Toward a 1.0 specification of the Common Component Architecture for High Performance Computing

Benjamin A. Allan[1] and The Common Component Architecture Forum[2]

[1] Sandia National Laboratories, Livermore CA 94550, USA,
baallan@ca.sandia.gov,
http://www.cca-forum.org/~baallan
[2] The Common Component Architecture Forum,
cca-forum@cca-forum.org,
http://www.cca-forum.org/

Abstract. The Common Component Architecture (CCA) provides a generically structured, language independent framework for component-oriented software development in high-performance computing. Here we present an overview of the CCA specification and the status of emerging CCA extensions that are anticipated for version 1.0 of the CCA specification.

1 The CCA specification

The CCA specification provides a light-weight mechanism for a component instance to publish (provide) or obtain (use) interfaces (called ports) without directly contacting other components. Specification of inter-component connections is left to the framework user. Readers seeking a complete exposition at the code level are advised to consult the specification [1] and the tutorial [2].

The specification is intentionally silent on issues such as low level network protocols, the choice of parallel communication, the launching of parallel applications, and whether or not a given port must support any form of remote network access (TCP/IP is not sewed into the specification). All these issues are keys in achieving good parallel performance and the forum views no single solution as appropriate to require for all implementations of the specification. Each framework implementor is free to explicitly support or prohibit any particular form of inter-process or inter-thread communication. Thus, a web-service component may not be directly usable in a high-performance framework implementation.

1.1 Scientific middleware: language and network neutrality

The CCA specification is written in SIDL [3], an interface definition language intended to improve programming in all the primary languages of UNIX computing: C, C++, Python, Java, and FORTRAN. The de facto standard for SIDL is its implementation in the Babel tool [4], which has only recently included its first complete prototype of remote method invocation (RMI) to support distributed computing [5]. A full treatment of SIDL and Babel is beyond the scope of this paper; however, it may be described

as a tool for bringing a common high-performance array object representation and a Java-style separation of interfaces and implementations to all the supported languages.

The CCA design pattern and the core features of the specification (the uses-provides pattern [6]) were fixed before SIDL existed, so there are several legacy bindings of the CCA specification with on-going support [7, 8] and another alternative binding [9] planned. The CCA-Lite binding, in particular, will be geared toward exploiting the C interoperability features of the Fortran 2003 to reduce the weight of the middle-ware required for integrating C, C++, and Fortran components into a single application.

1.2 Components and self-description

In the CCA component model, components instances are stateful black-box entities which are self-describing at run-time. They may dynamically add or remove the interfaces (called *p*orts) they provide and similarly may dynamically choose which interfaces they will obtain and use via the framework. Also, components may publish parameter sets in a generic form used for run-time exchange with other components or framework agents.

Services: using and providing black-boxes The life-cycle of a component instance is managed by a framework. A component is first constructed as an empty husk object (of a specific class, of course) with no information about the environment in which it is running. The framework initializes the component by allocating a Services handle and inserting it in the component through a prescribed interface. Through this handle, the component describes its ports to the containing framework. The ports the component will ultimately use may not be immediately available unless they are automatically provided by the framework.

At the user's or other agent's direction, the framework makes connections between the ports of the various component instances if multiple components are present. Only after the needed connections have been made can a component use the ports it requested during initialization. Invoking the component assembly to process data may be done through the CCA-specified GoPort or through another port accessed through the framework by the driving code if the framework implementation permits such access.

When computations are completed (as indicated by the driver shutting down the framework instance containing the components), the component instances are destroyed. Those components requiring special destruction processing (for example to release cached port references or global resources or to log final messages) may register to be once again served their Services handle.

Parameters: dynamic control and reporting CCA uses a specific port type, the ParameterPort, to support the exchange of run-time parameter sets. These parameters may be scalars or 1-D arrays of any of the C99 primitive types, strings, or complex numbers, and are typically used to control optional behaviors of the components. Components wishing to exchange more complex objects must define and use more domain-specific interfaces. A parameter set is bundled into an object, TypeMap, which is exchanged through the ParameterPort interface. To reduce the rote coding work of the component

writer, the ParameterPortFactory is also specified. The ParameterPortFactory handles all the mechanics of providing a ParameterPort once the component writer defines the parameters, their defaults, and valid ranges.

