CADENA: Enabling CCM-based Application Development in Eclipse *

Venkatesh Prasad Ranganath Adam Childs Jesse Greenwald
Matthew B. Dwyer John Hatcliff Gurdip Singh
Department of Computing and Information Sciences, Kansas State University, US
{rvprasad,achilds,jesse,dwyer,hatcliff,singh}@cis.ksu.edu

Abstract

To support the trend toward component-based systems, the Cadena project aims to provide an Eclipse-based development environment that includes support for design, behavior modeling, formal reasoning, and automated code synthesis for systems built using the CORBA Component Model. In this paper, we describe the basic functionality of the Cadena tool, and summarize how it is currently being evaluated by industrial partners for use in developing high-assurance avionics applications.

1 Introduction

Component-based architectures are often preferred in the development of large-scale and highly-configurable applications. Enterprise Java Beans[1] and CORBA Component Model(CCM)[2] are two such architectures with well-defined software development process. There are a number of tools and technologies that are being developed or supplemented in ways to support such architectures. Even companies involved in real-time safety/mission-critical domains have explored these architectures and have expressed interest in adapting them.

As always with any architecture, good tool support is needed to adapt and use it in real world. As the component-based architectures are relatively new, the required tool support is often lacking or primitive (command-prompt based scripts) – especially in the case of CCM. In particular, tools to provide design advice, synthesize aspects or properties of the application, perform optimizations, and manage the artifacts of the application are lacking. As the development of component-based applications involves development and assembling of components, reasoning about the correctness of the components and component assemblies can be an overwhelming task when carried out manually. Thus, the current state of affairs could benefit greatly from an environment to develop, reason, and prove properties about large-scale component-based applications. Given the interest of industries dealing with real-time safety/mission-critical domain, the degree to which support is available to address these issues will strongly influence the acceptance of these architectures.

We have developed the Cadena environment in an attempt to address these issues. *Cadena*[1][3] is implemented as an Eclipse plug-in and supports the development of component-based application based on CCM specifications. The plug-in provides the usual features found in an IDE such as editors for various file formats, support to manage related files as a project, interfaces to integrate and control various CCM implementations to build applications, and supplementary instrumentation interfaces to extend/enhance the plug-in or use it's results in useful ways. In contrast to all existing CCM development environments of which we are aware, Cadena supports various analyses that enable the user to perform sanity checks with ease, an intu-

*This work was supported in part by the U.S. Army Research Office (DAAD190110564), by DARPA/IXO's PCES program (AFRL Contract F33615-00-C-3044), by NSF (CCR-0306607) by Lockheed Martin, by Rockwell-Collins, and by Intel Corporation (Grant 11462).

[1]Available from `http://cadena.projects.cis.ksu.edu`

itive way to visualize the results of such analyses (based on *GEF*), a form-based editor to assemble applications in a type safe manner, and languages to describe various aspects of application models in human readable form along with translators to generate machine readable data from these descriptions.

Another distinguishing feature of the Cadena project is its emphasis on light-weight formal verification. Our group has extensive experience in the area of software verification techniques, and we have recently initiated the Cadena project to respond to the growing use of component-based systems, particularly in distributed real-time embedded (DRE) systems where high-assurance is required. We view Cadena as our vehicle for injecting various forms of verification based on static analysis and model-checking technologies into the development process for component-based systems. Accordingly, Cadena provides several different light-weight specification formalisms for capturing behavioral properties such as data and control dependencies and abstract state-machine semantics at the level of component interfaces. These semantic specifications are automatically checked using a combination of architectural level slicing (static dependence analysis) and model-checking [3].

Cadena was developed in the context of the DARPA Program Composition for Embedded Systems (PCES) project in which we are required to evaluate our technology using an experimental testbed from Boeing's Bold Stroke avionics middleware project. Bold Stroke provides support for developing the mission-control software for several of Boeing's military aircraft including the F-18 fighter. To provide support for this real-time embedded domain, Cadena leverages its behavioral specifications and dependence analysis to provide aspect synthesis for simple real-time and quality-of-service attributes.

