Diagnosing Inconsistent Requirements Preferences in Distributed Software Projects

A. Felfernig, M. Schubert, M. Mandl, and P. Ghirardini

Applied Software Engineering
Graz University of Technology
Inffeldgasse 16b
A-8010 Graz, Austria
{alexander.felfernig, monika.schubert, monika.mandl, philipp.ghirardini}@ist.tugraz.at

Abstract: Requirements specification is often performed by stakeholders who are organizationally and geographically distributed. In these scenarios preferences regarding requirements can become inconsistent and stakeholders need assistance in identifying reasonable tradeoffs. In this paper we introduce a basic approach that actively supports distributed groups of stakeholders in the specification of consistent requirement preferences. The approach automatically detects minimal sets of preferences that have to be adapted in order to restore global consistency in the given collection of preferences. This approach can be seen as a core technology of requirements management processes in (distributed) software projects.

1 Introduction

Effective requirements specification is extremely important for successful software development [YWK08]. Especially in organizationally and geographically distributed projects the identification and resolution of inconsistent stakeholder preferences is crucial, for example, one stakeholder could prefer the inclusion of database system X whereas another stakeholder would like to have the database system Y. The distributed characteristic of software development makes consistency maintenance of requirements specifications a challenging task. Requirement preferences of stakeholders at different geographical locations have to be integrated. This can result in inconsistent preferences that have to be resolved. The most valuable result for all stakeholders is the one that takes the existing preferences into account as much as possible.

In this paper we show how the concepts of model-based diagnosis [Rei87] can be exploited for determining minimal sets of stakeholder preferences that have to be changed in order to make the global set of preferences consistent. We are interested in minimal adaptations of existing preferences that can be offered to stakeholders engaged in a requirements negotiation process, for example, which requirements should be integrated in the next software release?

Note that our work relies on the existence of a formal representation of interdependencies in the given set of requirements (see, e.g., [LRA08]) and the existence of a constraint solver [Tsa93] that allows to reason about the consistency of requirements.
The remainder of this paper is organized as follows. In Section 2 we introduce a simple set of requirements with a corresponding model about the dependencies between those requirements. Furthermore, we provide a set of requirements preferences that have been articulated by stakeholders. In Section 3 we introduce the basic concepts for identifying minimal sets of requirements that have to be adapted such that global consistency can be achieved. In Section 4 we provide a discussion of related work. With Section 5 we conclude the paper.

2 Working Example

We now introduce a basic set of abstract requirements (including their dependencies) that will serve as a simple working example throughout this paper. We model requirements in the form of constraint variables $r_i$ with domain$(r_i) = \{0, 1\}$ where $r_i = 0$ denotes the fact that requirement $r_i$ should not be integrated in the next software release and $r_i = 1$ indicates the fact that requirement $r_i$ must be integrated in the next release. In our working example we have to deal with the following set of requirements:

$$R = \{r_1, r_2, r_3, r_4, r_5\}$$

As already mentioned, the domain of each $r_i$ is specified with domain$(r_i) = \{0, 1\}$.¹ On the basis of this given set of finite domain variables representing requirements, we are able to add additional constraints that represent formal dependencies between requirements.

Requires constraints indicate the fact that a certain requirement $r_a$ is allowed to be integrated in a release only if a certain requirement $r_b$ is also part of the same release, for example, if the average query response time should be below 100ms, the database system of company A has to be purchased.

Incompatibility constraints denote the fact that a certain requirement $r_a$ must not be integrated in the same release with requirement $r_b$, for example, the calculation of optimal solutions is incompatible with systems response time below 2 seconds.

Resource constraints define basic properties that must be satisfied by the requirements integrated in a release, for example, the overall efforts for the next release must not exceed a certain upper bound.

These are typical constraints included in most of the existing requirements engineering ontologies such as SWORE [LRA08]. Note that the above mentioned constraints act as a kind of background knowledge, i.e., knowledge that is assumed to be correct and that has to be taken into account for planning the new release. In our working example, we will take into account the following set of abstract constraints (background knowledge): $C = \{c_1, c_2, c_3, c_4, c_5\}$.

