# CASE-FX: Feature Modeling Support in an OO CASE Tool

Alain Forget
School of Computer Science
& Human-Oriented Technology Lab
Carleton University
aforget@scs.carleton.ca

Dave Arnold
School of Computer Science
Carleton University
darnold@scs.carleton.ca

Sonia Chiasson
School of Computer Science
& Human-Oriented Technology Lab
Carleton University
chiasson@scs.carleton.ca

## Abstract

Generative Programming advocates developing a family of systems rather than a set of single systems. Feature modeling can assist in supporting the development of such software product lines through software reuse. To our knowledge, CASE-FX is the first implementation of state-level feature modeling support within a CASE tool.

***Categories and Subject Descriptors*** D.2.2 [*Software Engineering*]: Design Tools and Techniques - Computer-aided software engineering (CASE); D.2.13 [*Software Engineering*]: Reusable Software - Domain engineering

***General Terms*** Design, Standardization

***Keywords*** CASE, feature modeling, Rational Rose-RT

## 1. Introduction

Generative Programming [4] advocates developing a family of systems rather than a set of individual systems in order to save development time and resources. *Feature modeling* is used to define a system familys features, which are the points of variability amongst the different instances within a system family. Instances of the system family can then be generated by varying the set of enabled feature values while maintaining the same core.

We implemented CASE-FX, a feature modeling add-in for Rational Rose RealTime [5]. Our goal was to demonstrate how feature modeling can be implemented in a CASE tool. CASE-FX bridges the gap between theory and application, being the first tool to support feature modeling at the state-level, to our knowledge.


*OOPSLA'07,* October 21–25, 2007, Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

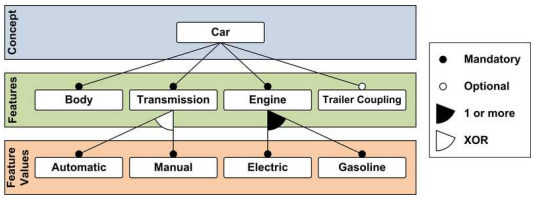


**Figure 1.** An example feature diagram of a Car [4]

## 2. Background

Czarnecki and Eisenecker [4] assert that object-oriented (OO) technology still leaves new opportunities to support reuse and configurability. They propose a system family approach wherein the product architecture is streamlined through domain analysis and design, establishing features, feature values, and configuration rules that can be implemented using generated components. Configuration rules consist of domain-imposed constraints that must be enforced upon the selection of feature values both within and across individual features.

Feature modeling can be described using the adapted feature diagram of a Car [4] provided in Figure 1. As shown, a Car must have a Body, Transmission (whose feature value is either Automatic or Manual, but not both), and an Engine (Electric, Gas, or both). It also has an optional Trailer Coupling. We distinguish between features and feature values to prevent an infinite cycle of features within features.

Existing work [2, 3] in software product line engineering applies software patterns, organizational workflows, and similar macro-level solutions to the same software reuse paradigm described by Czarnecki and Eisenecker. Our approach differs in that it instead provides support at a lower, state machine level.

Rational Rose RealTime (Rose-RT) [5] is a CASE tool with which UML constructs can be drawn, converted into source code, compiled, and executed. Rose-RT add-ins can be developed using its Extensibility Interface (RRTEI).

## 3. CASE-FX

CASE-FX supports feature modeling by allowing Rose-RT model developers to add, edit, and remove features and their values, as well as specify configuration rules. It is implemented as an add-in for Rose-RT version 6.3.