The rest of this paper is organized as follows. In the following section, we describe component-based architecture with its benefits and drawbacks from a software development process point of view. We also present reasons why we think Eclipse and component-based architecture development process are an ideal match. In Section 3, we present the various features and capabilities of Cadena. We summarize the real world use of Cadena in Section 4. We conclude in Section 5 with an outline of the extensions and enhancements to Cadena that are in progress at present and are planned for in the future.

2 Motivation

2.1 Component Architecture

There is no widely accepted definition of the term *component*. A general consensus in the software community is that it is a large executable entity, its interface with the enclosing execution context[2] is well defined, and it can be easily replaced in its context. Given this description, component-based architecture can be described as an architecture in which components can advertise their functionality via well-defined interfaces to the context. The well-defined interface between the component and the context is defined by the architecture.

In a concrete architecture, such as CCM, the component's interface will be defined using CORBA's *Interface Definition Language (IDL)* by the developer of the component. Components use and offer services via event-based ports and message-based facets/receptacles (outgoing/incoming interface handles). The developer can further control certain details pertaining to the structure of the components in terms of implementation and persistence via *Component Implementation Definition Language (CIDL)*. From these descriptions the developer can then mechanically generate skeletal implementations in languages of his/her choice. To make the component system independent, the developer will also describe the facilities required from the context by the component in a `ccd` file. From an implementation point of view, the developer will specify implementation details, such as object code format, about a particular component implementation in a `csd` file. This information is required to use/deploy the component. From a process point of view, it is the responsibility of the application assembler to create an assembly from the available components to realize the application. This assembly is stored in a `cad` file. From hereon, it is the responsibility of the deployer to use various artifacts(files) to deploy components on various application servers within suitable containers (context) to construct the specified application. Artifacts in this work flow can be combined, say

[2]Hereon, we will use the term context to indicate enclosing execution context.

CIDL with IDL, in various ways leading to many possible artifacts different in details but similar in functionality.

Given the number of artifacts, their plausible combinations, and distinct owners in the overview of the development and deployment process, it is easy imagine the degree of complexity involved in managing a collection of artifacts correctly to realize the specified application. Most of these artifacts are automation-ready as they are XML documents. This adds to the complexity as XML documents are human-unfriendly in the absence of an intelligent XML editor/viewer.

Features such as structured and automation-ready artifacts, well-defined work flow, and mechanical translations from high level interface languages to low level languages with environment specific implementations should provide an opportunity for tool developers to provide tools to deal with these artifacts at a higher level and to generate these more specific artifacts in a easy way. If these tools interplay on one development platform, the software development process will be simplified and human resources can be expended on more demanding issues such as reasoning about correctness.

2.2 Why Eclipse?

Eclipse is an extensible platform programmed in Java on which various specific task-related extensions have been stacked. Such stacking is possible due to the stable core architecture which is extensible via well-defined protocols to add new features and capabilities.

The platform is largely system independent. The fact that the extensions or plug-ins[3] to the platform must adhere to well-defined interfaces defined by the core of the platform and that the plug-ins can communicate via these interfaces allows tool providers to expose their tools as plug-ins with various degrees of dependencies on other plug-ins. Since plug-ins are identified via names/id and interfaces rather than classes, the dependence between plug-ins is orthogonal to their implementation. All these facts make Eclipse an ideal platform for tools to interplay.

We believe that the above facts about component-based architecture, together with features of Eclipse described above, make Eclipse a vehicle to tackle previously mentioned issues and present solutions in a simple and elegant form. The support for dealing with various concepts such as programming languages (Java and C++), architecture implementations (compilers), infrastructures (projects and workspaces), etc., in a seamless and extensible manner under one umbrella is another reason to choose Eclipse. Given such a platform, a new tool that is provided as a plug-in to the platform can be inducted into the main development cycle rather easily. These along with native IDE features of Eclipse are extensively used in Cadena to deliver a development environment for component-based applications.

[3]Hereon, we will use plug-ins to refer to extensions and plug-ins.