$c_1$: $r_1 = 1 \Rightarrow r_3 = 1$; /* the inclusion of $r_1$ requires the inclusion of $r_3 */$

¹ Note that requirements can as well be related to variables with domain size > 2, however, binary variables have been chosen for this working example.
c₂: not (r₁ = 1 and r₂ = 1) /* r₁ and r₂ are incompatible */
c₃: r₁ = 1 ⇒ (r₃ = 1 and r₄ = 1) /* the inclusion of r₂ requires r₃ and r₄ */
c₄: r₃ = 1 ⇒ (r₄ = 0 and r₂ = 0) /* including r₅ requires the exclusion of r₄ and r₂ */
c₅: r₁ + r₂ + r₃ + r₄ + r₅ < 4 /* at most 3 requirements can be included */

In order to complete our release planning task we have to collect the requirements preferences of the different engaged stakeholders. For modelling the preferences in our working example, we allow three different types of constraints:

First, include(rᵢ) denotes that fact that a certain stakeholder wants to have included requirement rᵢ in the next software release.

Second, exclude(rᵢ) denotes that fact that a certain stakeholder does not want to include requirement rᵢ in the next release.

Finally prefer(rᵢ, rⱼ) denotes the fact that a certain stakeholder prefers the inclusion of requirement rᵢ over the inclusion of requirement rⱼ.

These three types of requirements preferences have been introduced for the purposes of our working example, however, can be extended by different other constraint types seen as useful in a certain application context. For our simple working example we assume that three stakeholders are specifying their preferences regarding the given set of requirements R = {r₁, r₂, r₃, r₄, r₅}. In this context, PREF=∪prefᵢ denotes the set of stakeholder preferences where prefᵢ denotes the jth preference of stakeholder i.

pref₁₁: r₁ = 1; /* include r₁ */
pref₁₂: r₂ = 1; /* include r₂ */
pref₂₁: r₂ = 0; /* exclude r₂ */
pref₂₂: r₅ = 0; /* exclude r₅ */
pref₂₃: r₃ = 1 and r₄ = 0 /* prefer r₃ over r₄ */
pref₃₁: r₂ = 1 and r₁ = 0 /* prefer r₂ over r₁ */
pref₃₂: r₄ = 1 and r₃ = 0 /* prefer r₄ over r₃ */

A corresponding release planning task can be defined as follows.

**Definition 1 (Release Planning Task).** A release planning task can be defined as a constraint satisfaction problem [Tsa93] (R, C, PREF). R=∪ᵢ represents a set of finite domain variables where dom(rᵢ)={0,1}. C=∪ⱼ represents a set of constraints on the variables in R. Finally, PREF=∪prefᵢ is a set of stakeholder preferences.

On the basis of this definition of a release planning task we can now introduce the definition of a solution for a release planning task (also denoted as release plan).

**Definition 2 (release plan):** a solution (release plan) for a given release planning task (R, C, PREF) is represented by an instantiation I = {r₁=v₁, r₂=v₂, ..., r₅=v₅} of the variables in R, where vₖ ∈ dom₀ and dom₀ denotes the domain of variable r₀.
A release plan is consistent if the assignments in I are consistent with the constraints in C and the stakeholder preferences in PREF. Furthermore, a release plan is complete if it is consistent and all the variables in R have an assigned value. Finally, a release plan is valid, if it is both consistent and complete.

The release plans for the release planning task \((R, C, \{\})\) are depicted in Table 1. If we want to include the additionally defined set of stakeholder preferences \(PREF = \{\text{pref}_{11}, \text{pref}_{12}, \text{pref}_{21}, \text{pref}_{22}, \text{pref}_{31}, \text{pref}_{32}\}\), the constraint solver is not able to calculate a corresponding release plan for the release planning task \((R, C, PREF)\). The question we have to answer now is: which is the minimal set of stakeholder preferences that have to be changed in order to be able to calculate a release plan for the defined release planning task? An algorithm for calculating such minimal sets of stakeholder preferences will be sketched in the following section.

<table>
<thead>
<tr>
<th>(r_1)</th>
<th>(r_2)</th>
<th>(r_3)</th>
<th>(r_4)</th>
<th>(r_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Example release plans for the release planning task \((R, C, \{\})\).

Note that we do not expect a complete specification of requirements interdependencies and stakeholder preferences. Stakeholders do not necessarily know all the existing interdependencies between the requirements and also are often not able to completely enumerate their preferences regarding the given set of requirements. However, even for incomplete preferences and dependencies sets our approach is able to effectively support the effective identification and repair of inconsistencies.

