An Explanation Oriented Dialogue Approach for Solving Wicked Planning Problems

Gengshen Du
University of Calgary
2500 University Drive NW
Calgary, AB, T2N 1N4, Canada
dug@cpsc.ucalgary.ca

Michael M. Richter
TU Kaiserslautern
FB Informatik, P.O.Box 3049
67653 Kaiserslautern, Germany
richter@informatik.uni-kl.de

Guenther Ruhe
University of Calgary
2500 University Drive NW
Calgary, AB, T2N 1N4, Canada
ruhe@ucalgary.ca

Abstract

In this paper we discuss support for solving complex "wicked" planning problems by dialogues and explanations. Wicked problems are essentially imprecisely formulated problems, i.e. those that do not have a clear goal, well defined methods, and are subject to personal opinions of involved stakeholders that may be changeable. The method contains the following steps: (1) Reducing the complexity of the problem by the selecting a specific concern; (2) Obtaining a user defined ideal plan, called a prototype; (3) Comparing the actually generated plan and the prototype by a similarity measure. This will be aided by an explanation oriented dialogue. A major problem for the explanation is that we need to explain the result of an optimization procedure which excludes classical parsing oriented methods. The approach is generic and was instantiated in release planning, investment, and urban planning. We have simplified the original problems significantly in order to illustrate the principal approach.

1. Introduction

Although wicked planning problems don't have a precise definition, they have several properties [Rittel, Webber 1973] [Rittel, Webber 1984] but not all of them may occur:

- There is no definite goal of a wicked problem.
- There is no stopping rule and therefore one can always improve the solution.
- Solutions are not true-or-false but good-or-bad.
- The problem is not static but changes dynamically.
- The view on problems and their solutions is subjective and context dependent.
- One has different participants (called stakeholders) with different preferences and this makes it problematic to judge the quality of the solution.

Wicked is not the same as complex. For instance, chess is complex but not wicked because there is a clear goal, precisely defined moves etc. Complex problems can usually be solved (at least theoretically) by computer support.

Solving wicked problems calls for deep human insight into these problems. On the other hand, the complexity of the problems we consider necessitate tool support. As a result, we get a situation where human and software agents must cooperate. For a useful cooperation among agents they need to communicate.

In this paper we consider an interactive and explanation supported approach to planning problems that are both wicked and complex. The involved software agents are mostly optimization procedures.

In our approach we concentrate on the communication between the different agents. There are two types of communication: between humans and between humans and software agents. For both types of communication, computer support is useful.

In the first case – communication between humans – mainly involves protocols, knowledge support and guidelines.

In the second case – between humans and software agents – communication is supported by an additional software agent who has insight into the activities of both humans and software agent participating in the communication. Although we will also comment on the first communication type our main concern is on the second one.

A main problem with this communication is that humans can think intuitively while software agents need exact rules. The support is organized in the form of a dialogue between the agents that has an explanatory character.

This explanation method essentially differs from the methods used in traditional expert systems. During the dialogue stakeholder opinions can be changed or withdrawn, even if they are formulated as hard constraints (i.e. the constraints that traditionally cannot change).

For the second type of communication this method has a generic character and can, in principle, be applied to many other wicked and complex planning problems. We will

illustrate the methods in applications from very different areas: release planning, investment planning, and urban planning. The last application is much more involved than the other two, contains communications of both types, and computer support is less developed. Nevertheless, all of these applications share several essential properties. In particular, there is a special stakeholder who is responsible for the final decision; this is the leading project manager or the leading architect; we simply call this person the manager.

In summary, the scenario has the following participants:

- The stakeholders
- A particular stakeholder, the manager
- One or more software agents called planners
- If a human deals with a software agent we also refer to this human as the user.

The user is presented with the discrepancy between ideal solutions (from a certain point of view) and the actual solution produced by planners. The discrepancy is formulated using similarity measures. This allows a problem reduction so that stakeholders can get an overview of the situation and change their opinions.

In our approach presented later, we consider the discussion between the manager and one particular stakeholder. This is a necessary step that simplifies the procedure but can only be regarded as a first step towards the overall solution. This solution requires an integration of the individual views and preferences, but is not addressed in this paper.

2. General Problem Description

Because of the nature of wicked planning problems, their modeling is an important but difficult issue.

In order to get computer support for modeling one has to map all concepts, relations and intentions of the real world situation to a mathematical model. Specifically:

Intuitive utility and preferences → Model parameters

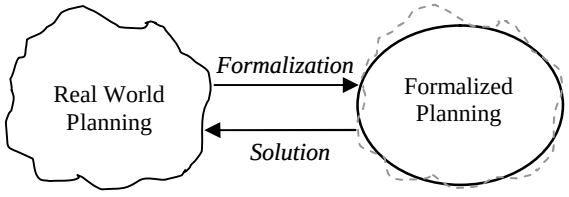

Figure 1: Mapping between real world situation and formalized situation

This mapping is based on the hope that the planner will generate a useful or at least acceptable plan. To some degree this is only a guess because if the planning problem contains the difficulties described above one cannot anticipate in which way the planner will use the parameter values in order to generate the plan. In addition, certain real world aspects may not be covered at all.

In such a situation a major problem arises. The user often has no insight into the major reasons that lead to the presented solution. In particular, if the solution is somewhat surprising she/he has difficulties accepting it. Therefore the user may not trust and may not accept the solution.

Because of the lack of understanding, the feedback from the user is mainly twofold:

- Asking for more (detailed) information
- Objecting to (parts of) the plan.

The support needed here involves bridging the gap between the human and the software agent. A better understanding between cooperating partners always requires one to explain motivations and results to the other. We concentrate on explaining the results of a software agent to a human. This cannot usually be done in one step and therefore we apply a dialogue approach.

3. Interactive Planning, Communication and Dialogues

The simplest planning strategy, and the one applied in Artificial Intelligence in the past, is the sequential one. This means that there is a strict order: First the requirements are collected, then the planning itself takes place, and at the end the plan is executed. For many applications, in particular for wicked problems, this strategy is obviously not applicable. One has to cope with incomplete, imprecise and sometimes incorrect requirements and has to start with planning and partial execution on such a basis. In addition, correcting requirements is often only possible if the plan is partially executed. Because execution is expensive it is often replaced by simulation.

