Status of SEAL

Contents

LCG Applications Area Meeting

22 July 2003

P. Mato / CERN

SEAL Versions Road Map

Release	Date	Status	Description (goals)
V 0.1.0	14/02/03 Released 14&26/02/03	internal	◆Establish dependency between POOL and SEAL ◆Dictionary generation from header files
V 0.2.0	31/03/03 Released 04/04/03	public	◆Essential functionality sufficient for the other existing LCG projects (POOL) ◆Foundation library, system abstraction, etc. ◆Plugin management
V 0.3.0	16/05/03 Released 23/05/03	internal	◆Improve functionality required by POOL ◆Basic framework base classes
V 1.0.0	30/06/03 Released 18/07/03	public	◆Essential functionality sufficient to be adopted by experiments ◆Collection of basic framework services ◆Scripting support

SEAL Team (credits)

Christian Arnault	(Dictionary)
Radovan Chytracek	(Foundation, Framework)
Jacek Generowicz	(Scripting, Framework, Documentation)
Fred James	(MathLibs)
Wim Lavrijsen	(Scripting)
Massimo Marino	(Foundation, Framework)
Pere Mato	(Framework, Dictionary, Scripting)
Lorenzo Moneta	(Foundation, Framework)
Stefan Roiser	(Dictionary)
RD Schaffer	(Dictionary)
Lassi Tuura	(Foundation, Framework, Infrastructure)
Matthias Winkler	(MathLibs)
Zhen Xie	(Dictionary)

Work Packages

1. Foundation and Utility libraries
2. Math Libraries
3. Component Model
4. LCG Object Dictionary
5. Basic Framework Services
6. Scripting Services
7. Grid Services
8. Education and Documentation

1. Foundation and Utility Libraries

- Foundation Packages

– SealPlatform	Platform dependent config.h file
– SealBase	~80 foundation classes
– SealIOTools	~30 I/O classes
– SealUtil	~5 utility classes
– SealZip	Compression utility classes
– PluginManager	Low level plug-in management classes
– PluginDumper	Program to dump the plug-in repository

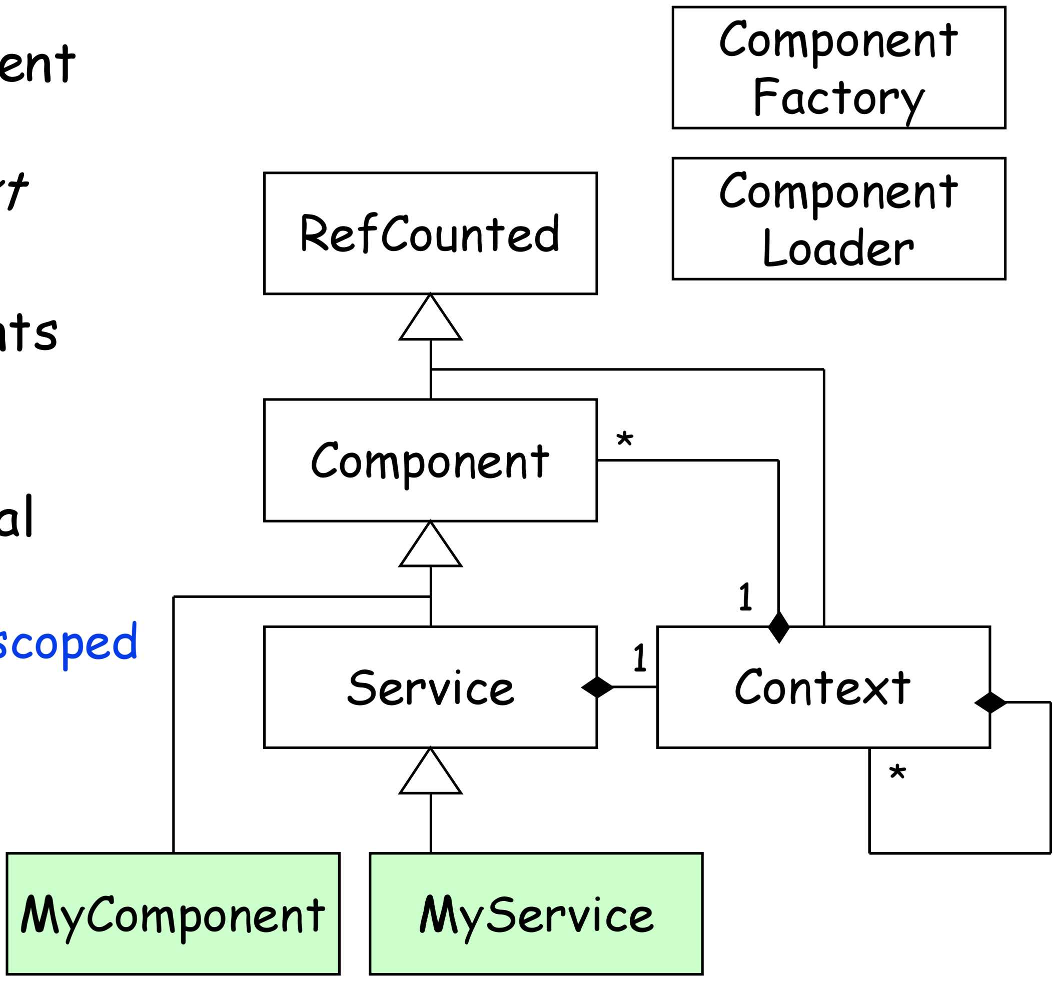
- Boost Library
  - In use in SEAL itself (utilities, shared pointers, format,…)
- CLHEP
  - Packages split ready to be released
  - New CVS repository (SPI provided)

