



# Introduction to Components

**CCA Forum Tutorial Working Group**

<http://www.cca-forum.org/tutorials/>  
[tutorial-wg@cca-forum.org](mailto:tutorial-wg@cca-forum.org)



## Overview

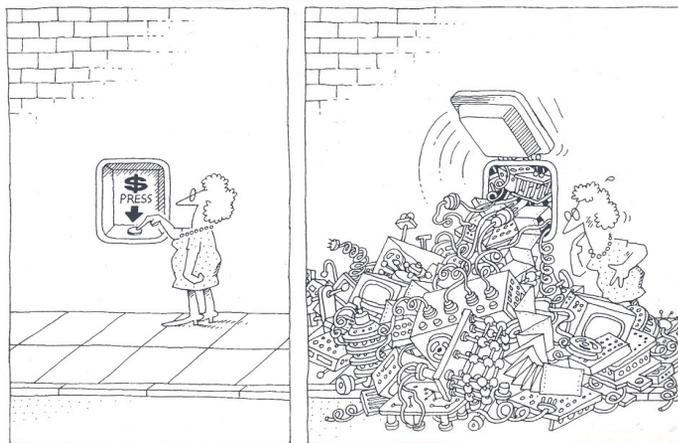
- **Why** do we need components?
- **What** are components?
- **How** do we make components?

## Why Components

- In “Components, The Movie”
  - Interoperability across multiple languages
  - Interoperability across multiple platforms
  - Incremental evolution of large legacy systems (esp. w/ multiple 3rd party software)
- Complexity

3

## Why Components



The task of the software development team is to engineer the illusion of simplicity [Booch].

4

## Software Complexity

- Software crisis
  - “Our failure to master the complexity of software results in projects that are late, over budget, and deficient in their stated requirements.” [Booch]
- Can't escape it
  - “The complexity of software is an essential property, not an accidental one.” [Brooks]
- Help is on the way...
  - “A complex system that works is invariably found to have evolved from a simple system that worked... A complex system designed from scratch never works and cannot be patched up to make it work.” [Gall]
  - “Intracomponent linkages are generally stronger than intercomponent linkages.” [Simon]
  - “Frequently, complexity takes the form of a hierarchy.” [Courtois]

5

## The Good the Bad and the Ugly

- An example of what can lead to a crisis in software:
- At least 41 different Fast Fourier Transform (FFT) libraries:
  - see, <http://www.fftw.org/benchfft/doc/ffts.html>
- Many (if not all) have different interfaces
  - different procedure names and different input and output parameters
- SUBROUTINE FOUR1(DATA, NN, ISIGN)
  - Replaces DATA by its discrete Fourier transform (if ISIGN is input as 1) or replaces DATA by NN times its inverse discrete Fourier transform (if ISIGN is input as -1). DATA is a complex array of length NN or, equivalently, a real array of length 2\*NN. NN MUST be an integer power of 2 (this is not checked for!).

6

## Components Promote Reuse



Hero programmer producing single-purpose, monolithic, tightly-coupled parallel codes

- Components promote software reuse
  - “The best software is code you don’t have to write”  
*[Steve Jobs]*
- Reuse, through cost amortization increases software quality
  - thoroughly tested code
  - highly optimized code
  - improved support for multiple platforms
  - developer team specialization

7

## What Are Components

- **Why** do we need components?
- **What** are components?
- **How** do we make components?

8

## What Are Components [Szyperski]

- A component is a binary unit of independent deployment
  - well separated from other components
    - fences make good neighbors
  - can be deployed independently
- A component is a unit of third-party composition
  - is composable (even by physicists)
  - comes with clear specifications of what it requires and provides
  - interacts with its environment through well-defined interfaces
- A component has no persistent state
  - temporary state set only through well-defined interfaces
  - throw away that dependence on global data (common blocks)
- Similar to Java packages and Fortran 90 modules (with a little help)

9

## What Does This Mean

- So what does this mean
  - Components are “plug and play”
  - Components are reusable
  - Component applications are evolvable

10

## Component Forms [*Cheesman & Daniels*]

- Component Standard
  - must conform to some sort of environment standard (Framework)
- Component Specification
  - specification of what a component does
- Component Interface
  - specification of procedure names and procedure parameters
- Component Implementation
  - written in a computer language (Fortran for example)
- Installed Component
  - a shared object library (.so file)
- Component Object
  - services and state joined together

11

## What is a Component Architecture

- A set of standards that allows:
  - Multiple groups to write units of software (components)
  - The groups to be sure that their components will work with other components written in the same architecture
- A framework that holds and runs the components
  - And provides services to the components to allow them to know about and interact with other components

12

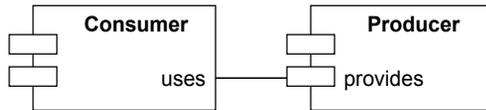
## What Are Components II

- Components live in **an environment** and interact with the environment through a framework and connections with other components.
- Components can **discover information** about their environment from the framework.
- Components must explicitly publish what capabilities they **provide**.
- Components must explicitly publish what connections they **require**.
- Components are a runtime entity.