Because of the interleaving of the activities in interactive planning the generation of a plan is not the end of planning, it is rather considered a step in the overall planning process. A plan may be changed or completed and at the same time the requirements in the plan may also undergo the same treatment.

In addition, it is often useful not to present a single plan but to offer several ones [Ruhe, Ngo-The 2004].

This whole process is a joint effort of the participating agents. As mentioned above, it requires communication between agents who exchange information and explain to each other their motivations and decisions.

An important point in any explanation is that it has to be

simple enough to provide a general overview, otherwise the information may not be understood. For this purpose we employ techniques that reduce the complexity of problem formulations.

4. Background of Explanations

4.1 General

Explanation is studied in many disciplines such as philosophy, artificial intelligence, teaching etc. The general background of the explanation methodology has its roots in these areas; we will not discuss this here, however.

If different agents participate in a scenario, those agents need to communicate with each other. To large degree this means that they explain their decisions to each other. Such an explanation has two major parts:

- Information to other agents about facts
- Information to other agents about opinions and motivations

The communication can take place between humans and between humans and software agents.

In this paper we concentrate mainly on situations where a result produced by a software agent is explained to a human. Explanations of activities by software agents play an essential role in interactive processes. An overview is given in [Wooley 1998].

In the explanation scenario there are three participants:

- The system or a system agent that provides the plan or decision etc.
- One or more users who are the addressees of the explanation.
- The explainer who presents the explanation to the user(s). The user applies the explanation in some way and the explainer is interested in the way how this is done. With this application a utility is connected, either for the user or for the explainer.

In our applications where the results of a software agent have to be explained the explainer is itself a software agent. This agent is based on the computational elements presented in the section on formalizations.

In principle, and in particular in a dialogue, explanations are always answers to questions, whether they are raised or not. The fundamental approach for the logic of question and answer is given in [Belnap, Steel 1976].

The approach of Case Based Reasoning (CBR, see e.g. [Lenz et a. 1998] also has some explanatory character built in. The central concepts are the similarity measure and the nearest neighbor notion. There is no explanation about the nearest neighbor itself, but rather an explanation as to why an object was chosen as a nearest neighbor. There are, however, some restrictions on what makes a convincing explanation. The major ones are:

- There should only be few attributes of high importance.
- The nearest neighbor selection should essentially rely on those attributes.

This allows one to reduce the complexity in problem formulations and to draw the attention of the user to a few but essential aspects.

Our main concern is explanations in the context of problem solving and decision making, in particular to the "Why" and "Why not" questions of the form:

Q = Why did you do/not do X?

We introduce a new classification of explanation that is intuitively understandable and also useful for implementation issues. When dealing with wicked problems we introduce two methods below that may be combined.

4.2 Backward Explanations

Backward explanations refer to something that happened in the past. A (backward) answer to Q is: Because you forced me to do/not do X. This can be formulated in form of a constraint or rule. The purpose of backward explanation is to increase the acceptance of a solution.

Besides this cognitive science view there is a pragmatic computer science view. This type of explanation was the traditional approach when dealing with knowledge based systems using declarative programming languages. In such systems constraints and rules have been stated explicitly; a condition was that the search for them was sufficiently efficient. In many classical knowledge based systems this was the case and backward explanations were made popular by following the parsing paradigm.

This approach was, however, not possible for procedural programs because the rules and constraints were implicit and hidden in the program. As a consequence, explanation components for procedural programs as optimization procedures were almost nonexistent.

4.3 Forward explanations

Forward explanations look into the future. The (forward) answer to a "why" question Q is: If I would have done/not done X then the following unwanted consequence Y would have occurred. The forward explanation can in certain situations also increase the acceptance of decision or solution.

In addition, it can be used to improve and complete the solution itself. The reason is that such explanations can be employed during the interactive problem formulation and solution process. In this respect it is closely related to the "What – If" analysis technique that is guided here by the stakeholder.

A technical advantage is that such explanations are not restricted to any kind of programming language or style. In particular, they can be used for optimization procedures.

A general restriction for both types of explanations is that they have to be simple and easy to understand, sometimes even at the cost of total correctness. The simplification is therefore an activity that applies to both types of explanations.

Because computer support for wicked problems uses a variety of programs, backward as well as forward explanations may be useful and applicable. In our applications we have emphasized forward explanations, but backward explanations will also occur.

5. A View on Constraints

In some respects wicked problems can be viewed as Constraint Satisfaction Problems (CSP, see [Tsang 1993]). Traditionally, one distinguishes between hard and weak constraints and the hard ones were not up to change while the weak ones gave rise to optimization problems so can be changed.

In situations where stakeholders are not sure about their opinions and may revise them, another distinction is useful:

- Factual constraints are constraints that cannot be weakened or withdrawn. They arise from logic, mathematics, physics, moral laws or anything that never can be revised under any circumstances.
- Normative constraints are the result of decisions by certain persons or organizations. Despite the fact that they are stated as being "hard" or "weak" they can be changed by withdrawing previous decisions and making new ones.

In wicked problem solving normative constraints and their underlying decisions rarely contain value in themselves. They are rather stated in the hope that they have a positive influence on the problem solution and the real utilities are maximally satisfied.

This is closely related to the difference between values and goals. Values are the aspects in which one is really interested and goals are the aspects that are subject to optimization. These two are usually not the same. What makes the problems wicked is that the values are difficult to grasp. In [Keeney 1992] deep insights into this area were given but the practical solutions provided were more of an art than the results of a systematic development.

A particular kind of normative constraint occurs when the user wants to change a solution. This is done by fixing the value of a solution parameter and this change should be formulated as a hard (but of course normative) constraint. It should be mentioned, however, that not all constraint solvers accept such kind of input.

Constraints are partially ordered by logical deduction. This allows the removal of some factual constraints by going up the hierarchy until one finds a normative constraint where the responsible agent is willing to withdraw it.

