KAPTUR, Elvis, Hendrix, and Other Acronyms: Domain Engineering at CTA

Sidney C. Bailin

CTA Incorporated
6116 Executive Boulevard
Rockville, MD 20852
Tel: (301) 816-1200
Email: sbailin@cta.com

Abstract

This is less a position paper than a short summary of what CTA's Software Development Automation Group (SDAG) is doing in the area of software reuse. You can think of it as a statement of what we see as important, and the key techniques to be pursued. For background on our methodology of reuse, see my WISR 92 position paper, as well as others listed in the references section.

Keywords: Domain Analysis, Reuse, Model-Based Reasoning, Case-Based Reasoning, Concept Formation, Repository Interconnection

Workshop Goals: Networking. Checkpointing the state of the community.

1 Background

We have been working in reuse for NASA/Goddard Space Flight Center's Data Systems Technology Division since 1986. This work has led to the development of most of the tools described below. Recently, we have begun working as part of the Unisys STARS team to introduce KAPTUR technology into that environment, and to unify it withthe STARS Orgasnizational Domain Modeling (ODM) method. We have also recently begun work with the Air Force's Rome Laboratory to incorporate a Hendrix-like learning capability in the Knowledge Based Software Assistant.

2 Position

KAPTUR: Domain Engineering Tool. KAPTUR (Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales) is a tool for recording, structuring, and reusing engineering knowledge. Such knowledge includes issues that were raised during development, alternatives that were considered, and the reasons for choosing one alternative over others. KAPTUR organizes software knowledge into domains, which are families of similar systems (examples of domains include satellite control center software, radar manager software, etc.). Within a given domain, assetsQ existing systems and subsystems, object classes, function implementationsQ are organized in terms of their distinctive features. Each distinctive feature represents an important engineering decision that went into the development of an asset. The distinctive features provide a means of comparing and contrasting alternative technical approaches in a domain. Assets at any hierarchical level can be compared and contrasted in this way, from system and subsystem architectures down to individual function implementations.

Each feature of an asset has certain information that is always attached to it:

- a description of the engineering decision that the feature represents
- a summary of the tradeoffs that were considered in arriving at the decision
- the ultimate rationale for the deicision.

KAPTUR uses hypertext techniques to allow the user to navigate among assets, their features, and the background information of a feature. The user can ask to see the altnernatives of a given feature, and will be pointed to those assets not possessing that feature.

ElvisC - A Tool for Building a Domain Taxonomy. ElvisC [1] is a tool that applies a concept formation algorithm (called Cobweb) to automatically organize assets into a hierarchy of meaningful categories. Asset descriptions are provided to ElvisC in terms of features. The feature space is open-ended, and can be interactively extended when an asset is entered. In essence, ElvisC looks for features that tend to occur together, and uses these as the basis for defining clusters of assets. ElvisC was originally developed for NASA/Goddard as a tool for organizing and maintaining a component repository, but we see it also as a domain analysis tool. Domains typically go through a process of evolution: in the early stages a faceted classification is often the most natural or feasible way to describe concepts in the domain; later, as practice becomes more regular and the alternatives become clearer, a hierarchical classification becomes feasible. ElvisC can be used as a tool to facilitate this evolution by suggesting likely categories in a hierarchical classification.

[1] ElvisC stands for Experiment in Libraries with Incremental Schemata and Cobweb.

Repository Interconnection Standard. We are active in an AIAA working group to define a standard for reuse repository interconnection. This work is being performed in conjunction with the Reuse Library Interoperability Group (RIG). The work builds upon the Asset Library Open Architecture Framework (ALOAF) developed for the STARS program, and is based on a three-level information model which carries the ALOAF to a greater degree of detail. This work has just begun and is expected to result in prototype demonstrations towards the end of the calendar year.

