A BEAM DYNAMICS APPLICATION BASED ON THE COMMON COMPONENT ARCHITECTURE

Douglas Ricker Dechow, Dan Tyler Abell, Peter Stoltz, Tech-X Corp., Boulder, CO 80303, USA
Lois Curfman McInnes, Boyana Norris ANL, Argonne, IL 60439, USA
James Frederick Amundson, Fermilab, Batavia, IL 60510, USA

Abstract

A prototype of a component-based beam dynamics application has been developed. The Common Component Architecture (CCA) [5] software infrastructure was used to compose a new Python-steered, FODO-cell simulation from a set of beamline elements provided by MaryLie/IMPACT (ML/I) [1]. The prototyped FODO-cell simulation is preparatory work for a larger, ongoing effort to model collective effects using a component-based version of the Synergia2 [12] beam dynamics framework. Synergia2 coordinates a suite of modeling services provided by two separate beam dynamics packages (Impact [10] and Chef) and two high-performance computer science packages (PETSc [3] and FFTW [7]). The development of the proof-of-concept application was accomplished via the following tasks: 1) addressing multilanguage interoperability in the ML/I code with Babel; 2) creating the necessary components by making the selected software objects adhere to the CCA protocol; and 3) assemblying the components with a newly developed, Component Builder gui. The eventual, component-based beam dynamics application will allow the Synergia2 framework to evolve simultaneously with the modeling and simulation requirements of the International Linear Collider (ILC).

INTRODUCTION

The development process of the accelerator simulation community has traditionally been dominated by the production of monolithic *codes*. The DOE is seeking to support the development of high-performance scientific software packages that manage complexity while facilitating portability and interoperability. One way to address these issues that has been gaining popularity, in the larger computational science community, is the migration of applications towards the use of a scripting language interface to *steer* simulations [4]. The Synergia2 [12] application, a 3-D, parallel, particle-in-cell beam dynamics simulation toolkit, adopted a Python-steered simulation interface with the release of version two.

Another software technique to address these issues is through the practices of Component-Based Software Engineering (CBSE) [8]. Migrating a computational science application like Synergia2 to a scripting language-based, steering interface is dependent upon clean, well factored software interfaces. This was a process which we had already undertaken with Synergia2. As such, it was only natural that we also began to evaluate component-based techniques, since they have many of the same requirements.

COMMON COMPONENT ARCHITECTURE (CCA)

For the purpose of prototyping a component-based accelerator simulation, we have chose to use the Common Component Architecture (CCA). The CCA is a specification and a toolset for developing scientific software from interchangeable parts, components.

The most basic unit of interaction between CCA components is the *port*. A port is analogous to a method in a OO language or a function, procedure, or subroutine in a procedural langauge. CCA port relationships are defined by the Provides/Uses design pattern. As such, they come in two flavors:

1. A **providesPort** defines a protocol that will be supported by the component implementation.

2. A **usesPort** specifies that a component will interact with another component by means of the protocal described in its *providesPort*

CCA Tools

The primary CCA tools that were used to create the prototype FODO-cell simulation are the following:

- Scientific Interface Definition Language (SIDL)
- Babel [9, 6]
- CCAFFEINE [2]

Connections between a component offering up a *providesPort* and a requesting component with a corresponding *usesPort* are managed by a component framework. The component framework is also used to provide runtime infrastructure to support a comoponent-based simulation. For this project, we used the CCA-compliant CCAFFEINE framework. Ports and their relationships are encoded in SIDL. SIDL is the programming language that the Babel tool uses to address the issue of language interoperability between components implemented in different languages.

Multi-language Interoperability

Because Synergia2 is a hybrid software application that is comprised of beam dynamics framework2, high-performance computational science libraries, and a steered

Table 1: The programming languages used by the Synergia2 scientific packages and libraries.

Packages	Python	C++	C	F90	F77
Synergia2-Simulations	X				
Synergia2-New Solvers		X			
Synergia2-IMPACT				X	
Synergia2-Chef		X			
MaryLie/IMPACT				X	X
PETSc			X		

simulation interface, tools which are written in several different languages, multi-language interoperability is of great interest to our current and planned work. Table 1 contains a matrix associating the primary Synergia2 project packages and libraries and their respective implementation languages.