The role of normative constraints in solving wicked problems is that a (forward) explanation may convince a stakeholder to withdraw it because certain unwanted consequences become visible.

6. The Generic Approach

6.1 Overview

We consider a wicked planning problem and look for a solution that is acceptable to a set of given stakeholders. In such problems there may be one or more software systems involved; here we restrict ourselves on one system at a time, called the *planner*. We also restrict ourselves to one stakeholder at a time. In order to cover more planners and stakeholders one has to repeat the approach. Next we mention the applications for which our approach is intended to cover in a generic manner.

For the planner we have on the following assumptions:

- The planner can accept modifications of the solution as an input.
- The planner presents several solution alternatives.
- The system is stable in the sense that small modifications of the input result in small modifications of the result.
- For each plan, changing the consequences under the hard factual constraints can be independently computed, and for each such consequence the responsible constraints can be named (this is a partial backward explanation).

We assume that some stakeholder (or the manager who takes his/her position) discusses a presented plan with the system via dialogues. This stakeholder (the user) is assumed to prefer certain plans, possibly from different perspectives. The collection of these plans that the stakeholder prefers is not clearly defined and hence is not a set in the classical sense. It is rather a category (see [Lackoff 1987]) that has no membership function but certain ideal elements that are called prototypes. We call

the perspective of some stakeholder a concern C and denote a prototype with respect to C by prot(C). In an abstract setting the concerns form a set. For the stakeholders we assume:

- A stakeholder is able to define a prototype prot(C) for each concern C.

In order to discuss a solution with a stakeholder the actual solution should be compared with the prototype of the stakeholder. If the solution looks fairly good then no objection will take place, otherwise the stakeholder will probably oppose the solution. In order to measure degrees of coincidence and distance between plans we use similarity measures (see [Burkhard, Richter 2000]).

The generic approach proposed in this paper is called EXPLAIN-DIALOGUE. This system is the explainer in the explanation scenario. It carries out a dialogue between a system (planner) and one stakeholder, as mentioned in the introduction. The main steps of the approach are:

- Step 1: Generate of one or more solution alternatives from the planner.
- Step 2: The stakeholders formulate the values of some input parameters to the planner. We call the set of these values a vote of the stakeholders. The voting is done for each concern C.
- Step 3: Transform the votes into prototypes prot(C). Here we assume that this can easily be done (Step 3.1). Select one of the best solutions from the solution alternatives generated in Step 1. If there is only one solution generated by the planner, it is already the best one (Step 3.2).
- Step 4: Compare the prototype for C and the best solution chosen in Step 3.2 by a similarity measure sim_C. In order to simplify the situation the system shows a reduced form of the best plan to the user that just contains the attributes that are relevant, and where the prototype and the actual plan differ significantly.
- Step 5: Allow the user to propose a minimum number of possible changes in the solution and to remove or weaken normative constraints which are the decisions made by humans.
- Step 6: Show the consequences of the changes that are stated explicitly under the hard factual constraints and the participating constraints.
- Step 7: Generate one or more solutions and again choose one among them. Go to step 5 if no feasible plan can be generated. Otherwise, repeat the process. The presentation of the new plan and the reduced comparison with the old plan and the prototype is a forward explanation. The user may now react in three different ways: (1) prefer the new plan or (2) still accept the old plan because the reasons are now understood better or (3) choose another concern and iterate the process.

If desired, the process can be repeated with another concern.

The complexity reduction in this approach is performed in two ways:

- By focusing on one concern at a time. This can be repeated but only one concern is handled at a time (step 2).
- Simplifying the comparison of plans by presenting only the relevant attributes (step 4).

In addition, the use of qualitative descriptions also contributes to a better understanding and focusing the attention of the user to a specific but important view (the concern).

In particular, the forward explanation is shown in a reduced way by showing major qualitative consequences of proposed plan changes.

6.2 Formalization

In this section we will present a formal basis of the dialogue approach. It is the basis of the explainer, a software agent.

First we introduce the dialogue approach, without going in too many details (see Section 7.1). The user interface of the dialogue is standard. The user is presented with a template with questions or answers where the user can enter information. Examples are shown in Section 7. From the implementation point of view the user interactions are events. Events are handled in the standard way using Event-Condition-Action (ECA) rules (see e.g. [Dellen, Maurer 1998]).

We first introduce the major functions and predicates that are invoked in the dialogue.

If many applications are to be covered, it is important that only a few undefined and application dependent terms occur while the remaining dependent terms have application independent definitions.

For simplicity, we model plans as an attribute-value representation, hence a plan is represented as:

$$plan = (p_i \mid i \in I)$$

and PLANS is the set of plans. Next we assume similarity measures that compare plans:

$$sim_C: PLANS \times PLANS \rightarrow [0,1]$$

The similarity measures are, in general, concern-dependent. They are formulated as weighted sums with a weight vector:

$$w = (w_1, ..., w_n)$$

of non-negative coefficients and local similarity measures $sim_{C,i}$; i.e. we take the most common form of similarity measures (see [Burkhard, Richter 2000]):

$$sim_C((a_1, ..., a_n), (b_1, ..., b_n)) = \sum (w_i \times sim_{C,i}(a_i, b_i) \mid 1 \le i \le n)$$

where a_i and b_i $(1 \le i \le n)$ represent the given plans.

The following similarity measure functions are essential for the dialogue approach.

First we introduce concepts related to the similarity of the actual plan and the prototype. These two plans may differ significantly and may violate the interests of the stakeholder. We call this prototype is in danger. The comparison can take place in a numerical as well as in a qualitative way:

- DangerDegree(C, plan) = $1 - sim_C$ (plan, prot(C))
- QualitativeDanger(C, plan) is introduced by taking three qualitative regions $0 < \alpha < \beta < 1$ in which the function takes the values low, medium and high, respectively.
- The predicate $Danger_C$ is defined by:

$$Danger_C(plan) \leftrightarrow QualitativeDanger(C, plan) = high$$

Next the user is shown a reduced view on the danger that still contains the essential elements.

