples for dependencies that can be deduced from the context in which a decision was made. Rules (e.g. in the form of Event-Condition-Action rules, [6]) can be constructed that automatically add the corresponding dependencies to a decision if the context meets certain conditions:

Event:	DecisionMade(<task>)
Condition:	HasInputs(<task>,<inputs>)
Action:	rationale:= CreateRationale(<inputs>) CreateDecision(<task>, rationale)

Event:	PrededenceDecisionMade(<task>)
Condition:	HasPrecondition(<task>, <cond>)
Action:	rationale := CreateRationale(<cond>) CreateDecision(<task>, rationale)

The functions called in the above examples are events happening in an underlying process-centered software engineering environment (PSEE), or create data structures in the Redux dependency management system. For example, the event DecisionMade(<task>) indicates that some (object-level or meta-level) operation has been executed in the PSEE, and the function CreateDecision((<task>, <rationale>) builds a Redux decision with its dependencies which represents the operation in question.

The above examples illustrate single events that indicate a clearly defined set of dependencies to be captured. Other situations exist where a series of interrelated events need to captured in a set of interdependent decisions. An example for such a situation is the development tool change illustrated in Figure 4: The fact that EJB basic classes are no longer available will cause a number of interrelated changes to plan and enactment state; e.g. a plan change allowing for additional implementation steps, additional inputs (i.e. the new EJB classes) to existing processes, and changes to already implemented components to accommodate for the interface requirements the new EJB classes pose. All these changes will cause events in the underlying PSEE, which in turn should be captured by a tracking component, and represented as a network of interdependent decisions in Redux.

Finding a formalism to ensure

a) that interrelated events can be identified as such, and
b) that the correct dependencies are extracted from these events, and the desired responses and notifications are triggered if the initiating fact changes for a series of interrelated decisions

is still ongoing research.

In the above example, desired change notifications would be notifications to the planner and programmers involved in the original change if the original development (including the EJB classes in question) should become available again for some reason: The planner might want to remove the additional implementation processes from the plan, the additional input parameters might be removed from the processes changed after the tool became unavailable, and earlier implementations based on the original EJB interface might be reconsidered. On the other hand, implementing the new EJB classes might be so far advanced that it is decided not to make use of the VisualAge classes. In this case, the interdependent decisions made after the original change need to stay in place as a whole.

4 Summary and State of Ongoing Research

This paper outlines an approach to facilitating planning and management activities in software development projects. We define an extendable model of planning and enactment interactions that allows us to portray the object-level process of software development activities as well as the meta-level process of project planning activities. Operations performed on these data structures are captured as decisions, which can depend on any other decision on the object or meta level. The knowledge contained in this model can be utilized to automatically capture decision rationales, identify the necessity of planning and replanning activities, and notify project participants whenever changes of impact to their activities occur.

Thus, our approach aims at providing extensive support for the process of planning and enacting software development projects, in the form of dependency management, user notifications, and, where possible, automation of selective process steps.

So far, we have finished analyzing the typical activities likely to occur during project planning and management, and formally represented them as Redux goal types. Operations solving meta-tasks have also been formalized in Redux, in the form of typed decisions with standard dependencies represented as rationales to be instantiated when such a decision is made. An interface between an underlying PSEE and our dependency management system has been developed which utilizes the PSEE's events to create and maintain the corresponding Redux data structures and dependencies via ECA rules (see Figure 5).

Remaining conceptual steps to be done are:

- Identifying heuristics for capturing non-standard dependencies between decisions.
- Finding a formalism that facilitates identifying interrelated events in the underlying PSEE.
- Defining a formalism that captures such interrelated events in interdependent decisions in the dependency management system.
- Extending the ECA rule-based interface to include these non-standard dependencies and their

representation in the dependency management system.