2. Math Libraries

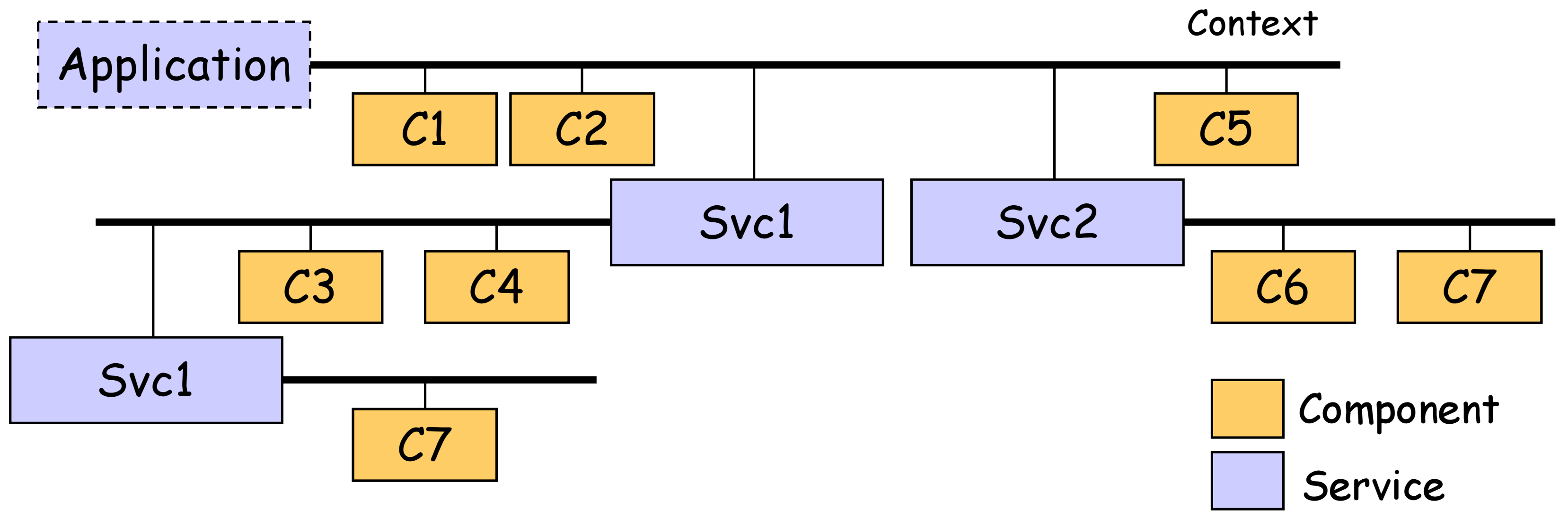
- MINUIT
  - API: adding user interfaces for parameters, migrad, minos
  - design: separate user parameters and user function, including change of function interface (FCN) to a minimal required interface
  - fixing/releasing of user parameters is now supported
  - the number of calls to the user function is counted
  - two examples added: MigradGaussSim.cpp, MinosGaussSim.cpp
- GSL
  - Not yet a public report of the evaluation. Agreement in recommending GSL to be used by experiments.

3. Component Model

- Designed a hierarchy of bases classes to support the component model
- Each *Component* is in a *Context*
- Tree of *Contexts*
- Support for locating components
  - If not in local context, look in parent
- A *Service* provides its own local *Context*
  - Components of a *Service* are scoped in its own *Context*
- User classes inherit from *Component* or *Service*
  - Plug-in functionality for free
- FEEDBACK REQUESTED !!