3 Cadena

In this section, we provide the details of Cadena in terms of artifacts that make up a project under Cadena along with the features of Cadena that aid in the development of these artifacts. We intersperse the presentation with Eclipse projects and platform entities that have been used to realize the feature.

3.1 Artifacts

A component-based application is made up of a collection of files of various types. As mentioned in Section 2.1, there are `idl` files written in *CORBA IDL* that describe the interface of the components. Each such component definition can then be augmented with some minimal implementation details. Such details are described in `cidl` files via *CORBA CIDL*. Hence, it is possible to combine an `idl` file with many `cidl` files. These artifacts are used to generate `java` files in which business logic needs to be filled in by the developer. Also, there are two descriptor files (`csd, ccd`) which are crucial in deployment of the component. These are the only artifacts that are critical to the component developer(designer and implementor).

We have recognized that it is possible for the developer to express abstractions of the behavior of components in an implementation independent manner by referring only to entities exposed in the artifacts listed above. This is accomplished allowing developers to create light-weight behavioral descriptions via a simple notation that we have developed called the *Component Property Specification (CPS)* language. At present, this language enables

the developer expose the dependency between various ports, facets and receptacles of a component. Through the `cps` file, it is possible to specify dependency in a component mode-specific manner. For example, it is important in some of applications to be able to specify that when a component is in a particular mode (i.e., state), it only responds to events on a particular subset of its event ports.

As described earlier, these defined components are assembled into a system by an assembler in XML-based *CORBA CAD* (`cad`) files. Instead of providing a XML editor to edit such file, we have opted for a human readable language in Cadena to describe such assemblies. The language is general enough to represent all the properties that can be captured in `cad` files, hence, the translation in either direction is trivial. We refer to the proprietary assembly language as *Cadena CAD* language and corresponding files as `scenario` files.

Depending on the type of backend used to generate the implementation of the defined system, Cadena generates various deployment files compatible with that backend. Currently, Cadena can generate an assembly script in Java, `cad` files which are CORBA compliant, and assembly descriptors which are *Boeing*[4] component model specific.

3.2 Tying them together

Cadena provides a project wizard in Eclipse to create a project with a pre-defined structure in which the above mentioned artifacts can be managed. Cadena provides *at least* one editor for each artifact described in the previous section. Separate editors is dedicated to `idl, cidl,` and `cps` files is provided. All text editors provided in Cadena are provide syntax highlighting.

The `idl` editor uses an interface repository via standard CORBA interfaces to type check the interface description. Hence, the developer can use any interface repository (remote/local) when developing a component. Like the `idl` editor, the `cps` editor also uses an interface repository to type check property descriptions of a component.

One of the most sophisticated features of Cadena is its support for assembling systems from individual components. Cadena provides a multi-page editor for `scenario` files that presents three views/editors of the same assembly on different pages. The text editor can be used to edit the `scenario` files via the keyboard. The user is bound to make errors in this view as he/she has to manually ensure the type safety of the assembly in terms of the types of the ports, facets, and receptacles being connected.

The above situation can be avoided by using the form-based editor. This editor allows the user to view the assembly as a spread-sheet of components and connection between components. The user can add/change/delete components and/or connections. The editor ensures syntactic and semantic correctness of the assembly in terms of type-safe connections and unique component identifiers. This is done by using the interface repository to provide the user with only valid choices when he/she wishes to modify the assembly. For example, if the user wants to change the end of a connection, he/she will be provided with a list of ports which can accept the connection and conform to the type of the port on the other end of the connection. There are few analyses that can synthesize information such as dispatch rates and distribution based on minimum information available in the assembly.

In addition to the text and spreadsheet system assembly views, Cadena provides a graphical assembly view implemented using *Graphical Editor Framework(GEF)*, an Eclipse tool project. This view provides multiple analyses that utilize the associated `cps` specification along with the assembly description to provide information intuitive while dealing with graphs. These include detection of cycles in connections and forward and backward slicing along port connections. For example, this latter feature allows the developer to select a particular port and then have the analysis find all "downstream" components and ports that are affected by an action on that port. The results of the analyses are indicated graphically by changing the color of displayed elements.

