Guiding Modelers through Conflict Resolution: A Recommender for Model Versioning

Petra Brosch  Martina Seidl  Konrad Wieland

Business Informatics Group
Vienna University of Technology, Austria
{lastname}@big.tuwien.ac.at

Abstract
Like traditional code, software models are usually developed in teams requiring collaboration support in terms of version control systems (VCS). One use case of such a system is integrating concurrently evolved versions of one model into one consistent version. When the modifications are contradicting, then the VCS reports the conflict, but the cumbersome resolution process is left to the user. We present a recommender system which suggests automatically executable conflict resolution patterns.

Categories and Subject Descriptors D.2.9 [Software Engineering]: Management—Programming teams

General Terms Design, Languages

1. Introduction
When multiple developers concurrently work on the same artifact, conflicting modifications are very likely to be performed. Merging the different versions poses a very time-intensive, repetitive challenge in order to obtain a consistent artifact which integrates the work of all involved developers. To support the merging process, version control systems provide conflict detection components which highlight potential conflicts. The following task of conflict resolution is then manually carried out by one developer, preventing him/her from doing productive work. In order to unburden and support developers, if the artifact under consideration is a software model, we present a recommender system which suggests automatically executable conflict resolution patterns applicable to the given scenario. The recommender system is integrated into the adaptable model versioning system AMOR [1] which offers an enhanced conflict detection component. This component reports not only conflicts where atomic changes like add, update, and delete are involved, but also composite operations like refactorings which imply a multitude of possible resolution approaches.

2. Conflicts in Model Versioning
In the context of optimistic model versioning, where multiple modelers are allowed to modify the same artifact at the same time, conflicts due to overlapping changes occur whenever two or more modelers modify an artifact in a contradicting manner, i.e., if the involved operations are sequentially applied, the result is dependent on the order of the operations. Another kind of conflicts are violations occurring in the model, which integrates the valid versions of different modelers. For a detailed discussion on the notion of conflicts in model versioning, we kindly refer to [2].

An example scenario is shown in Fig. 1. The base version \( V_0 \) contains a UML Class Diagram consisting of the classes \( B \) and \( C \) which both contain an attribute \( \text{att} \) and which are both subclasses of the class \( A \). Then one modeler performs...
a refactoring pullUpField and shifts the common attribute to the superclass. At the same time, another modeler introduces the new class \( D \) which is also a subclass of \( A \). A naive merge incorporating all changes would result in a model, where \( D \) inherits the attribute \( att \). The conflict detection component of AMOR is able to report such a conflict due to the violation of one precondition of the refactoring, i.e., all subclasses must have an attribute with the given name. Then multiple resolution strategies are possible, including the trivial solutions like abandoning the modifications of one modeler or including all modifications. More sophisticated resolution strategies are shown in the lower half of Fig. 1.

In order to support the modeler in charge of conflict resolution we propose a recommender system offering resolution patterns. The recommendations should facilitate the resolution process since they are automatically executable, but also offer guidance by showing different strategies to eliminate a conflict.

3. Realization

The proposed recommender system is based on the Eclipse Modeling Framework and Eclipse Team Support. It extends the manual conflict resolution facilities of AMOR as shown in Fig. 2 which implements the basic interplay with the versioning server (cf. (1) in Fig. 2) and offers the possibility to remodel artifacts to resolve conflicts (2). The actual recommender system (3) supports the conflict resolution by providing a list of automatically applicable resolutions patterns for each conflict looked up from a repository of conflicts. The resolution patterns in this repository are either defined manually or are automatically mined as described in [1]. The proposed resolution patterns may be previewed, rolled back, and manually refined. For easier identification of conflicting operations in the preview mode, the conflicting operations are marked with the dedicated user symbol combined with annotations indicating the application of \textit{add}, \textit{delete}, and \textit{update} on the respective model elements. Recommended resolution strategies are marked accordingly with a system symbol (the cog). Previewing many operations at once may on the one hand overflow the model, but on the other hand may be necessary to understand changes. Therefore, the user may decide which operation should be displayed.

The recommended resolutions are ranked by their relevance. The relevance is calculated by a combination of the edit distance between the current conflict situation and the stored one, the number of applications so far, and the impact of the user who created the conflict resolution, by aggregating the application count of all resolutions created by this user. Resolution specific information is displayed in a dedicated property view (4). The property view contains metadata about the resolution’s origin, application and edit distance. Furthermore, since automatically derived resolutions do not have a name, users may enhance the resolution pattern by providing a meaningful name and tags.

References


Guiding Modelers through Conflict Resolution: A Recommender for Model Versioning

- Like traditional code, software models are usually developed in teams requiring collaboration support in terms of version control systems (VCS).
- One use case of such a system is integrating concurrently evolved versions of one model into one consistent version.
- When the modifications are contradicting, then the VCS reports a conflict, but the cumbersome resolution process is left to the user.
- We present a recommender system which suggests automatically executable conflict resolution patterns.

Resolution

- Recommendations
  The recommender system supports the conflict resolution by providing a list of automatically applicable resolutions patterns.

- Resolution Properties
  Resolution specific metadata is displayed in a dedicated property view.

Conflict

- Project Explorer
  The Eclipse Team Support plugin yields the basis for the interplay with the versioning server.

- Model Editor
  The model editor offers the possibility to remodel artifacts to resolve conflicts. Model elements are marked with a dedicated symbol indicating changes.

Sally’s Version

```
A
  └── att
     └── B
```

Harry’s Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```

Origin Version

```
A
  └── att
     └── B
```