Versioning of Knowledge Bases and Configurations

Andreas Falkner

Siemens Austria, Electronics Development Center
Erdberger Laende 26
A-1030 Vienna, Austria
andreas.falkner@siemens.at

Abstract

When knowledge about a configurable system grows and user requirements change over time, the corresponding knowledge base of the configurator software has to change as well. That will also affect existing configurations based upon that knowledge base. The knowledge engineer must find the appropriate balance between organizing the different versions of constraints in the knowledge base and upgrading configurations, thus making them consistent with the new constraints. Both of these alternatives have drawbacks and cause effort for knowledge engineers and users. A formal solution for this problem is still missing.

Introduction

The general context of a configuration system consists of:

- User requirements for a configurable system (range of products).
- Knowledge base (KB) as main part of a configurator software, consisting of a component hierarchy (object classes) and constraints for checking and/or generating objects and setting values. It is modeled by a knowledge engineer, based upon the user requirements.
- Configurations are object networks fulfilling all the constraints. They are built by user interaction, based upon the KB.
- The real world products (assembly of parts) are represented by the configurations.

In our technology center for configurators and diagnosis systems we have implemented several large-scale configuration systems, among them a configurator for the hard- and software of railway interlocking systems. The configurator comprises approximately 500 object classes and 3000 constraints and rules. Due to new features of the interlocking system and changes to the requirements we have to deploy up to 3 versions of the configurator per year with up to 400 new or changed constraints each. Experience shows that 20% of the changed constraints might contradict existing configurations. The handling of these contradictions depends on the needs of the users (whether the product must stay unchanged or not).

In this paper we describe critical types of changes and how to cope with them. We focus on the relationship between a KB and a configuration based upon it. The problems arising from that tight relationship are not covered by literature on software versioning like (Conradi and Westfechtel, 1996) or (Estublier and Casallas, 1995).

Types of changes

The following changes do not cause versioning problems:

- Changes to the configuration (using the same KB).
- Changes to the KB if the new KB is used only for new configurations starting from scratch.

Changes to the KB concerning components comprise:

- Change: e.g. new attributes, different behavior.
- New component type (object class): e.g. new features.
- Delete a component type: e.g. when no longer supported.

They are similar to schema evolutions in data bases (see Monk and Sommerville, 1993) and will not be considered further in this paper. As a general principle, constraints which only refer to new or deleted component types or attributes, do not interfere with existing constraints.

The following changes to the KB concerning constraints normally have effects on the existing configuration:

- Same premises, different conclusion: configuration might become invalid.
- New or more special constraint: configuration might become invalid.
- More general or deleted constraint: there might be a more efficient configuration.

Together with the user the knowledge engineer has to decide whether a change to the KB requires changes to the configuration. In that case updates are necessary – e.g. replacing out-dated components in the configuration (in the installed product, too) or changing a combination of components based on a constraint that turned out unsuitable.

Often the change to the KB need not or even must not affect existing parts of the configuration. E.g. if a component is no longer produced in the real world, most probably it will not be replaced with its successor for cost reasons. The same is true when a constraint is changed in

order to achieve improvements in the configuration (e.g. cheaper combination of components). In such cases the component type or constraint must remain in the KB. The KB must be able to handle mixtures of old and new features, which might lead to the difficult situation of contradicting constraints.

Contradicting constraints

Whenever the changed KB is used for accessing an old configuration, it might contain a new constraint which contradicts an old constraint. More precisely, the new constraint contradicts some part of the configuration, which is valid with respect to the old KB. Examples include simple cases like new versions of components, but also quite complicated relations between objects, e.g.:

- A new version of a certain card consumes less space in the frame, so that a changed constraint fills the frame more compactly. An old card will cause a contradiction.
- According to a new constraint each site must have a printer, whereas in former times sites of a special type were not allowed to have a printer.
- An old constraint states that there is a common failure indicator for the ventilation of all racks of an assembly, whereas nowadays they have to be monitored separately.

In general, it is not possible to detect contradicting constraints by just checking the set of all constraints in the KB. The question is whether at least one configuration fulfils all constraints. If the constraint language is powerful enough (e.g. has quantors and multiplication), this question is undecidable. For simpler languages the best algorithm might be of exponential order. An alternative approach based on diagnosis is described in (Felfernig et al., 1999).

Contradictions are prevented if constraints only of a certain KB version are evaluated at the same time. This kind of filtering requires the constraints labeled with the KB versions they are valid for. Objects, too, and in certain cases even attributes must be labeled in the same way if this is necessary for selecting the correct version of the constraint.[1]

Another possibility is using the KB versions or time stamps explicitly in the code of the constraint. The new constraints may also be implemented so that they accept old states during checks, but never generate them.

Keeping track of the history of changes in the ways mentioned above turned out to be very complex in our applications and was only carried out in particular cases with tailor-made implementations. In most cases we could avoid contradictions by keeping KB versions separate.

[1] The same situation applies to dynamically produced KBs when filtering produces several configurator variants for different countries, product lines, release dates, etc.

Separate knowledge base versions

This solution requires that the user can select the version of the KB or configurator to be used (e.g. use the old KB for old configurations). Whenever a user wants to access an old configuration with a new KB, s/he has to upgrade the configuration to the new KB beforehand. This is typically only necessary when the real world product also changes, and can happen even years after the last access to the configuration. Therefore upgrades over several KB versions must be allowed and supported.

In order to reduce the work load for the user, the upgrade process must be automated as much as possible. Nevertheless the user should check the modifications done by the configurator, especially when constraints could not be fulfilled. Then s/he might have to set new attributes, replace components, etc.

The trade-off between automatic upgrade (which might be quite expensive due to complex relationships and special cases) and manual user interactions (which might become time consuming if many configurations or components are involved) should be considered thoroughly. In our applications we noticed considerable upgrade effort both for knowledge engineers and users.

Conclusion

Before a knowledge engineer deploys a new version of a knowledge base, s/he has to find out whether the changes contradict existing configurations based upon that KB. In the general case this is undecidable or at least expensive. In our applications, upgrading configurations (thus making them consistent with the new KB) was useful in most cases, but sometimes costly. Organizing the different versions of constraints in the KB could be solved with tailor-made implementations where necessary. A formal approach with clear semantics would help to increase the productivity of knowledge engineers as well as users, but is still an open problem.

References

Conradi, R.; and Westfechtel, B. 1996. Configuring Versioned Software Products. *Lecture Notes in Computer Science* Vol.1167:88-109.

Estublier, J.; and Casallas, R. 1995. Three Dimensional Versioning. *Lecture Notes in Computer Science* Vol.1005:118-135.

Felfernig, A.; Friedrich, G. E.; Jannach D.; and Stumptner, M. 1999. Consistency based diagnosis of configuration knowledge-bases. *University of Klagenfurt, Technical Report* KLU-IFI-99-2.

Monk, S.; and Sommerville, I. 1993. Schema Evolution in OODBs Using Class Versioning. *SigMOD Report* 22(3):16-22, Sep.