Context Hierarchy



- Any *Component* may attempt to locate another *Component* in the running application (collaboration network)
  - By "type" or by "key"
  - If the not found in the current context, the parent context is searched is recursively

Component Model: How-To (1)

MyComponent.h

```
#ifndef MYCOMPONENT_H
#define MYCOMPONENT_H 1
#include "SealKernel/Component.h“
class MyComponent : public seal::Component{
  DECLARE_SEAL_COMPONENT;
public:
  MyComponent (seal::Context* context);
  MyComponent (seal::Context* context, const std::string & label);
  // implicit copy constructor
  // implicit assignment operator
  // implicit destructor
  //.....component member functions..
  void doSomething();
};
#endif // MYCOMPONENT_H
```

Component Model: How-To (2)

MyComponent.cpp

```
#include "MyComponent.h“
#include <iostream>
DEFINE_SEAL_COMPONENT (MyComponent, "seal/example/MyComponent");
MyComponent::MyComponent (seal::Context* context)
    : Component (context, classContextKey ()){}
MyComponent::MyComponent (seal::Context* context,
                          const std::string & label)
    : Component (context, label){}
// member function implementations
void MyComponent::doSomething() {
 std::cout << "MyComponent:  Hello World ! " << std::endl;
}
```

Module.cpp

```
#include "MyComponent.h“
#include "SealKernel/ComponentFactory.h“
#include "PluginManager/ModuleDef.h“
DEFINE_SEAL_MODULE ();
DEFINE_SEAL_PLUGIN (seal::ComponentFactory, MyComponent,
                     MyComponent::classContextLabel ());
```

Component Model: How-To (3)

OtherComponent.cpp

```
#include “SealKernel/ComponentLoader.h”
#include “MyComponent.h”

Handle<MyComponent> handle = component<MyComponent>();
handle->doSomething();
```

4. Object Dictionary

- Reflection packages
  - Some minor bug fixes and improvements were applied to Reflection and ReflectionBuilder
- Dictionary generation
  - The *lcgdict* command for generating the dictionary sources issues some new warnings. Minor bug fixes.
- New Common Dictionaries
  - SealCLHEP. Dictionary for the Random and Vector subsystems of CLHEP (more dictionaries will be generated on request)
  - SealDict. Dictionary information of the Reflection package itself. Useful for working with the dictionary information with the python binding
  - SealSTL. Dictionaries for std::string and some instantiations of fundamental types for std::vector<T> and std::list<T> (more will be generated on request)

5. Basic Framework Services

- Developed the first set of Basic Services based on the new Component Model
- Application
  - Defines the top level *Context*
  - Possibility to set the initial set of *Components* to be loaded in the application
- Message Service
  - Message composition, filtering and reporting
- Configuration Service
  - Management of *Component* properties and loading configurations

Application

- Establishes the "top" *Context*
  - But, it can be inserted in an exiting *Context*
- Instantiates a basic number of Components (or Services) useful to all applications
  - *ComponentLoaded* (interface to Plug-in manager)
  - *PropertyManager* (application level configuration parameters)
  - *MessageService*
  - *ConfigurationService*

main.cpp

```
int main(int,char**) {
  Application theApp;
  //----Get Loader
  Handle<ComponentLoader> loader = theApp.component<ComponentLoader>();
  //----Instantiate the plug-in
  loader->load("SEAL/Kernel/Test/Loadable");
  //----Get a handle to it
  Handle<Loadable> loadable = theApp.component<Loadable>();
}
```

Message Service

- The user instantiates a *MessageStream* to compose messages. It reports to the *MessageService* when message is completed
- *MessageService* dispatches and filters all messages of the application

OtherComponent.cpp

```
#include “SealKernel/MessageStream.h”

MessageStream info( this,”MyName”, MSG::INFO);
info << “Hello world” << flush;

MessageStream log( this, “OtherName”);
log(MSG::ERROR) << “This is an error” << flush;
```

output

```
MyName     INFO    Hello World
OtherName  ERROR   This is an error
```

Configuration Service

- A *Component* may declare its own *Properties*
  - Templated *Property* instances (any type with a stream operator<<)
  - References to data members (any type with a stream operator<<)
  - Possibility to associate "callback" update function
  - Properties have a "name" (scoped) and a "description"
- The *PropertyCatalogue* is the repository of all properties of the application
  - It is filled from the "configuration file" (Gaudi JobOptions format currently)

Component.cpp

```
struct callObj { operator()(const Propertybase&) {…} };
int m_int;
Property<double> m_double("double", 0.0,"descr", callObj);
propertyManager()->declareProperty("int", m_int, 0, "descr");
propertyManager()->declareProperty(m_double);
```

6. Scripting Services

- Guidelines for developing Python bindings
  - Evaluated existing options: SWIG, Boost.Python, SIP, and raw Python C-API
    - » http://cern.ch/seal/work-packages/scripting/evaluation-report.html
- PyLCGDict: Python binding to the LCG Dictionary
  - It allows the user to interact with any C++ class for which the LCG Dictionary has been generated
  - With this module, there is no need to generate specialized Python bindings or wrapper code to interact with C++ classes