- The reduced comparison of two plans plan1 and plan2 is:

$$compare_{red}(plan1, plan2, sim) = ((plan1_i, plan2_i) | i \in I, sim_{C,i}(plan1_i, plan2_i)) < s, w_i > t))$$

- Given a concern C the reduced representation of a plan p is obtained by comparison with the prototype:

$$plan_{red}(p, C) = (pi \mid i \in I, sim_{C,i}(p_i, prot(C)_i) < s, w_i > t))$$

The thresholds s and t are user determined. This means the reduced plan shows those attributes that are the major reasons for the deviation from the prototype.

- The function $compare_{red}$(plan1, plan2, sim) takes two plans, computes their similarities and produces the reduced comparison.

Finally the consequences and the hard factual constraints are computed. These are the absolutely necessary consequences, independent of any opinions or optimizations

- The function computeConsequences( ) takes as input the values of some (changed) solution variables and the hard factual constraints. It generates the logical consequences of fixed values for the variables under the constraints.
- The function involvedConstraints(changes) are those explicitly formulated constraints that were involved in the computation of computeConsequences( ).

7. Applications

7.1 Release Planning

This example is relatively less "wicked" than other wicked problems; in particular, not all properties of wicked problems are present. This is due to the fact that the number of stakeholders as well as the number of technical constraints is quite limited. Nevertheless, release planning presents a problem that currently is not solved, overall, in a satisfactory way. Quite a number of partial problems can be formalized but creative aspects remain. Complex optimization procedures are applied but it is not clear how to formulate the input and to see a priori what the significance of the result is. It is typical (also for other problems like investment planning, see below) that formalisms lack the ability to completely encode the equivalent of the analytical mind with which the human decision-maker evaluates problem situations.

Release planning is conducted in early stages of software development to generate release plans. A release plan assigns requirements (the tasks) $\{R\} = \{R_1, \ldots, R_n\}$ to be developed for a (large) software product to release options $k \in \{1,2,\ldots K\}$, and an options K+1 if the requirement is postponed.

$$Requirements \rightarrow Release\ options$$

The releases are usually executed in a temporal order. Here we assume K =2.

A release contains a maximum number of requirements so that the total effort of the segment fit into the effort constraints allowed for a release, all the technology constraints (see below) are met, and the stakeholders' satisfaction is maximized according the objective function of the planner (see [Ruhe, Ngo-The 2004]).

Release planning is a problem that has many of the previously mentioned properties of the wicked planning problems. Specifically, its difficulties are [Ruhe, Ngo-The 2004]:

- Many aspects, even the objectives, are not stated explicitly and precisely in a formal way and are, in addition, context dependent. Hence the planner may not be a true image of the real world problem.
- There are different stakeholders who have diverging and unclearly formulated interests. The customers have a complex cost structure in their mind (consisting of direct and many types of indirect costs) that are difficult to map onto the input parameters of the planner.
- The complex interactions between the constraints are

difficult to understand.

- The demands of the customers are often in conflict with each other and the demands of the manager.
- There are uncertain estimates concerning effort (this is a standard problem in software development).

We consider two types of stakeholders. In reality there are many more:

- The customers: They do not know each other and have no communication.
- Members of the company, in particular the manager: As mentioned in the introduction, the manager knows the customers and communicates with them.

In this paper, we refer to the web-based decision support system ReleasePlanner® (http://www.releaseplanner.com). It is based on the hybrid intelligence approach proposed by [Ruhe, Ngo-The 2004]. The overall architecture of this approach, called EVOLVE*, is designed as an iterative and evolutionary procedure mediating between the real world problem of software release planning and the available tools of computational intelligence (optimization).

The planner uses an objective function that has to be maximized. The optimization itself takes care of the weak constraints. The objective function and the optimization are dealing with the interests of the company and on preferences of the stakeholders expressed in terms of votes (see below). These different aspects give rise to different objective functions that are integrated by the planner. To address the inherent uncertainty, the system is offering a set of qualified alternative solutions. These solutions form the starting point for the dialogue with the user of the system.

For our purposes, namely to illustrate the dialogue component, we have simplified the voting and do not go into further technical aspects of the planner.

The planner considers the following constraints [Ruhe, Ngo-The 2004] which are formulated explicitly:

- Precedence constraints between requirements R_i and R_j that specify R_i must be implemented before R_j (hard, factual).
- Coupling constraints between requirements R_i and R_j that specify that Ri and Rj must be implemented in the same release (hard, factual).
- Resource constraints that indicate the available capacity for each release and the needed capacity for each requirement (weak).
- Pre-assignments that fix the release of a requirement (hard, normative).

The resource constraints are hard factual. A plan is called feasible if all the hard constraints are satisfied.

Pre-assignments are hard but normative constraints. They are employed in plan changes. Only the manager can perform pre-assignments, but they can be demanded by customers.

The other stakeholders are customers that do not communicate with each other (usually they do not even know each other). The only communication that takes place is between customers and the manager. Customers do not exchange information but they may object to a given plan and express this in terms of demands for pre-assignments. These demands may very well contradict earlier votes.

In addition, other weak constraints are supposed to reflect opinions and preferences of different stakeholders involved in voting.

In what follows, we will instantiate the approach described in Section 6 for the case of software release planning.

We assume that ReleasePlanner® has generated a set of five alternative release plans summarized in the set PLANS. PLANS is represented in the following form. Each requirement R_i ($1 \leq i \leq n$, n is the total number of requirements) gives rise to some attribute that assigns a release $rel(R_i)$ to R_i. Therefore a plan P from PLANS is represented to the customer as a vector:

$$P = (rel(R_1), \ldots, rel(R_n))$$

Additional attributes for the manager depend on the specific situation. An example attribute is resource, i.e. how to balance resource consumptions Res_i over K releases (here K=2).

The concerns are expressed as a voting of the stakeholders or aspects of the company management like balancing of resources. An example concern in release planning may be "balance of resource Res_1". That means that ideally there should be a relatively stable level of consuming resources for the K releases. So far, the optimization done for ReleasePlanner® focuses mainly on maximizing stakeholder satisfaction. However, less attention is put on balancing of resources. Therefore, here in this instantiated example we take the concern as "balance of resources" and this, with the explanation component, provides added value to the current release planning.

