

Static Component Configuration Support for Real-Time Platforms

Chris Gill and Venkita Subramonian
{cdgill,venkita}@cs.wustl.edu
Department of Computer Science
Washington University, St. Louis

Nanbor Wang
nanbor@txcorp.com
Tech-X Corporation
Boulder, CO

1 Introduction

Component based approaches [1] are gaining increasing momentum in the development of software systems. Implementations of component models like EJB [2], COM [3] and the CORBA Component Model (CCM) [4] provide standard frameworks which isolate the functional concerns of a software system from para-functional concerns like transactions, security, realtime, etc. Each of the above component models defines details of the component development process - implementation, packaging, deployment and configuration. The configuration of a component is separated from its implementation so that the same component can be run within different contexts. Apart from configuration, connection information between components is also specified so that components can be composed to realize the entire functionality of a system. In this position paper, we describe the component assembly process in the context of an open source implementation of the CORBA Component Model (CCM). We then describe extensions to component assembly to support the requirements of embedded real-time operating system (RTOS) platforms.

2 Dynamic Component Assembly

In this section, we describe the process of online assembly of components in CCM, as shown in Figure 1. The first stage of the our CCM system lifecycle occurs off-line, when component package (.csd) and assembly (.cad) files are generated by a modeling tool or other prior stage of the tool chain. These files contain an abstract specification of the configuration that is to be achieved by our CCM implementation in each particular deployment environment.

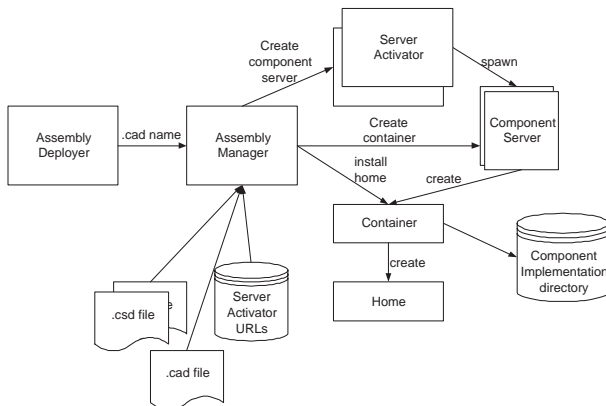


Figure 1: Online Component Assembly

Our implementation interprets these .csd and .cad files, and creates and configures the components, their run-time server environments, and QoS properties within the supporting ORB and other related infrastructure. The dynamic version of our CCM implementation currently runs several daemon processes for each deployment environment: one or more Component Installation/Server Activation (CISA) daemons on each machine where components can be deployed, an additional Assembly Manager daemon and an Assembly Deployer process used by the system developer.

The Assembly Manager stores an internal table with the target platform availability information. The Assembly Deployer tells the Assembly Manager which assemblies of components (each assembly is defined in a separate .cad file) should be deployed on which target machines. The Assembly Manager parses the XML structures in the .cad file, and generates its own internal data structure as an intermediate representation of that assembly. The Assembly Manager then traverses this intermediate representation, instructing each CISA daemon to install and configure specific component servers and containers, to create specific homes, and to instantiate specific component instances. Each CISA daemon has additional information about the component implementations available on that endsystem

The dynamic assembly of component applications suffers from the following drawbacks:

- XML parsing may be too expensive to be performed during system initialization.
- Multiple process address spaces may be required to coordinate the creation and assembly of components.
- Online loading of component implementation from DLLs or shared objects may not be supported by RTOS platforms like VxWorks [5] where such facilities are not available.

3 Static Component Assembly

To address the drawbacks of dynamic assembly approaches, we implemented an alternative approach to component assembly wherein as much work as possible in the assembly process is done offline. The fundamental intuition in understanding our approach is that in DRE systems the stages of the overall

system lifecycle are similar to those in more dynamic conventional component-oriented client-server applications. However, in our static configuration approach several phases of the CCM lifecycle are compressed into the compile-time and system-initialization phases, so that (1) for testing and verification purposes the set of components in an application can be identified and analyzed before run-time, and (2) overheads for run-time operation following initialization are reduced and made more predictable. Furthermore, due to the nuances of the platforms traditionally used for deploying DRE systems, not all features of conventional platforms are available. Our approach therefore avoids certain mechanisms that are either unavailable or too costly in terms of performance.

We follow these intuitions in our approach, taking the existing configuration phases in the online approach described in the previous section and pushing several of them earlier in the configuration lifecycle. We ensure that our approach can be realized in the context of platforms like VxWorks, by refactoring the configuration mechanisms and retargeting them to use only the services available on the target real-time platforms.

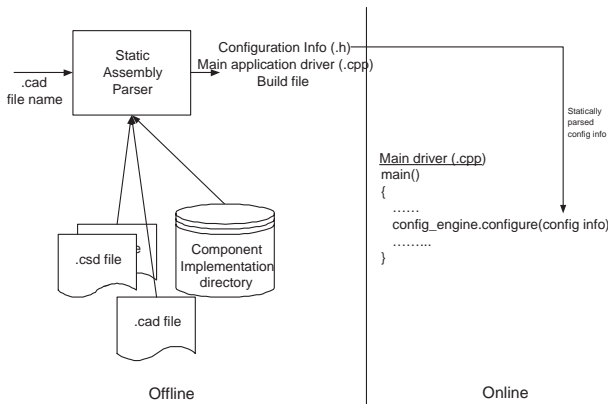


Figure 2: Static Component Assembly

In the static configuration approach, shown in Figure 2, configurations XML files are translated in a code generation step just before compile time (managed by the same projectMakefile processes that do the compilation) into C++ header and source files that are then compiled and linked with the main application. The result is that all such XML parsing has been moved off-line (before run-time), and the resulting information is linked statically into the application itself. Each endsystem boots and initializes in a single process address space, so that there is no need for inter-process communication to create and assemble components.

References

[1] C. Szyperski, *Component Software—Beyond Object-Oriented Programming*. Santa Fe, NM: Addison-Wesley, 1998.

[2] Sun Microsystems, “Enterprise JavaBeans Specification.” java.sun.com/products/ejb/docs.html, Aug. 2001.

[3] Microsoft Corporation, *Distributed Component Object Model Protocol (DCOM)*, 1.0 ed., Jan. 1998.

[4] Object Management Group, *CORBA Components*, OMG Document formal/2002-06-65 ed., June 2002.

[5] Wind River Systems, “VxWorks 5.3.” www.wrs.com/products/html/vxworks.html.