PyLCGDict: Supported Features

- Loading LCG dictionaries
- C++ classes
  - Mapped to Python classes and loaded on demand.
- C++ namespaces
  - Mapped to python scopes. The "::" separator is replaced by the python "." separator
- Class methods
  - Static and non static class methods are supported
  - Method arguments are passed by value or by reference
  - The return values are converted into python types and new python classes are created if required.
  - Method overloading works by dispatching sequentially to the available methods with the same name until a match with the provided arguments is successful.
- Class data members
  - Public data members are accessible as python properties
- Emulation of python containers
  - Container C++ classes (currently only *std::vector* like) are given the behavior of the python collections to be use in iterations and slicing operations.
- Operator overloading
  - Standard C++ operators are mapped to the corresponding python overloading operators

PyLCGDict: Example (1)

MyClass.h

```
#include <iostream>
namespace Example {
  class MyClass {
  public:
    MyClass() : m_value(0) {}
    MyClass(const MyClass& m ) : m_value(m.m_value) {}
    ~MyClass() {}
    int doSomething(const std::string&amp; something ) {
      std::cout << “I am doing something with “
                << something << std::endl;
      return something.size();
    }
    int value() { return m_value; }
    void setValue(int v) { m_value = v; }
  private:
    int m_value;
  public:
    float fprop;
    std::string sprop;
  };
}
```

PyLCGDict: Example (2)

```
> python2.2
Python 2.2.2 (#1, Feb 8 2003, 12:11:31) [GCC 3.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyLCGDict
>>> PyLCGDict.loadDict('MyClassDict')
>>> m = Example.MyClass()
>>> dir(m)
['__class__', '__delattr__', '__dict__', '__doc__', '__getattribute__', '__hash__',
'__init__', '__module__', '__new__', '__nonzero__', '__reduce__', '__repr__',
'__setattr__', '__str__', '__weakref__', '__zero__', '_theObject', 'doSomething',
'fprop', 'setValue', 'sprop', 'value']
>>> m.doSomething('care')
I am doing something with care
4
>>> m.setValue(99)
>>> print m.value()
99
>>> n = Example.MyClass(m) # copy constructor
>>> print n.value()
99
>>> n.fprop = 1.2
>>> n.sprop = 'String property'
>>> print n.sprop, n.fprop
String property 1.2
```

PyLCGDict: Example (3)

- Little script to dump the contents of a LCG dictionary

```
#!/usr/bin/env python2
import sys, string, PyLCGDict
PyLCGDict.loadDict('SealDictDict')
reflect = PyLCGDict.makeNamespace('seal::reflect')
#-----------------------------------------------------------------------
def dumpdict(dict = None):
  if dict : PyLCGDict.loadDict(dict)
  for c in reflect.Class.forNames() :
    print 'class',c.fullName()
    bases   = c.superClasses()
    if len(bases) : print '  inherits: ',
                    string.join(map(reflect.Class.fullName, bases),', ')
    fields  = c.fields(0)
    for f in fields :
      print '     ', f.typeAsString(), f.name()
    print ' '
```

8. Education/Documentation

- Existing documentation
  - Currently still limited to a number of topical How-To pages
- Python course
  - Originally prepared for ATLAS, is now part of CERN technical training program
  - First given in July, scheduled another one for August

Software Process

- Testing
  - Many more Unit Tests have been introduced in 1.0.0 release
  - Not yet all of them CppUnit or PyUnit
- Nightly Builds
  - Just started with NICOS (output already available)
- Platforms
  - Released platform: Linux RedHat 7.3/gcc-3.2 optimize and debug
  - Actively working on *icc* and *windows* releases
- Effort is being put to "standardize" tool usage and conventions
  - Avoiding divergences between LCG projects
  - Adaptation of tools to "LCG proper" working models
- Release procedures
  - Automation of the process is essential (rotating release manager role)

Next Immediate Steps

- Get feedback (from experiments + POOL+…) about Component model and Framework services
  - Corrections and re-designs are still possible
- Increase the number of supported platforms
  - Windows is urgent for LHCb
  - ICC and ECC
- Adapt SEAL to new SPI conventions and tools
- New Functionality
  - Framework: Start designing and implementing the *Whiteboard* service and *Dictionary* service
  - Scripting: Completion of PyLCGDict (more C++ features)
  - Dictionary: New design of the reflection model (complete C++ types)

Summary

- SEAL 1.0.0 is out with some delay (3 weeks)
  - Takes always longer than foreseen
  - Combining existing designs into a "common" one is not trivial
- The Highlights of 1.0.0 are:
  - First design and implementation of the SEAL component model (base classes to support the model provided)
  - First set of basic Framework services
  - New packages providing dictionaries for standard libraries
  - Python bindings for LCG dictionaries
- Ready to be used (tested) by experiments frameworks
  - LHCb is planning to do so in September