Domain Analysis in Flight Simulation. The Mission Simulator System (MSS) is a highly configurable flight simulator. MSS is sold as a product and has also served as the basis for several part-task trainers (PTTs) built for the Government, including the F-15/F-16 PTTs for the Air National Guard. The configurability of MSS stems from the fact that instruments, controls, engines, weapons, operational flight programs (OFPs), jammers, artilleries, radars, and missiles (JARMs) can be added or deleted without affecting the remainder of the simulation. MSS is a commercially successful example of domain engineering: the software adheres to a reference architecture for flight simulators that can be (and has been) instantiated to meet widely varying requirements.

Domain Analysis in Satellite Control. In 1988 CTA performed a domain analysis of satellite control centers for NASA/Goddard Space Flight Center. The analysis was based on a study of seven existing systems, from which a reference architecture was abstracted. The architecture was then refined using object-oriented partitioning criteria. The KAPTUR tool was developed initially to support this work. Since then, a domain analysis of satellite command management systems has been performed for the same client, and the results put into KAPTUR. Work is now continuing on the development

Knowledge from Pictures Environment. This is a multi-tool environment intended to support high-level model-based reasoning about software. The environment infrastructure consists of a graphical language for describing component interconnection, a table-based behavior description language, and a model repository. The repository is a set of model descriptions that cross- reference one another. In the model repository, there is no explicit notion of system, only the notion of component. A component may contain other components, and in this sense it may be considered as a system; on the other hand, a component containing other components may itself occur as a subcomponent in a still higher-level component. This uniform treatment of "components" and "systems" encourages the reuse of existing models as building blocks in new models, with the consequent semantic benefits mentioned in the Introduction.

We distinguish between component types and component instances. Each model in the repository describes a component type. Instances of this type may occur in other (higher-level) models. The description of a higher-level model M references the descriptions of the component types whose instances M contains.

The distinction between component types and instances has a couple of advantages. First, a model can contain more than one instance of a given component type. For example, the model of a building's climate control system may contain more than one air-conditioning unit. Second, a change to the definition of a component type is automatically propagated to its instances in all other models. The user of the tool is need perform multiple updates to implement a single conceptual change.

Tools that operate in the KFP environment include the Formal Interconnection Analysis Tool for verifying design properties, the Diagnostics Inferred from Graphics Tool which generates fault detection and isolation rules from the model descriptions, and the Multi-Aspect Simulation Tool, described below.

Multi-Aspect Simulation Tool (MAST). MAST is an environment for building and executing object-oriented models of complex electro-mechanical systems. The design is based on the connection manager approach described in the Software Engineering Institute's (SEI's) recommendations for flight simulators. The SEI approach has been extended by independently formalizing each aspect of a component's behavior, integrating work on discrete event simulation done by Bernard Zeigler at the University of Arizona, and implementing the design using the object-oriented techniques of multiple inheritance and virtual base classes. The models produced in this environment are highly comprehensible and unusually maintainable. The subcomponents produced during construction of the model have been reused in different models without modification. The types of behavior exhibited by the model during simulation have been modified and extended without difficulty.

Hendrix: A Meta-Tool. Hendrix [2] is an existing meta-modeling capability which has grown out of CTAUs work for NASA/Goddard and is based on CTA's Configurable Graphical Editor. Hendrix started off as an automated software design critic which is configurable to support different graphical design notations and different design rules. The Hendrix rule base is implemented in NASA's CLIPS language. Hendrix supports two functions that have made its evolution into a meta-tool a technically straightforward task: 1) the ability of the user to easily define new design rules (without having to code them in CLIPS), and 2) the ability of the user to define new design concepts in terms of previously defined concepts.

Defining new rules in Hendrix. The user defines a new design-evaluation rule by drawing an example of the erroneous situation which is to be caught by the tool. Hendrix generalizes the example into a CLIPS rule that will detect instances of this situation within an engineering model. The user specifies a diagnostic message to be issued when the rule fires. The diagnostic message can reference the design elements in the exampleQthese element identifiers are replaced by variables in the generated rule, and in any particular model will be instantiated to the model elements involved in the violation.

Defining new concepts in Hendrix. A similar approach is used in Hendrix to allow the user to define new concepts. In this case, the user draws an example of an instance of the concept, and Hendrix generates a CLIPS rule that asserts the concept as holding whenever this pattern is detected in a model. More than one pattern can be designated as examples of a particular concept. Hendrix generates one recognition rule for each pattern (this allows the user to define recursive concepts).

