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) assembling 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 language. 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 protocol 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 component-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.

<table>
<thead>
<tr>
<th>Packages</th>
<th>Python</th>
<th>C++</th>
<th>C</th>
<th>F90</th>
<th>F77</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synergia2-Simulations</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synergia2-New Solvers</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synergia2-IMPACT</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synergia2-Chef</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaryLie/IMPACT</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PETSc</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.

```java
interface BeamOpticsPort extends gov.cca.Port
{
    void fquad3(in double 1,
                 in double gb0,
                 inout array<double,1,column-major> h,
                 inout array<double,2,column-major> mh);
    void dquad3(in double 1,
                 in double gb0,
                 inout array<double,1,column-major> h,
                 inout array<double,2,column-major> mh);
    void drift3(in double 1,
                 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:

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

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:

```bash
#!/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
SynergiaCCA.Syn2ProtoDriverComp
repository get-global Synergia.BeamOptics
instantiate Synergia.BeamOptics SynergiaBeamOptics
connect SynergiaCCA.Syn2ProtoDriverComp BeamOpticsPort
SynergiaBeamOptics BeamOpticsPort
```

D05 Code Developments and Simulation Techniques
Currently under development is a GUI-based tool for creating component simulations. The Component Builder GUI will serve 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):

```python
# 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 accelerator 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