### 3 Diagnosing Inconsistent Preferences

In situations where no release plan can be identified for the release planning task \((R, C, PREF)\) we have to activate a diagnosis functionality [Rei87] that helps us to identify those stakeholder preferences that are responsible for the given inconsistency, i.e., preferences that have to be changed in order to be able to find at least one consistent release plan. We are interested in minimal changes since we want to keep the original set of stakeholder preferences the same as much as possible. The calculation of minimal changes to stakeholder preferences is based on the identification of conflict sets [Jun04].
Definition 3 (Conflict Set). A conflict set is a set of constraints $CS \subseteq \text{PREF}$ s.t. $CS \cup C \cup R$ does not allow the calculation of a release plan. $CS$ is minimal if there does not exist a conflict set $CS'$ with $CS' \subset CS$.

In the working example we are able to identify the three conflict sets $CS_1$: $\{\text{pref}_{11}, \text{pref}_{12}\}$, $CS_2$: $\{\text{pref}_{11}, \text{pref}_{31}\}$, and $CS_3$: $\{\text{pref}_{12}, \text{pref}_{21}\}$. All three are conflict sets since $\{\text{pref}_{11}, \text{pref}_{12}\} \cup \text{PREF}$, $\{\text{pref}_{11}, \text{pref}_{31}\} \cup \text{PREF}$, and $\{\text{pref}_{12}, \text{pref}_{21}\} \cup \text{PREF}$ are inconsistent, i.e., no release plan exists. Furthermore, all three conflict sets are minimal: there does not exist a subset $CS'$ such that $CS' \subset CS$ and $CS' \cup \text{PREF}$ is inconsistent.

In order to restore consistency in the given set of preferences $\text{PREF}$ we have to resolve each of the identified conflict sets by deleting at least one of the elements in the conflict set. A systematic way to calculate minimal diagnosis is to apply the concepts of model-based diagnosis (for details, see [Rei87]).

Definition 4 (Stakeholder Requirements Diagnosis). A stakeholder requirements diagnosis is defined as a set $\Delta \subseteq \text{PREF}$ s.t. the release planning task $(R, C, \text{PREF})$ has a corresponding solution. A diagnosis $\Delta \subseteq \text{PREF}$ is minimal if there does not exist any stakeholder requirements diagnosis $\Delta'$ with $\Delta' \subset \Delta$.

The major concept discussed in [Rei87] is the Hitting Set Directed Acyclic Graph (HSDAG) algorithm that allows us to calculate all minimal sets of stakeholder preferences that have to be adapted in order to be able to identify at least one release plan. The construction of such a HSDAG is exemplified in Figure 1. The approach assumes the existence of a corresponding algorithm that supports the identification of minimal conflict sets. In our prototype implementation we apply the QuickXplain algorithm proposed in [Jun04].

Let us now assume that our conflict detection algorithm returns $CS_1$: $\{\text{pref}_{11}, \text{pref}_{12}\}$ as the first conflict set. Since QuickXplain guarantees the minimality of calculated conflict sets, we only have to delete at least one element from each identified conflict set in order to be able to calculate a corresponding minimal diagnosis, i.e., a set $\Delta$ of stakeholder preferences that have to be deleted from $\text{PREF}$ in order to make the release planning task $(R, C, \text{PREF}-\Delta)$ consistent (at least one release plan can be found).

To resolve the conflict set $CS_1$, we can either delete the stakeholder preference $\text{pref}_{11}$ or the preference $\text{pref}_{12}$. We now decide to delete the preference $\text{pref}_{11}$ from the given set of preferences $\text{PREF}$ and again call the conflict detection algorithm with the reduced set of preferences $\text{PREF} = \{\text{pref}_{12}, \text{pref}_{21}, \text{pref}_{31}, \text{pref}_{22}, \text{pref}_{23}, \text{pref}_{31}, \text{pref}_{12}\}$. For the given release planning task $(R, C, \text{PREF}-\{\text{pref}_{11}\})$ we are still not able to identify a solution and the conflict detection algorithm returns the next conflict set which is $CS_3$: $\{\text{pref}_{12}, \text{pref}_{21}\}$.
Again, we have two alternatives to resolve the conflict. If we delete the element pref\_12, we are able to derive the first diagnosis $\Delta_1 = \{\text{pref\_11, pref\_12}\}$ since there exists at least one solution for the release planning task $(R,C,\text{PREF}-\{\text{pref\_11, pref\_12}\})$. If we delete the element pref\_21, we are able to derive the second diagnosis $\Delta_2 = \{\text{pref\_11, pref\_21}\}$ since at least one solution exists for the release planning task $(R,C,\text{PREF}-\{\text{pref\_11, pref\_21}\})$. Note that if a diagnosis has been identified, the conflict detection algorithm will not be able to calculate further conflicts [Rei87, Jun04].