OASCR has funded the Babel project as a tool for addressing multi-language interoperability problems. Unlike other language interoperability solutions, Babel specializes in high performance computing and in Fortran 77/90 support. Babel makes use of CHASM [11] to provide Fortran 77/90 support.

The Babel compiler generates glue-code for each supported language based upon user-defined types that are described in SIDL. The Babel runtime library provides basic facilities and infrastructure to keep the model consistent. At present, Babel supports Fortran 90, Fortran 77, Python, Java, C, and C++.

One of the primary reasons for examining CBSD practices was a desire to use the Babel tool to handle the creation of our necessarily wide-range of language bindings.

CCA/SYNERGIA PROTOTYPE APPLICATION

Our initial target for a component-based beam dynamics simulation was the basic unit of modern accelerator design, the FODO-cell. A FODO-cell lattice requires the ability to calculate transfer maps for only three types of beamline elements: a focusing quadrupole, a drift, and a defocusing quadrupole. Transfer maps from each of these beamline elements can be obtained in a simple, linear format in the MaryLie/IMPACT [10] application.

The *BeamOpticsPort* encapsulates services related to applying transfer maps to the simulation particles in the beam bunch. It is a one-to-one mapping of the ML/I routines that are implemented by the component. Shown below is the interface described in SIDL.

```
interface BeamOpticsPort extends gov.cca.Port
{
void fquad3(in double l,
     in double gb0,
     inout array<double,1,column-major> h,
     inout array<double,2,column-major> mh);

void dquad3(in double l,
     in double gb0,
     inout array<double,1,column-major> h,
     inout array<double,2,column-major> mh);

void drift3(in double l,
     inout array<double,1,column-major> h,
     inout array<double,2,column-major> mh);
}
```

Only the beamline magnetic elements that were necessary to create the FODO-cell simulation were described in the interface.

In order to execute a CCA-based simulation, it is necessary to create a driver component. The driver component is required to register/deregister the necessary ports, to create the appropriate connections between the ports, and to initiate the simulation. The port manipulation routines take place in the *setServices()* method of the driver component.

For the purposes of the FODO-cell simulation, the driver component was named *Syn2ProtoDriverComp*. An example of how to register a *usesPort*, taken from *Syn2ProtoDriverComp*, is shown below:

```
# Register uses port
try:
  services.registerUsesPort(
                'BeamOpticsPort',
                'Synergia.BeamOpticsPort',
                mymap);
except:
  print sys.exc_type,
        sys.exc_info()
  print 'exception registerUsesPort()'
```

In order to register components with the CCAFFEINE framework and execute a simulation, it is necessary to create and run an rc script. An example of the rc script that was used for the prototype is shown below:

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this line.--
path set /home2/cca/techx/aas-cca/components/lib
path set /home2/cca/techx/mli/cca/components/lib

repository get-global SynergiaCCA.Syn2ProtoDriverComp
instantiate SynergiaCCA.Syn2ProtoDriverComp
            SynergiaCCASyn2ProtoDriverComp

repository get-global Synergia.BeamOptics
instantiate Synergia.BeamOptics SynergiaBeamOptics