For the releases we assume a temporal ordering and we also assume that stakeholders prefer in general earlier releases.

The votes of the stakeholders indicate the priority of requirements which a stakeholder wants to see released as early as possible. ReleasePlanner® offers many other possible stakeholder voting options, such as value-based or urgency-based voting. For a more detailed description see [Ruhe, Ngo-The 2004]. One typical format of votes by one stakeholder is of the following form:

Requirement	R_1	R_2	…	R_n
Priority	7	8	…	5

Table 1: Stakeholder votes

With "balance of resources" as the concern, a best plan P_{best} with the best resource balance is chosen from PLANS set {S}. The following table shows the resource Res_1's consumption in each release of P_{best}.

Release	Release 1	Release 2
Res_1 Consumption in P_{best}	87.2%	73.6%

Table 2: Res_1 Consumption in P_{best}

A greedy algorithm is used to generate a prototype, based on the concerns and the votes of one stakeholder. A prototype for this concern is a plan in which the balance is ideal. Particularly, there are two ways to use the votes to come up with a prototype plan (this can also be used for investment planning mentioned in Section 7.2):

- Rank requirements according to one stakeholder's votes: the higher the vote is for a requirement, the earlier the release it should be put in.
- Rank requirements according to combined votes from more than one stakeholder: similarly, the higher the combined vote is for a requirement, the earlier release it should be put in. However, different stakeholders have different votes and therefore different similarity measures on the prototype.

No matter which of the above two ways is used, before assigning a requirement to a feature, feasibility in terms of available resources is checked.

For the similarity measure sim_C, used to compare P_{best} and the prototype, we take a very simple approach:

- The local measures have only the values 1 (in case of equality) and 0 otherwise.
- The weights correspond to the priority to develop a requirement in Release 1. As mentioned above, earlier releases are always preferred over later ones.

The stakeholder may now object to the plan. This is done by demanding certain new pre-assignments in ReleasePlanner®. The number of new pre-assignments should be as low as possible because the results generated by ReleasePlanner® are near optimal. These near optimal plans may be in conflict with a stakeholder's votes and indirectly the stakeholders wish to revise them. For this purpose the stakeholders are shown a template of the following form:

Name requirements to move to another release
Move requirement R_1 to release 1

Table 3: Identify changes

This corresponds to the question Q: Why is R not in release 1? For the answer several computations have to be performed. The answer will be presented to the customer in the form:

The following requirements are connected with R_1 by: Precedent constraints: (R_3, R_1), (R_4,R_1) Coupling constraints: R_5
In addition to R, the planner has moved: R_6, R_7, R_8
The planner has moved out of release 1 the following requirements despite high votes for them: R_2

Table 4: Consequences of the changes

Upon the request of new requirement pre-assignment, a set of new feasible plans are generated by ReleasePlanner® and P_{best}' is chosen as the best plan in terms of the concern.

Finally, the result is summarized in the following diagram where the plans are compared:

Release	Release 1	Release 2
Res_1 Consumption in P_{best}	87.2%	73.6%
Res_1 Consumption in P_{best}'	92.2%	83.1%

Table 5: Comparison of two plans in terms of the concern

The following table is also provided to the user to show the differences in the assignment of requirements between P_{best} and P_{best}'.

Requirement	R_1	R_2	…	R_n
Priority	7	8	…	5
Desired Release in Prototype	1	1	…	Postponed
Actual Release in P_{best}	2	1	…	Release 2
Actual Release in P_{best}'	1	2	…	Postponed

Table 6: Comparison of two plans in terms of requirement assignment

As a summary for the above two tables, they not only show the improvement of P_{best} in terms of the chosen concern, but also highlight the consequential changes in terms of requirement assignments to releases.

The user sees, however, only that part of the diagram where differences between attributes of high importance occur. These are forward explanations for the original decision of the planner, P_{best}. The user can now react in different ways:

- Accept the new plan

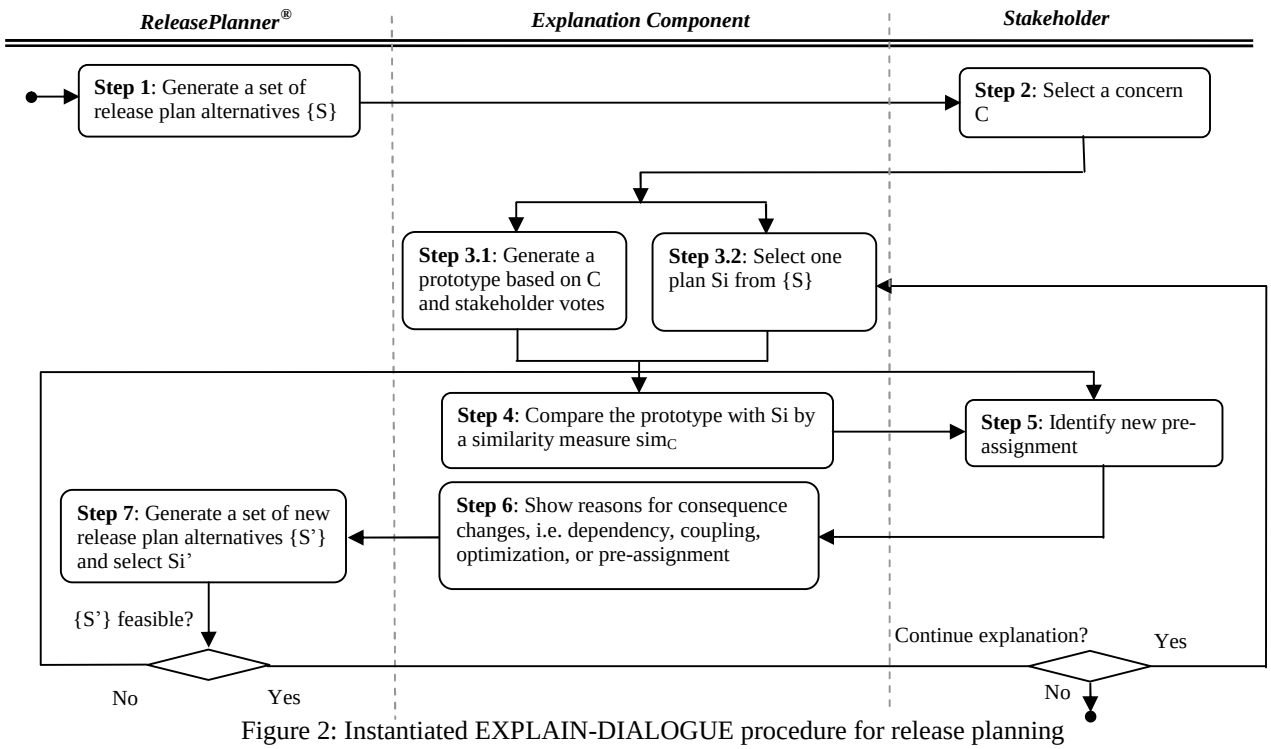

Figure 2: Instantiated EXPLAIN-DIALOGUE procedure for release planning

- Stay with the old plan
- Select a new concern and iterating the procedure

In the first two situations, some normative constraints have to be revised.

This view on release planning can be refined in many different ways in order to have a more detailed view on the preferences of the stakeholders. For example, the value or urgency of requirements can be considered directly; here we have summarized them in terms of a single vote. Similar arguments apply for constraints concerning the internal preferences of the company.

Figure 2 shows the instantiated EXPLAIN-DIALOGUE procedure within the context of release planning.

This approach exclusively looks at one specific stakeholder without considering the impact to other stakeholders. This more general approach would need additional negotiation components to find a solution that all involved stakeholders agree on [Denzinger, Ruhe 2004].

7.2 Investment

7.2.1 Investment Planning as a Wicked Problem

In investment planning [Bodie et al. 1993] the task is to invest resources into projects:

Resources $\rightarrow$ Projects

There are many factual constraints, e.g. one investment can necessitate others. In addition, there are usually many personal opinions of the participating stakeholders. The economical purpose of investment is ultimately to increase profit (ROI, Return Of Investment). This is the difference between benefit and costs where one distinguishes the following aspects:

- Predictable consequences
- Consequences that can be estimated
- Unpredictable and unforeseen consequences.

The two latter aspects can only be evaluated in the future, and give rise to various kinds of risks. A particularly difficult aspect is concerned with the strategic influence of the investment, and here is a point where stakeholders often have quite different opinions.

The benefits are almost always in the future and are hard to predict. But also costs are often invisible. We will restrict the discussion on costs.

Investments can be classified into different types. A popular classification is:

- New business (completely new company or new branch)
- Investment in extensions (more capacity, avoiding outsourcing)
- Investment for change (replacing old equipment by

new ones that are more economical). An example is investment into the IT-structure of the company.

- Investment in ecology
- Long range investment (diversification in order to be equipped for changing demands, to get closer to the market (e.g. in foreign countries), etc. There are direct and indirect costs involved, and all of these aspects give rise to different kinds of constraints to be reflected in the voting.

Not only uncertainties, but also different opinions and interests of the participating stakeholders are involved here. Often the uncertainties are modeled as probabilities, but it should be mentioned that they are not based on a model. Rather, they are subjective probabilities and therefore depend on the person, see [Fishburn 1986].

There are many kinds of constraints. Three important ones are:

- Coupling constraints: Two investments are only possible if done together.
- Consequence constraints: One investment necessitates another one (possibly at a later time).
- Predecessor constraints: One investment necessitates that another investment was done earlier.

The difficulty that arises here is that the second involved investment may be invisible at a later time and may be difficult to predict. This may result in a change of the plan.

There are different approaches to describe cost. A simple example was given by the Gartner group, see [Emigh 1999]. There the term "Total Cost of Ownership" was introduced and described for investment in IT- business:

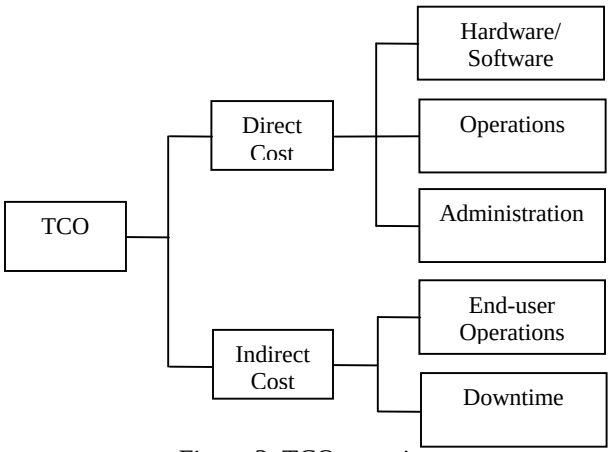

Figure 3: TCO overview

The indirect costs are often invisible and there are different opinions about their size. Experimentation with the TCO model shows that in IT investment they cover on the average 50% of the costs, and are often much higher.

In practice there exists quite a number of optimization algorithms for investment planning. Here we do not refer to a specific one in detail but will rather mention a few ones in general:

- The Black/Scholes model [Hommel, Pritsch 1999]. This algorithm considers buying and selling as options and has various input parameters to statistical computations.
- Various tools to support the TCO model.
- The Balanced Scorecard tool [Kaplan, Norton 1993]. This algorithm combines financial measures with non-financial measures, such as customer satisfaction, internal processes and innovation. An important property of the scoreboard is that it can be extended at a later time.

As long as financial computations with exact input are concerned all of these tools are precise, but cover only part of the situation. The non-financial considerations are based on subjective input and can be revised. They are not only imprecise but also depend on the interests of the stakeholder. This gives rise to discussions among stakeholders and also to discussions between humans and software agents representing the tools used. For this purpose we suggest another instance of the generic explanation method.

7.2.2 Investment Planning as an Instance of the Generic Approach

In an abstract view, release planning can be seen as the problem of distributing objects (the requirements) to boxes (the releases) under a number of constraints.

$$\text{Objects} \rightarrow \text{Boxes}$$

Some stakeholders have preferences to put certain objects in specific boxes while other stakeholders are concerned with the internal management of the boxes that are assumed to be of limited capacity. The real motivations for this distribution may not be clear and easy to formulate.