If we decide to delete the second element of CS\_1 (which is pref\_12) from the given set of stakeholder preferences in PREF, the conflict detection algorithm returns another minimal conflict set for the remaining preferences PREF-\{pref\_12\} which is CS\_2: {\text{pref\_11, pref\_31}}. By resolving CS\_2, we are able to derive one further diagnosis which is $\Delta_3 = \{\text{pref\_12, pref\_31}\}$. $\Delta_3$ is a diagnosis since the release planning task $(R,C,\text{PREF-}\Delta_3)$ allows the calculation of at least one solution. Note that all three identified diagnoses $\Delta_i$ are minimal since there does not exist a $\Delta' \subset \Delta$ that fulfills the property of a diagnosis.

This way we are able to determine all minimal sets of stakeholder preferences that have to be changed (or deleted) in order to be able to find a release plan – such alternative diagnoses are then displayed to the stakeholder who is in charge of deciding on which minimal set of preferences should be changed.

The presented approach is a way to effectively support distributed decision making processes in situations where stakeholders articulate contradicting preferences regarding the set of requirements to be integrated into the next software release. Currently we have implemented the sketched conflict detection and diagnosis algorithms – an evaluation of those algorithms clearly shows their applicability in interactive settings [FFS09].

We are currently working on the design of a graphical user interface that supports requirements specification processes in distributed scenarios. With this implementation we want to evaluate the applicability of diagnosis concepts.
4 Related Work

The authors of [FFJ04] show how to apply the concepts of model-based diagnosis for the identification of minimal sets of faulty constraints in configuration knowledge bases – in this context, conflicts are induced by positive and negative examples for results expected to be calculated by a configuration knowledge base. In [SPF07] such minimal sets are called minimal exclusion sets, in [God97, McS04] the complements of such sets are denoted as maximally successful sub-queries. The calculation of minimal conflict sets in our context relies on the QuickXplain algorithm which has been introduced in [Jun04]. This algorithm is based on the principle of divide and conquer and thus guarantees logarithmic runtime performance in terms of the needed number of consistency checks.

The major contribution of our paper is to show how to apply the concepts of model-based diagnosis and conflict detection in the context of distributed requirements engineering scenarios – this approach shows potentials to significantly improve the overall quality of requirements engineering in terms of improved effectiveness in the identification and resolution of inconsistent sets of stakeholder preferences. We are currently working on the development of algorithms that further improve the performance of the diagnosis and the underlying conflict detection. First related results have been published in [SFM09].

Knowledge-based technologies enable a formal reasoning about the properties of a given set of stakeholder preferences. There already exist applications of knowledge-based technologies in the context of requirements engineering and release planning scenarios (see, for example [REP03, Goe08] ). All those approaches rely on the existence of a corresponding requirements engineering ontology (see, for example, [LRA08]) and a formal model of the dependencies between a given set of requirements. Compared to the existing work in the field of knowledge based systems & requirements engineering our work extends existing concepts with automated diagnosis techniques that help to reason about inconsistent requirements sets and to actively support stakeholders in release planning and decision making scenarios.

5 Conclusions

The effective identification of inconsistent preferences is crucial for a successful stakeholder decision support in (distributed) requirements engineering scenarios. In this paper we showed how to apply the concepts of model-based diagnosis to achieve this goal. The presented diagnosis approach is a key enabling technology that allows effective consistency management in different requirements engineering scenarios, such as open source projects or large and geographically/organizationally distributed software projects. Future work will include the recommendation of diagnoses, i.e., the identification those diagnoses with the highest utility for the stakeholders engaged in the decision making process. Finally, we plan detailed empirical studies in distributed projects.
References