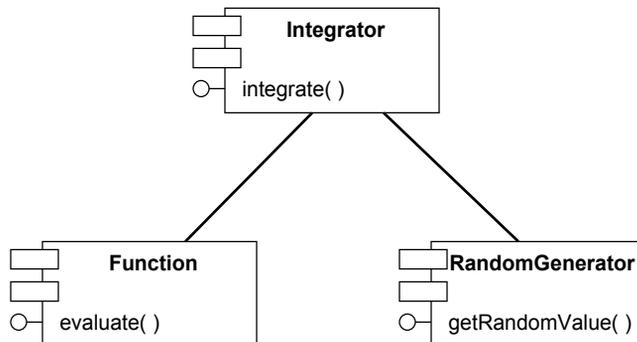
## Components Are Different From Objects

- You can build components out of object classes.
  - (or out of Fortran procedures)
- But a component is more than just an object.
- A component only exists in the context of a Component Standard (Framework).

## Pictorial Example



## Three Components





## Publish the Interface in SIDL

- Publish the interface
  - interfaces are published in SIDL (Scientific Interface Definition Language)
  - can't publish in native language because of language interoperability requirement
- Integrator example:

---

```
interface Integrator extends cca.Port
{
    double integrate(in double lowBound,
                    in double upBound,
                    in int count);
}
```

19

## F90 Integrator Interface

```
MODULE Integrator

interface

    !
    ! Returns the result of the integration from lowBound to upBound.
    !
    ! lowBound - the beginning of the integration interval
    ! upBound - the end of the integration interval
    ! count - the number of integration points
    !
    function integrate(port, lowBound, upBound, count)
        use CCA
        type(CCAPort) :: port
        real(kind(1.0D0)) :: integrate, lowBound, upBound
        integer :: count
    end function integrate

end interface

END MODULE Integrator
```

20

## F90 Program

```
program Driver
  use CCA
  use MonteCarloIntegrator
  type (CCAPort) :: port

  print *, "Integral = ", integrate(port, 0.0D0, 1.0D0, 1000)
end program
```

21

## C++ Abstract Integrator Class

```
/**
 * This abstract class declares the Integrator interface.
 */

class Integrator : public virtual gov::cca::port
{
public:
  virtual ~Integrator() { }

  /**
   * Returns the result of the integration from lowBound to upBound.
   *
   * lowBound - the beginning of the integration interval
   * upBound - the end of the integration interval
   * count - the number of integration points
   */
  virtual double integrate(double lowBound, double upBound, int count) = 0;
};
```

22

## C++ Object-Oriented Program

```
#include <iostream>
#include "MonteCarloIntegrator.h"

int main(int argc, char* argv[])
{
    MonteCarloIntegrator* integrator = new MonteCarloIntegrator();

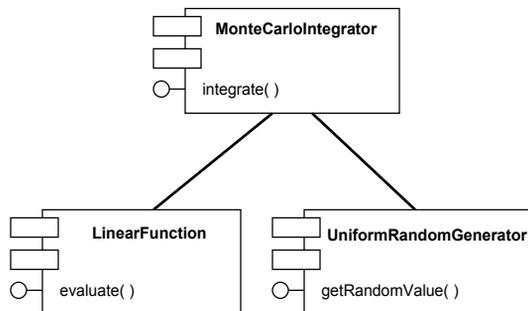
    cout << "Integral = " << integrator->integrate(0.0, 1.0, 1000) << endl;

    return 0;
}
```

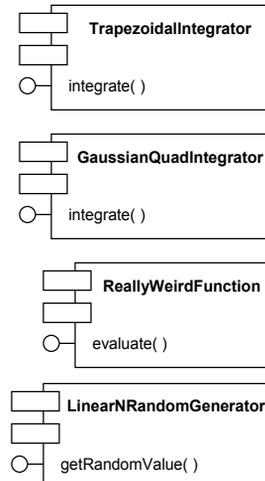
23

## Component Program

### Program



### Component Library



24

## Questions and Answers

- Is CCA similar to CORBA or COM/DCOM?
  - yes, but is a **component architecture** oriented towards high-performance computing
- Is CCA for parallel or distributed computing?
  - both, but currently only one or the other
- Can I use CCA today for scientific applications?
  - yes, but it is a research project
- Where can I get more information?
  - <http://www.cca-forum.org/>
  - join the CCA Forum

25

## Final Thought

- Components are reusable assets. Compared with specific solutions to specific problems, components need to be carefully generalized to enable reuse in a variety of contexts. Solving a general problem rather than a specific one **takes more work**. In addition, because of the variety of deployment contexts, the creation of proper documentation, test suites, tutorials, online help texts, and so on is more demanding for components than for a specialized solution. *[Szyperski, p. 14]*

26

## Bibliography

Booch, G. 1994. *Object-Oriented Analysis and Design with Applications*. Second Editions. Santa Clara, CA: The Benjamin/Cummings Publishing Company, p. 8.

Brooks, F. April 1987. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer* vol. 20(4), p. 12.

Cheesman, J. and J. Daniels. *UML Components: A Simple Process for Specifying Component-Based Software*. New York, NY: Addison-Wesley

Courtois, P. June 1985. On Time and Space Decomposition of Complex Structures. *Communications of the ACM* vol 28(6), p. 596.

Gall, J. 1986. *Systemantics: How Systems Really Work and How They Fail*. Second Edition. Ann Arbor, MI: The General Systemantics Press, p. 65.

Simon, H. 1982. *The Sciences of the Artificial*. Cambridge, MA: The MIT Press, p. 217.

Szyperski, C. 1998. *Component Software: Beyond Object-Oriented Programming*. New York, NY: Addison-Wesley, p. 30

Next: **CCA Concepts**