In a simple setting of investment planning we can assume that there are only financial resources in the form of monetary units, each unit can be used for every project and the set of potential projects for investment is fixed. This means that the monetary units are the objects and the projects are the boxes.

Such a view opens the possibilities for applications that are closely related. First, there are two ways of optimization, depending on where the shortage is:

- Type 1: There is a limited capacity in the boxes.
- Type 2: There are a limited number of objects to distribute.

In both cases one wants to put as many objects into the boxes, respecting the demands of the stakeholders.

There are two major differences between release planning and investment planning that play a role in our considerations.

- Release planning is of Type 1, a box can only contain a limited amount of requirements, while this is not the case in investment planning.
- In investment planning there is no natural ordering of the boxes (like a temporal ordering), i.e. the projects, while release planning has an ordering.

Because the shortages are just the opposite in release planning it is suitable to reverse the roles of attributes and values. Hence the attributes are of the form project(investment), where investment denotes a number of monetary units.

Given are n projects $\{project_i\}$ $(1 \leq i \leq n)$ and a number of N monetary units. An investment plan is represented in this way as a vector:

$$plan = (project_i(investment_i) \mid 1 \leq i \leq n)$$

In analogy to Section 7.1 a plan is called feasible if the sum of all investments is no more than N. In our approach only feasible plans are considered. The prototypes of stakeholders are just their votes; they describe simply how much money they want to invest in various projects.

$$vote = (project_i(investment_i, w_i) \mid 1 \leq i \leq n)$$

where w_i is a an integer between 1 and 10.

The corresponding prototype is:

$$prot(project_i(investment_i) \mid 1 \leq i \leq n)$$

The similarity measures are asymmetric. The first argument is always the query that is demanded by a stakeholder, i.e. the prototype. The local measures are of the form:

- $sim_{C,i}(project_i(investment_1), project_i(investment_2)) = 1$, if $(investment_1) \leq (investment_2)$, and
- $sim_{C,i}(project_i(investment_1), project_i(investment_2)) = (investment_2)/(investment_1)$, if $(investment_1) \geq (investment_2)$, i.e. higher investment is always better in the view of the stakeholder.

For the global similarity the votes are just the weights. In our simplified approach the voting is very easy because the stakeholders can directly name the weight in the form of the importance of the project.

The remaining parts of the formalism can be instantiated as for release planning.

7.3 Urban Planning

Urban planning belongs to the earliest types of wicked planning problems, and made this term popular (see [Rittel, Webber 1973, 1984]). The use of a dialogue component arose from discussions with Karsten Droste from the ETH Zurich. We present mainly perspectives for applications here. There are several differences between urban planning and release planning as well as investment planning which we will shortly mention.

In urban planning many stakeholders are participating who all know or can know each other. There are three major types of stakeholders:

Figure 4: Some stakeholders in an urban planning project [Christiansen et al. 2006] (City of London)

- Active stakeholders that perform urban planning (usually architects and technicians, civil engineers).
- Active stakeholders that have to be asked about their opinions (like local authorities, railway companies, churches, business organizations etc.).
- Passive stakeholders who watch the planning process. These are usually citizens. However, citizens may switch to the group of active stakeholders.

Figure 4 gives an impression of major stakeholders in an urban planning problem in the city of London.

All of these stakeholders communicate with each other. Technically, the best way to communicate is via a web portal. For an example see [Hillenbrand, Reuther 2003]; we will not discuss this here. In human conversations and documents many factual constraints occur such as legal arguments. If properly recorded, backward explanations are quite helpful in this situation.

The stakeholders usually have clear opinions on what they ultimately want, but it is not easy to formulate this directly in terms of votes. They can usually formulate an ideal prototype in terms of the resulting system that can be the input to an optimization procedure as a hard constraint, where the distance from the actual plan is measured by a similarity measure. In practice, this is actually done but only manually.

Presently the dialogue among humans in this application is done with very little computer support. . The dialogue component must have more knowledge of different kinds of constraints than in release planning because it is not possible that all the stakeholders provide a vote in the beginning of the dialogue.

Another major difference to release planning is that there is (presently) no universal software agent that acts as a planner and optimizer. Instead, there are usually many software agents and each of them causes the same problems as a single planner. Some of the problems that need software agents are:

- Classical planning aspects like planning resources or establishing schedules: This is, in principle, an optimization problem as in release planning, although more complex because specification, planning and execution are more interleaved. Urban planners make heavy use of optimization algorithms. Here interactive planning is used, in particular the technique of "What-If" analysis, which is a rudimentary form of forward explanation.
- Shortest paths: Planning should observe that citizens can reach supermarkets, schools, offices etc. in the shortest possible time. This is again an optimization problem.
- Waste optimization: Simulation is used in [Baetz 1990].
- The shade problem: High buildings throw shade on neighboring buildings. The shade problem involves minimizing this kind of shade. However, there are other constraints when buildings are moved, e.g. those arising from railway organizations or churches [Christiansen et al. 2006].

The present computer support for the man – machine communication is quite limited, despite the fact that many optimization algorithms exist. The only advanced support is done by offering simulations.

A twofold support is needed:

- Support communication among humans. A first step was done in [Hillenbrand, Reuther 2003] where a web portal for urban planning was introduced. Presently an extension to more powerful support is going on.
- Support the human-machine interaction, e.g. with a dialogue system as proposed here. This is not yet done but urban planners have seen the need for it.

8. Implementation and Evaluation

In [Du 2004], the formal approach was preliminarily implemented for release planning using ReleasePlanner® as a release planning tool. It essentially uses (modifications of) the templates mentioned in Section 7.1 and has, in particular, Table 6 as the final template.

A real statistical valid evaluation is in progress. However, the approach was qualitatively tested in a graduate course at the University of Calgary. In this course the students (6 groups of four students each) had to simulate a company with three customers of different importance. These customers together with the project manager voted and these votes were the input to the ReleasePlanner® tool. In this experiment three releases were considered, Release 1, Release 2 and Release "postponed".

