In Proceedings of the First European Workshop on Model Driven Architecture with Emphasis on Industrial Application, March 17-18, 2004. Enschede, The Netherlands.

Tool Support for Model Based Architectural Design for Automotive Control Systems

Kevin Steppe, David Garlan, Greg Bylenok, Bradley Schmerl, Kanat Abirov, Nataliya Shevchenko

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15221

{ksteppe, garland, gbylenok, schmerl, kanatknt, san}@cs.cmu.edu

Abstract

In conjunction with Ford Motor Company, we built a tool to support multilevel architectural design. The tool, called Synergy, allows Ford to visually design architectures of vehicle control components. The components are imported from existing Simulink models; then the tool automatically generates a detailed view showing all required connections and ports. The resulting model is exported to Simulink for further analysis. In this paper we describe the conceptual and technical challenges encountered in building Synergy and our design choices for solving them.

1. Motivation

Over the past few years, Ford Motor Company has experienced significant benefits in software development using component models. By using Simulink [3] models of software components, Ford is able to analyze component behavior before committing to code. While rigorous component analysis has been successful at Ford, the approach has been difficult to scale.

Scaling the approach requires producing assemblies of these components. Currently these assemblies, if they are built at all, are constructed manually. Because components typically have dozens of interfaces each, manually connecting them is tedious and error prone. For a simple six-component subsystem, Ford engineers report that construction takes approximately two weeks. Large (50-component) vehicle control subsystems have taken six months to produce. Manual construction quickly becomes infeasible as the number of component choices and design possibilities explodes upwards.

2. Related Work

There is considerable interest in model-based approaches to embedded control systems, including avionics as well as automotive systems. The DARPA-sponsored MoBIES Project [1], for example, specifically focuses on this area. The MoBIES community anticipates considerable benefits in quality through model checking, reuse of proven components, and sharing of tools. Our work fits within that general category of research, but explores the specific consequences of using architecture description languages as the carriers of embedded systems designs.

Recently, the Object Management Group has been promoting model-based design using a two-tiered approach that they refer to as "Model-Driven Architecture" (MDA) [4]. MDA is motivated by similar concerns to ours, but attempts to advance the state of understanding about how to carry out such an approach in the context of real systems, complementing existing development methods, and leveraging special features of a product domain (in our case automotive control systems).

3. Proposed Solution

Ford's challenge is to be able to quickly design large systems without sacrificing the detailed model analysis capabilities, which have already proven to be successful. The two-tiered approach supports both goals. The abstract view allows engineers to work at an architectural level, facilitating building larger, more complex software systems within vehicles. The architectural tier allows the designer to think in terms of components or component groups, without viewing all the details needed to perform the analyses. The second tier then provides the details needed for the rigorous analysis of the design. A tool able to work on both levels is insufficient, however. Generating the detailed view requires connecting dozens of ports for every component, which is tedious and error-prone when performed manually. An effective tool must automate the generation of the detail tier.

The two-tiered approach shows promise even conceptually, but there are a number of technical challenges to overcome in making it feasible for use. These include:

- representation of the views
- scaling via hierarchy
- model interface specification
- design alternatives
- support for multiple types of analysis
- visual clarity,
- inter-tier consistency

The rest of this abstract gives a brief summary of these issues and our solutions.

3.1. Representation

The most obvious issue in a two-tier approach is representation of each level. As our intent was to simplify component compositions, the abstract level needed to be easily created by the user. We also wanted to facilitate some analysis at the abstract level before generating detailed assemblies. The detailed level had to support the many ports and connections present in the system, as well as the properties to be analyzed. We used the Acme Architecture Description Language (ADL) to describe both levels of design, with separate but similar styles for each level [2]. This allowed for presentation of different design aspects, while making automatic conversion straightforward.

3.2. Hierarchy

Managing the one hundred plus components in Ford's architectures requires breaking the system into modules and subsystems via architectural hierarchy. Ford engineers generally work with only five or six components at a time, constructing larger systems from these subsystems. Additionally, hierarchy provides a mechanism for reusing assemblies. Thus large architectures can be built in a bottom-up fashion from previously built assemblies. In our case, hierarchical design was supported through Acme "representations". In representations, an abstract component is a placeholder for the underlying substructure, a scheme which matched our intended use of hierarchy.