As for the issue of translation of these artifacts to machine readable artifacts, Cadena provides translators from proprietary languages to machine readable and/or standard specific languages. For translations beyond this Cadena uses compilers available with various CCM implementations. Hence, once configured, the user can drive the compilers via graphical user interface. At present, Cadena supports *OpenCCM*[4](a Java implementation of CCM) and *CIAO*[5] (a C++ implementation of CCM) as the CCM implementations which can be driven from Cadena. We have defined a generic

[4]Please refer to section 4 for more info.

minimal API via which various backends can be integrated into Cadena.

4 Real World Experience

Thus far, the Boeing Bold Stroke test-bed has been our primary "real-world" application. Boeing researchers have been applying Cadena to various representative systems since March 2003, and we have made several minor releases with additional features requested by Boeing engineers. As part of the DARPA PCES project, we have developed an extensive collection of metrics in conjunction with Boeing engineers that are being used to assess the benefits of Cadena in the Bold Stroke development process.

We have also begun interactions with researchers at *Rockwell-Collins Advance Technology Center* where there is significant interest in using middleware and component-based systems in various avionics applications. Our current efforts now are focusing on assessing the extent to which aspects of the Joint Tactical Radio System (JTRS) can be modeled in Cadena. The JTRS program is having a significant influence on the design of component deployment infrastructure for embedded systems.

5 Conclusions

There are a number of directions that we are pursuing in future development of Cadena, and we mention three of those here. First, are working on adding a meta-architecture framework by which collections of attributes and artifacts for particular product lines can be established. This will allow, for example, a Bold Stroke product-line profile to be defined which will automatically present to the developer interfaces for specifying real-time and platform attributes specific to the Bold Stroke development process. Second, we are working toward a deeper integration of Cadena with Bogor [6] – an extensible domain-specific model-checking environment that we have also developed in Eclipse. This deeper integration will include more sophisticated support for temporal property specifications, and for display of error trace information provided by the model-checker when a property violation is found. Finally, we are continuing our intensive interaction with researchers working on CIAO in order to provide high-level means of configuring CIAO and associated ACE/TAO layer to meet the requirements of a variety of distributed real-time embedded applications.

For more information about Cadena, including screenshots, tutorials, technical papers, presentations, and distribution information please visit `http://cadena.projects.cis.ksu.edu`.

About the Authors

Venkatesh Prasad Ranganath is a Ph.D. student at Kansas State University (KSU) working on static analyses and program transformations to customize and adapt Java programs. Adam Childs is M.S. student at KSU working on specification languages and analysis for component-based architectures. Jesse Greenwald is a Research Associate in the Santos Laboratory at KSU and is currently the primary developer of Cadena. Matthew Dwyer and John Hatcliff are Associate Professors at KSU working in the areas of static analysis, model-checking, and other forms of software verification. Gurdip Singh is a Professor at KSU working in the area of distributed systems, middleware, and protocol verification.

References

[1] "Enterprise JavaBeans Specification (version 2.0)." This specification is available at http://java.sun.com/products/ejb/index.html., Aug 2001.

[2] "CORBA Components." This specification is available at http://www.omg.org., Sept 2002.

[3] J. Hatcliff, W. Deng, M. B. Dwyer, G. Jung, and V. P. Ranganath, "Cadena: An integrated development, analysis, and verification environment for component-based systems," in *Proceedings of the 2003 International Conference on Software Engineering (ICSE'03)*, May 2003.

[4] "OpenCCM, a java implementation of ccm." This software is available at http://openccm.objectweb.org.

[5] "Component-integrated ace orb, a c++ implementation of ccm." This software is available at http://www.cse.wustl.edu/ nanbor/projects/CIAO/.

[6] Robby, M. B. Dwyer, and J. Hatcliff, "Bogor: An extensible and highly-modular model checking framework," in *Proceedings of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE'03)*, 2003.