The groups considered between 23 and 62 requirements. The concerns considered were mostly from the viewpoint of a customer, but also the view of the project manager was considered. The analysis of the generated output used the dialogue method from Section 7.1. The following observations were made:

- The plan initially generated always gave rise to objections from the view of the concerns.
- The system was stable in the sense that only new pre-assignments were demanded and they led to few further changes in the plans.
- In three experiments one new plan had to be generated, in one experiment two plans and in one experiment five plans.
- In half of the experiments the old plan was preferred because the new plans showed unexpected and

unwanted results.

- In one experiment there were two plans with big differences from the concerns of two customers, and the project manager had to make a decision.

Although most of the explanation procedure was performed manually, the usefulness of the procedure became evident.

9. Conclusion

We presented an explanation based dialogue approach to improve solutions of wicked planning problems. It was based on the observation of the properties of wicked problems: Humans with unclear and contradicting opinions and software agents are involved. For the dialogue we focused on the communication between human and software agents. The dialogue has an explanatory character that gives insight into decisions so that the user may change previous opinions, or may stay with existing decisions. For this we proposed the forward explanation type.

In our approach a stakeholder is actively involved. First the stakeholder presents an ideal plan called a prototype. This plan is then compared with the actual plan using a similarity measure. The result is shown to the stakeholder in a simplified form and the stakeholder is asked for changes to the plan. These changes are given to the problem solver and the results are made visible in the same way by using the similarity measure. This then leads either to an improved plan or a better understanding of the old plan.

Therefore the proposed generic approach aims not only at increasing user trust on wicked planning results but also at improving the plans interactively. This approach can be instantiated and applied to many applications, e.g. release planning, investment planning, and urban planning. Release planning was the original motivation for our approach, and was the application where it was worked out in most detail and partially implemented.

Future work is concerned with the fact that the presented approach focuses on only one stakeholder's votes. An improved approach in the future should be able to integrate different stakeholders' votes together, and the explanations through dialogues should reflect this overall view. A second and theoretically important aspect is that presently there is no use made of dialogues for future problems. A learning support approach is underway, by the authors, which could contribute to bridging the gap between explanation and learning.

Acknowledgements

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for its financial support of this research. Many thanks are also due to Jim McElroy for his paper review and stimulating discussion.

References

[Baetz 1990] Baetz, B. J. W. 1990. Optimization/Simulation Modeling for Waste Capacity Planning Management. Journal for Urban Planning and Development 116, p.59-79.

[Belnap, Steel 1976] Belnap, N. D., Steel, T. B. 1976. *The Logic of Questions and Answers*. Yale University Press, New Haven, CT.

[Bodie et al. 2002] Bodie, Z., Kane, A., Marcus, A. J. 2002. *Investments (5th edition)*. MacGrawHill, Irwin.

[Burkhard, Richter 2000] Burkhard, H.D., Richter, M. M. 2000. On the Notion of Similarity in Case Based Reasoning and Fuzzy Theory. Soft Computing and Case Based Reasoning (Pal, S. et al eds). Springer Verlag.

[Christiansen et al. 2006] Christiansen, K., Droste, K., Hovestadt, L., Lehnerer, A.. 2006. To appear: 2006 Bishopsgate - A Society of Towers Performance Driven Simulation in Urban Design.

[Dellen, Maurer 1998] Dellen, B., Maurer, F. 1998. Change Impact Analysis Support for Software Development Processes. International Journal of Applied Software Technology, ISSN 1198-5577, Vol 4, No 2/3, International Academic Publishing, Scarborough, Ontario, Canada.

[Denzinger, Ruhe 2004] Denzinger, J., Ruhe G. 2004. Decision Support for Software Release Planning Using e-Assistants. Journal of Decision Support Systems, Vol 13 - No. 4, p. 399-421.

[Du 2004] Du, G. 2004. Design and Realization of an Explanation Component for Software Release Planning. Masters Thesis, Dept. of Computer Science, University of Calgary.

[Emigh 1999] Emigh, J. 1999. Total Cost of Ownership. Computerworld 51/1999, p.17-19.

[Fishburn 1986] Fishburn, P.C. 1986. The Axioms of Subjective Probability. Statistical Science, p. 335-358.

[Hillenbrand, Reuther 2003] Hilllenbrand, M., Reuther, B. 2003. Building Block for Web Application. 7th IASTED IMSA, Hawaii, p. 757-761.

[Hommel, Pritsch 1999] Hommel, U., Pritsch, U. 1999. Marktorientierte Investitionsbewertung mit dem Realoptionsansatz. Management 13, p.121-144.

[Kaplan, Norton 1993] Kaplan, D., Norton, R. 1993. Putting the Balanced Scoreboard to Work. Harvard Business Review, p.71-79.

[Keeney 1992] Keeney, R. L. 1992. *Value Focused Thinking: A Path to Creative Decision Making*. Harvard University Press.

[Lackoff 1987] Lakoff, G. 1987. *Women, Fire and Dangerous Things: What Categories Reveal about the Mind*. Chicago. IL: The University of Chicago.

[Lenz et al. 1998] Lenz, M., Bartsch-Spoerl, B., Burkhard, H. D., Wess, S. (eds) 1998. *Case-Based Reasoning Technology: From Foundations to Applications*. Springer Verlag.

[Rittel, Webber 1973] Rittel, H., Webber, M. 1973. Dilemmas in a General Theory of Planning. Policy Sciences 4, p. 155-169.

[Rittel, Webber 1984] Rittel, H., Webber, M. 1984. *Planning Problems are Wicked Problems.* Developments in Design Methodology (Cross, N. eds). Wiley.

[Ruhe, Ngo-The 2004] Ruhe, G., Ngo-The, A. 2004. Hybrid Intelligence in Software Release Planning. International Journal of Hybrid Intelligent Systems, Vol 1, p. 99-110.

[Tsang 1993] Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

[Wooley 1998] Wooley, B. A. 1998. Explanation Component of Software Systems. ACM CrossRoads.