connect SynergiaCCASyn2ProtoDriverComp BeamOpticsPort
        SynergiaBeamOptics BeamOpticsPort
```

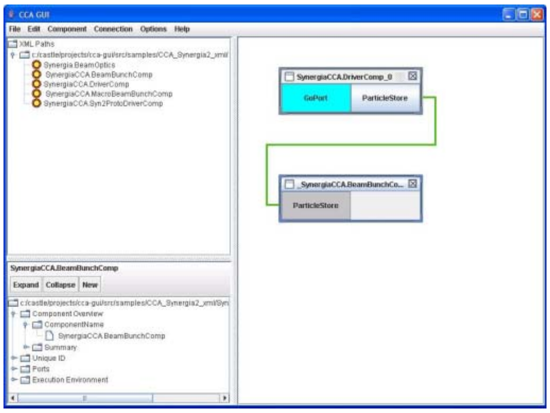
CCA GUI
File
Edit
Component
Connection
Options
Help
XML Paths
c:/castle/projects/cca-gui/src/samples/CCA_Synergia2_xml/
Synergia.BeamOptics
SynergiaCCA.BeamBunchComp
SynergiaCCA.DriverComp
SynergiaCCA.MacroBeamBunchComp
SynergiaCCA.Syn2ProtoDriverComp
SynergiaCCA.DriverComp_0
GoPort
ParticleStore
_SynergiaCCA.BeamBunchCo...
ParticleStore
SynergiaCCA.BeamBunchComp
Expand
Collapse
New
c:/castle/projects/cca-gui/src/samples/CCA_Synergia2_xml/Syn
Component Overview
ComponentName
SynergiaCCA.BeamBunchComp
Summary
Unique ID
Ports
Execution Environment

Figure 1: The Component Builder GUI.

```
go SynergiaCCASyn2ProtoDriverComp GoPort

quit
```

Currently under development is a GUI-based tool for creating component simulations. The Component Builder GUI will served as an alternative to writing CCAFFEINE rc scripts by hand. The Component Builder Gui is shown in figure 1.

The *go* command in the rc script sends a message to the *go()* method in the driver component. In the case of our FODO-cell simulation, the *go()* method implements a straightforward algorithm of invoking the ML/I beamline element subroutines through the component interface. An example of this is demonstrated below (only the most salient pieces of Python code are shown):

```
# Get a handle to the BeamOpticsPort
bmprt = BeamOpticsPort.BeamOpticsPort(gnport);

# Initialize the BeamOpticsPort
result = bmprt.initialize()

# Apply the fquad3 transfer map
(h,mh) = bmport.fquad3(l,gb0,h,mh)
```

FUTURE WORK

The ultimate aim of the project is to create a componentized version of Synergia2. This will allow us to flexibly create new collective effects-based acceleratio simulations from a wide-range of component services. The prototype effort to create a FODO-cell lattice simulation from an ML/I component has been deemed successful, and we are moving ahead with our broader componentization effort.

REFERENCES

[1] A. Dragt et al., MARYLIE 3.0 Users Manual, University of Maryland, Physics Department Report (1999).

[2] B. Allan, R. Armstrong, S. Lefantzi, J. Ray, E. Walsh, and P. Wolfe. Ccaffeine – A CCA component framework for parallel computing. `http://www.cca-forum.org/ccafe/`, 2005.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.0, Argonne National Laboratory, 2001.

[4] D. M. Beazley and P. S. Lomdahl. Building flexible large-scale scientific computing applications with scripting languages. 1997.

[5] D. E. Bernholdt, B. A. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou. A component architecture for high-performance scientific computing. *Intl. J. High Perf. Comp. Appl.*, 20(2):163–202, 2006.

[6] T. Dahlgren, T. Epperly, and G. Kumfert. *Babel User's Guide*. CASC, Lawrence Livermore National Laboratory, version 0.10.8 edition, July 2005.

[7] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. In *Proceedings of the IEEE*, pages 216–231, 2005. Invited paper, Special Issue on Program Generation, Optimization, and Platform Adaptation.

[8] G. T. Heineman and W. T. Councill, editors. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[9] Lawrence Livermore National Laboratory. Babel. `http://www.llnl.gov/CASC/components/babel.html`, 2007.

[10] J. Qiang, R. D. Ryne, S. Habib, and V. Decyk. An Object-Oriented Parallel Particle-in-Cell Code for Beam Dynamics Simulation in Linear Accelerators. *Journal of Computational Physics*, 163:434–451, Sept. 2000.

[11] C. E. Rasmussen, M. J. Sottile, S. Shende, and A. D. Malony. Bridging the language gap in scientific computing: The Chasm approach. *Concurrency and Computation: Practice and Experience*, 2005. to appear.

[12] P. Spentzouris and J. Amundson. Simulation of the Fermilab Booster using Synergia. *Journal of Physics Conference Series*, 16:215–219, Jan. 2005.