Associating concepts with graphical symbols in Hendrix. Having defined a new concept to Hendrix, the user is prompted to select either an arc type or a node type to represent the concept. A palette of available arc types (e.g., dotted, dashed, with/without arrows, etc.) and node types (i.e., different geometric shapes) are presented. If the new concept is a relation between objects, the user is prompted to select an arc type; if the new concept is a type of object, the user is prompted to select a node type to represent that type of object.

3 Comparison

The KAPTUR approach to domain analysis is similar in spirit and in some details to the STARS Organizational Domain Modeling method (see the paper by Roberta Burdick in this workshop). Specifically, the emphasis on describing exemplars, the distinction between descriptive and prescriptive modeling, the use of a hierarchical feature space for characterizing alternatives in the domain, and the emphasis on capturing contextual information such as tradeoffs and rationales,

[2] Hendrix stands for Help Evaluating New Designs with Rules Interactively Extendible.

are all common between KAPTUR and ODM. This has led us to seek a unification of the two approaches.

Our work on model-based reasoning in software engineering draws on ideas developed at the Software Engineering Institute (see Sholom Cohen's paper in this workshop) and on ideas of Parnas (1990) and Harel (1992). The basic motivation is to view softwae engineering as a process of creating models, asking questions about them, and refining them, with as much automated code geneation as possible to convert the models into code.

There has been a fair amount of experimental work in applying machine learning to aspects of software development (Esteva and Reynolds, 1990; O'Reilly and Oppacher, 1991; Reynolds and Maletic, 1991; Willis and Paddon, 1991; Wu and Leong, 1991) but our work views learning as an essential aspect of a knowledge-based software development environment. In this sense, our work is guided more by encounters with the problems of knowledge-based software assistance than by an interest in applying machine learning to a new domain.

4 References

S.C. Bailin and S. Henderson. A tool for reasoning about software models. Proceedings of the Computer Assurance Conference, ACM Press, June 1993.

S.C. Bailin and S. Henderson. An application of machine learning to the organization of institutional software repositories. To appear in Telematics and Informatics, September 1993.

S.C. Bailin, F. Paterra, W. Truszkowski, and S. Henderson. Model- based reasoning for system and software engineering. To appear in Telematics and Informatics, September 1993.

Esteva, J.C. and Reynolds, R.G., 1990. Learning to recognize reusable software by induction. Proceedings of the 2nd International Conference on Software Engineering and Knowledge Engineering, Skokie, IL. June 1990.

Harandi, M. and Lee, H-Y. Acquiring software design schemas: a machine learning perspective. Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference, IEEE Computer Society Press. 1991.

Harel, D., 1992. Biting the silver bullet: Toward a brighter future for system development. IEEE Computer, January 1992.

Lee, K. et. al., 1990. An OOD paradigm for flight simulators, 2nd edition. Technical Report of the Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

O'Reilly, U.M. and Oppacher, F. Learning new features and heuristics for matching and using cases. Proceedings of the Fourth Florida Artificial Intelligence Research Symposium, Florida AI Research Society. April 1991.

Parnas, D., Asmis, G., and Madey, J., 1990. Assesment of safety-critical software. Technical Report 90-295, ISSN 0836-0227. Telecommunications Research Institute of Ontario. Queens University, Kingston, Ontario.

Reynolds, R.G. and Maletic, J.I., 1991. Operationalizing software reuse as a problem in machine learning. Proceedings of the Fourth Florida Artificial Intelligence Research Symposium, Florida AI

Research Society. April 1991.

Willis, C.P. and Paddon, D.J. Combining explanation-based learning and Knuth-Bendix completion for equational reasoning. Proceedings of the Fourth Florida Artificial Intelligence Research Symposium, Florida AI Research Society. April 1991.

Wu, F.Y. and Leong, S. A syntax directed approach for learning software translation knowledge. Proceedings of the Fourth Florida Artificial Intelligence Research Symposium, Florida AI Research Society. April 1991.