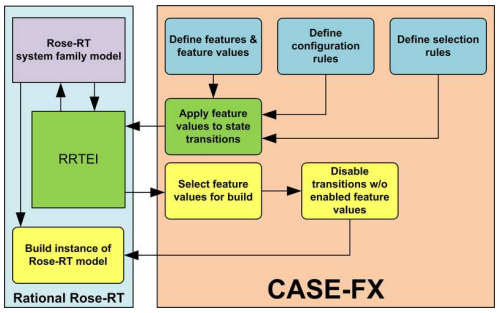


**Figure 2.** Flow diagram of CASE-FX and Rose-RT

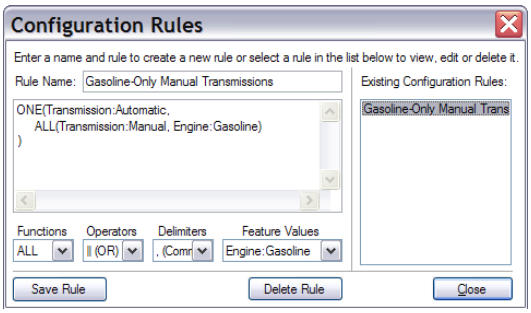


**Figure 3.** An example CASE-FX configuration rule

**Overview.** Figure 2 illustrates CASE-FXs interaction with Rose-RT. Developers define the initial system model as usual in Rose-RT. Features and feature values are defined in CASE-FX, as well as configuration rules describing the allowable system configurations. Transitions in the Rose-RT models state machines can then be tagged with the feature values they support. When building the model, CASE-FX ensures that only the chosen sets of feature values are active in the current instance of the system. CASE-FX extends Rose-RTs functionality by handling the semantics of the feature modeling constructs, storing the additional data in the existing Rose-RT model file.

**User Interface.** CASE-FX provides a standard form-based user interface for defining the features, feature values, and configuration rules (for an example, see Figure 3). The same interface is also used to enable the appropriate feature values for the current instance of the system.

**Configuration Rules.** CASE-FX uses an EBNF grammar to specify configuration rules. Figure 3 illustrates how configuration rules are managed in CASE-FX. The rules are interpreted using recursive-descent parsing [1]. The EBNF grammar is syntactically similar to fundamental C++, which should make it familiar to most developers. If a syntactically incorrect rule is entered, CASE-FX will notify the developer immediately where in the rule the error is located.

**Selection Rules.** Each feature in CASE-FX has a special type of configuration rule known as a *selection rule*. Selection rules define how many of a features feature values may be selected at any given time. This rule is set after a feature and its values are defined. The possible selection rules are: "one", "one or more", "zero or more", and "zero or one".

**Building CASE-FX-Enhanced Rose-RT Models.** As previously noted, state machine transitions must be tagged with the feature values they support. Before building their model, Rose-RT developers must tell CASE-FX to *pre-build* the model. Pre-building first ensures that the selected feature values do not violate the configuration rules. CASE-FX then sets the guards for all transitions with no enabled feature values to *false* and appends the suffix "_DISABLED" to each excluded transitions name. Next, developers build the model (using the standard Rose-RT build function), assured that only the selected feature values will be used in this built instance of the system family. After the build is completed, developers must *post-build*, which removes the "_DISABLED" suffixes from excluded transitions and resets the *false* guards to their previous values.

**Generating System Family Instances.** After the initial creation of features, feature values, and configuration rules, followed by the tagging of corresponding transitions, the CASE-FX-enhanced Rose-RT model contains a family of systems. Generating distinct instances of the system requires only selecting the desired feature values and completing the 3-step build process. Furthermore, all information is stored permanently within the native Rose-RT model file, so the initial feature modeling setup need only be done once.

## 4. Discussion and Conclusion

Pre-compiler instructions and *if-then* statements could accomplish the same task as CASE-FX, but leading to reduced performance and code that is difficult to manage. Our tool comprehensibly utilizes Czarnecki and Eiseneckers feature modeling approach, enabling Rose-RT developers to better create and manage a complete set of system family features. To our knowledge, this is the first attempt to implement state-level feature modeling functionality into a CASE tool.

Due to limitations of the RRTEI, we were unable to provide a one-step build process and unused transitions had to be disabled rather than removed from the build.

Feature modeling is a powerful design-for-reuse concept which can prevent superfluous work and save development time. With CASE-FX, we have shown how these theoretical concepts can be added to existing CASE tools to help developers create and manage system families. We hope the area will be further explored and that feature modeling capabilities become a standard component of CASE tools.

## References

[1] Appel, A. Modern Compiler Implementation in Java Second Edition. Cambridge University Press, 2002.

[2] Bosch, J. Design and Use of Software Architectures: Adopting and Evolving a Project-Line Approach. Addison-Wesley, 2000.

[3] Clements, P. and Northrop, L. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[4] Czarnecki, K. and Eisenecker, U. Components of Generative Programming. Proc. of the 7th European software engineering conference, September 1999.

[5] IBM Rational Software. Home page. Accessed June 2007. http://www-306.ibm.com/software/rational/