3.3. Interfaces

Any design will need to interact with other systems, requiring that the system representation include facilities for identifying that interface. The interface is also crucial to using hierarchy and building progressively larger architectures. Options for determining the interface include explicit specification by the designer, automatic generation from the available ports of the system, or leaving unbound ports. Explicit specification enables the system to check for completeness, but requires a extra work for complex interfaces. Automatic generation is easier on the user, but can mask sub-system incompleteness by generating additional inputs. Leaving unbound ports allows the user to review the incomplete system to pick needed inputs, but prevents full analysis. Because Ford needs to determine whether all input ports in a model are connected we chose to have the interfaces be explicit. Explicit interfaces allows us to verify the completeness of the system, both for inputs and expected outputs.

3.4. Design Alternatives

Exploration of design alternatives is an important feature for Ford. With a large collection of components there are often multiple choices, which fulfill the required interface yet have different properties. Ford often uses a single architecture across several car makes and models, the differences coming in the choice of components to fill in that architecture. The abstract component representation must be able to express these alternatives. We used Acme's support for multiple representations to indicate mutually exclusive design alternatives for any component or subsystem in the model.

3.5. Support for Multiple Types of Analysis

Additionally various properties are needed to support analysis of the system. Synergy supports analysis of three kinds. First are topological constraints and properties checked by AcmeStudio. Another set of properties is pulled from the Simulink models – either analyzed through Simulink directly, or added to the architectural description automatically. The component linking mentioned above enables Ford to take full advantage of Simulink's power to analyze assemblies generated by Synergy. Lastly, our tool uses its own component characterization file containing properties that can be analyzed through a plug-in framework in Synergy. Two such plug-ins were built – one checking resource usage of components, the other suggesting scheduling order.

3.6. Visual Clarity

As mentioned previously, rather than source-code, Synergy generates a detailed model from the abstract architecture. The newly generated model contains significantly more detail than the source model, and care must be taken to keep the model readable. Components should be laid out intelligently so that connections between components remain clear. Clarity remains a challenge within these highly detailed models, where each component may involve twenty to thirty individual connections. Synergy employs a simple layout engine, which keeps the detailed layout similar to the user constructed architecture.

3.7. Inter-tier Consistency

With multiple models involved in the development process, consistency between models becomes an important issue. Synergy gives the developer free-reign to tinker with the design at both the high and low levels of abstraction, so keeping them synchronized remains a challenge. Changes to the system architecture or individual component models must be carried forward into the generated assemblies. With our tool this is accomplished by regenerating the assemblies. Reverse engineering is not supported, because developers are expected to make at most minor targeted changes to the assemblies.

4. Conclusion

The above discussion illustrates the major issues in building Synergy: architecture representation, hierarchy, interface specification, analysis support, visual layout, and consistency. Our choices focused on Ford's needs and ease of implementation. Similar but separate styles make automatic conversion easy, while supporting the difference in detail. We used AcmeStudio's built-in representations to support hierarchy and thus enable generation of large modules in an understandable fashion. The external interface for a design is specified by the user through use of a special component. Design space is also supported by Acme representations, considering each representation to be set of mutually exclusive alternatives. Lastly, an analysis plug-in framework is part of both Synergy and AcmeStudio. We felt that our architectural representation is rich enough to support many more analyses than we could develop. Thus Synergy makes it easy for the customer to add more without having to address representational issues.

We used AcmeStudio to provide graphical editing and the previously mentioned representational facilities. While automatic layout will always be difficult for large, complicated systems, we took advantage of the user constructed system architecture as a starting point. Integration with other tools was a major requirement from Ford, which led us to select Eclipse and AcmeStudio as a base for plug-in development. Ford is focused on the architecture and pushing for automatic generation, thus Synergy maintains consistency in a feed-forward mechanism, re-generating the detailed model when requested.

References

[1] Bay, J.B. Model-Based Integration of Embedded Software.
http://dtsn.darpa.mil/ixo/programdetail.asp?progid=38.
[2] Garlan, D., Monroe, R.T., and Wile, D. Acme: Architectural Description of Component-Based Systems. *Foundations of Component-Based Systems*. Leavens, G.T., and Sitaraman, M. (eds). Cambridge University Press, 2000 pp. 47-68.
[3] The Mathworks. Simulink 5.1. http://www.mathworks.com/products/simulink.
[4] Object Management Group. MDA: The Architecture of Choice for a Changing World.
http://www.omg.org/mda.