1.3 Frameworks

The current component architecture is deliberately averse to making strong requirements of framework authors, as many different kinds of framework are optimal for various scientific computing scenarios. Among the many optional features are:

- Fault tolerance.
- Check-pointing.
- Remote method invocation.
- ParameterPort support.
- Multi-threaded or otherwise asynchronous behaviors.

Where frameworks are required to be similar is in supporting common interfaces for framing assemblies and sub-assemblies of components [10].

AbstractFramework The AbstractFramework interface is the common API for framework instances. Drivers should be able to easily plug-in alternative framework implementations if only using the AbstractFramework interface functions, provided of course the alternative framework supports the environmental requirements of the components in the application. A main program (including a GUI) may interact with the framework contents by registering itself as special kind of component, one that fetches its Services handle from the AbstractFramework rather than receiving the handle through the usual mechanism of implementing the Component interface. Once registered as a component, the driver manipulates the frame contents by using the BuilderService port, just as all other components in the frame may.

BuilderService The BuilderService is a minimalist interface intended for advanced programmers and graphic user interface developers. Any component may manipulate its own connections, or those of other components, may cause new peer component instances to appear in the same frame, or may perform any other of the most primitive application framing operation by requesting and using the BuilderService port from the framework. It may be used at any time in the running program. The BuilderService has been demonstrated [11] to allow automated self-tuning of applications by run-time component substitution.

2 Toward a 1.0 specification

A number of extensions to the CCA standard and environments are currently being investigated.

- Data-flow component framing support.

- A multi-level specification for events spanning the range from simple synchronous in-process call-backs to brokered, asynchronous, events over networks.
- An interface to simplify framing MPMD component applications, a detail currently left to the advanced MPI user.
- A standard interface to support interacting with multi-threaded GUI implementations, and a reference GUI implementation using it.

Most of these extensions may take the form of additional standard ports and standard component libraries. Additional, less complex, extensions are also under discussion by the Forum in [12], and at-large contributions are also solicited.

References

1. CCA Forum, Common Component Architecture Specification, `http://www.cca-forum.org/specification`, 2006.
2. CCA Forum, Common Component Architecture Tutorial at SC2005, `http://www.cca-forum.org/tutorials/archives/2005/tutorial-2005-11-14/index.html`, 2006.
3. Scott Kohn, Gary Kumfert, Jeff Painter, and Cal Ribbens, Divorcing Langauge Dependencies from a Scientific Software Library, in *Proc. 10th SIAM Conf. Parallel Process.*, Portsmouth, VA, 2001, Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-140349.
4. T. Dahlgren, T. Epperly, G. Kumfert, and J. Leek, *Babel User's Guide*, CASC, Lawrence Livermore National Laboratory, version 0.11.0 edition, 2006.
5. Jim Leek, Gary Kumfert, Tom Epperly, and Tamara Dahlgren, Babel RMI and You, `http://www.cca-forum.org/download/mtg/2006-01/BabelRMI-intro.pdf`, 2006.
6. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
7. Benjamin Allan, Common Component Architecture Specification, `https://www.cca-forum.org/wiki/tiki-index.php?page=CcafeInstallNeo`, 2006.
8. Keming Zhang, Kostadin Damevski, Venkatanand Venkatachalapathy, and Steven Parker, SCIRun2: A CCA Framework for High Performance Computing, in *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*, pages 72–79, IEEE Computer Society, 2004.
9. David Bernholdt, CCA and SciDAC2, `https://www.cca-forum.org/wiki/tiki-download_wiki_attachment.php?attId=21`, 2006.
10. D. E. Bernholdt, R. C. Armstrong, and B. A. Allan, Managing Complexity in Modern High End Scientific Computing through Component-Based Software Engineering, in *Proc. of HPCA Workshop on Productivity and Performance in High-End Computing (P-PHEC 2004), Madrid, Spain*, IEEE Computer Society, 2004.
11. H. Liu and M. Parashar, Enabling Self-Management of Component Based High-Performance Scientific Applications, in *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, 2005.
12. CCA Forum, CCA Specification Development, `https://www.cca-forum.org/wiki/tiki-index.php?page=CCA+Specification+Development`, 2006.