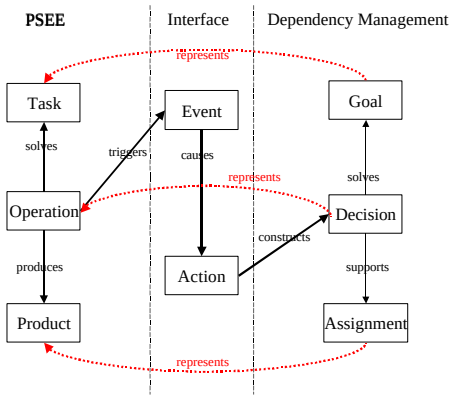

Figure 5: Interface between PSEE and Dependency Management

These concepts will be implemented in a prototype based on the PSEE MILOS [12] as its underlying PSEE, and Redux as its dependency management system. Necessary implementation steps are:

- The definition of an interface between MILOS and Redux as specified above.
- The implementation of a tracking component which captures and groups interrelated MILOS events based on the formalisms outlined above.
- The creation of a wrapper around the basic Redux system which implements decision types and their dependencies suitable to our purposes.
- The adaptation of existing Redux notifications to the needs of project planners as well as software developers. These notifications will be delivered via email.
- The implementation of a graphical user interface which provides access to the networks of dependencies captured by the tracing component. This GUI will be based on an existing HTML interface to the MILOS Project Trace component, which allows access to the trace of events happening in a project, as well as their temporal relationships.

The resulting prototype will be deployed and used in the MILOS development process, allowing us to assess the usefulness of automatically captured dependencies and resulting notifications, and if necessary adapt the heuristics implemented in the prototype.

References

[1] AutoPlan: http://www.tufan.com/autoplan/autoplan_enterprise.htm

[2] S. Bandinelli, E. Di Nitto, A. Fugetta: Policies and Mechanisms to Support Process Evolution in PSEEs. Proceedings of the 3rd International Conference on the Software Process, Reston, USA, Oct. 1994.

[3] Beck, K., Fowler, M.: Planning Extreme Programming. Addison Wesley Longman, USA 2000. ISBN 0201710919.

[4] Conklin, J. and M. Begeman: *gIBIS: A Hypertext Tool for Exploratory Policy Discussion*. Proceedings of CSCW '88 (Computer Supported Cooperative Work), September 1988.

[5] Jon Doyle, *A Truth Maintenance System*. In Journal of Artificial Intelligence, 12(3), pp 231-272, 1979.

[6] Barbara Dellen: *Change Impact Analysis Support for Software Development Processes*. Dissertation, University of Kaiserslautern, Germany, Feb. 2000.

[7] J.C. Grundy and J.G. Hosking,: *Serendipity: integrated environment support for process modelling, enactment and work coordination*. In Automated Software Engineering: Special Issue on Process Technology 5(1), January 1998, Kluwer Academic Publishers, pp. 27-60.

[8] IBM MQSeries: http://www-3.ibm.com/software/ts/mqseries/workflow/

[9] G. E. Kaiser, St. E. Dossick, W. Jiang, J. Jingshuang Yang and S. X. Ye,: *WWW-based Collaboration Environments with Distributed Tool Services*. In World Wide Web, Baltzer Science Publishers, 1999.

[10] Microsoft Project: http://www.microsoft.com/project/

[11] MicroTool InStep: http://www.formulabls.com/e/html/body_instep.htm

[12] Maurer, F., Dellen, B., Bendeck, F., Goldmann, S., Holz, H., Kötting, B., Schaaf, M.: *Merging project planning and web-enabled dynamic workflow for software development*. In IEEE Internet Computing, Special Issue on Internet-Based Workflow - May/June 2000.

[13] Muñoz-Avila, H. & Weberskirch F.: Planning for Manufacturing Workpieces by Storing, Indexing and Replaying Planning Decisions. In: Proc. 3rd Int. Conference on AI Planning Systems (AIPS-96), AAAI-Press, 1996.

[14] Charles J. Petrie: *The Redux' Server*. In Proceedings of the International Conference on Intelligent and Cooperative Information Systems (ICICIS), Rotterdam, May 1993.

[15] Petrie, C., Goldmann, S., Raquet, A.: *Agent-Based Project Management*. In Springer LNAI 1500 Journal on "Artificial Intelligence Today", Springer, 1999.

[16] Stephen T. Polyak: Austin Tate, *Rationale in Planning: Causality, Dependencies, and Decisions*. In Knowledge Engineering Review 13 (3), September 1998, pp 247-262.

[17] TIBCO InConcert: http://www.tibco.com/products/in_concert/

[18] Manuela M. Veloso: *Towards Mixed-Initiative Rationale-Supported Planning*. In A. Tate (ed.), Advanced Planning Technology, pp 277-282. AAAI Press, May 1996.