SEAL
Framework & Services

LCG AA Internal Review
20 October, 2003

Radovan Chytracek / CERN
on behalf of SEAL team

Motivation

- Allow LCG developers and users to write portable and modular code across all LCG projects
- Provide developers with handy abstractions for modeling software systems
- Learn from mistakes in the past and come with software component system which gets the best ideas which work well in existing systems

Requirements

- There are simply too many to list here
  - Too many parties and interests involved
  - Very often controversial between two parties
- Inspired by technologies in HEP and outside
  - Gaudi, Iguana, J2EE, Qt, COM, .NET, Python

- Basic requirements:
  - KISS, flexible, type-safe, fast(if possible)
- Features:
  - contextual composition, type-safe lookup, abstract interface support, plug-ins friendly

Requirements (2)

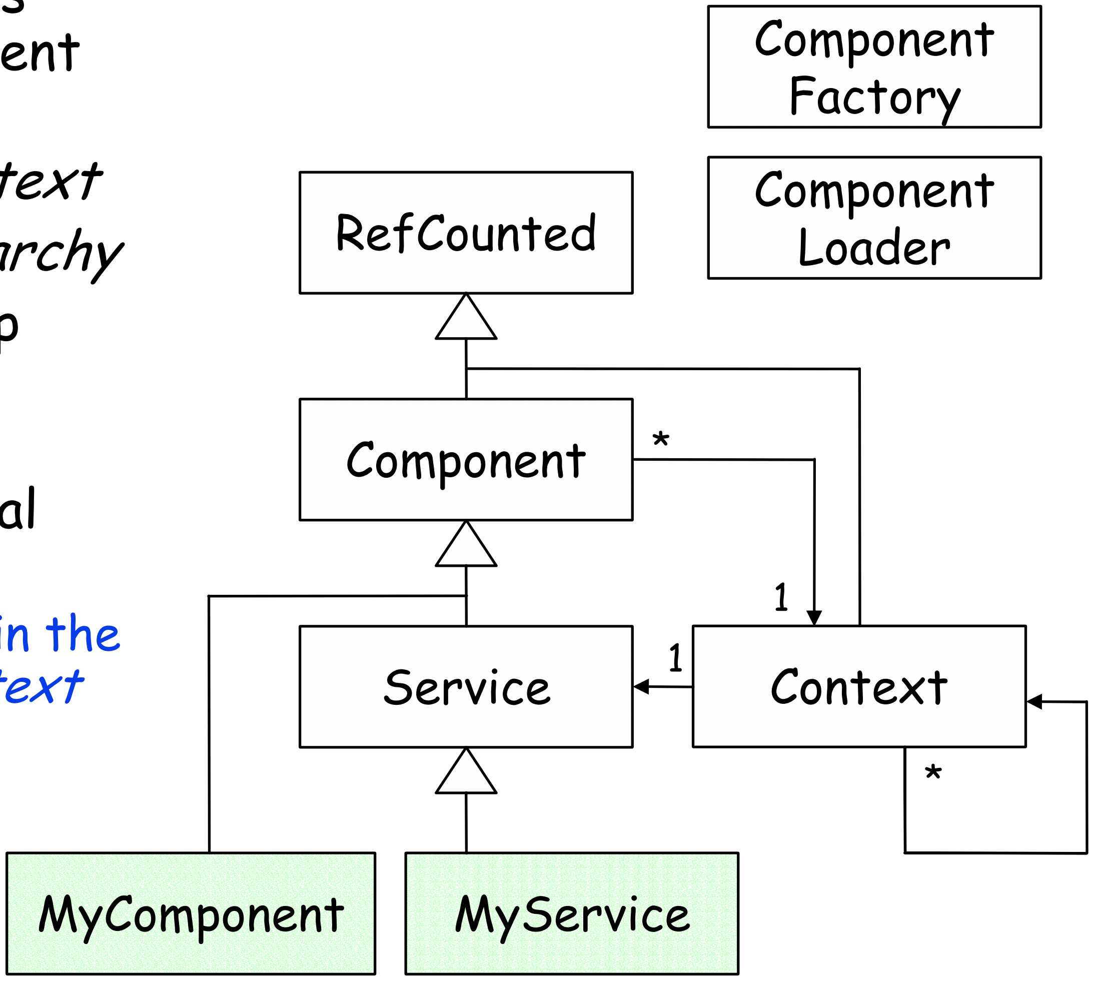
- Encourage well-known communication protocols
  - call-backs, signals/slots,, observer-notification
- Encourage simple services
  - direct use, targeted interfaces
  - configurable components via state/context instead of a->b->c->d style of navigation
- Aim at lightweight framework, simple, repeatable usage pattern
  - common wiring, not a lot imposed on experiment framework
- Component model must fit needs of majority and must be easily adopted by the rest

How we did it

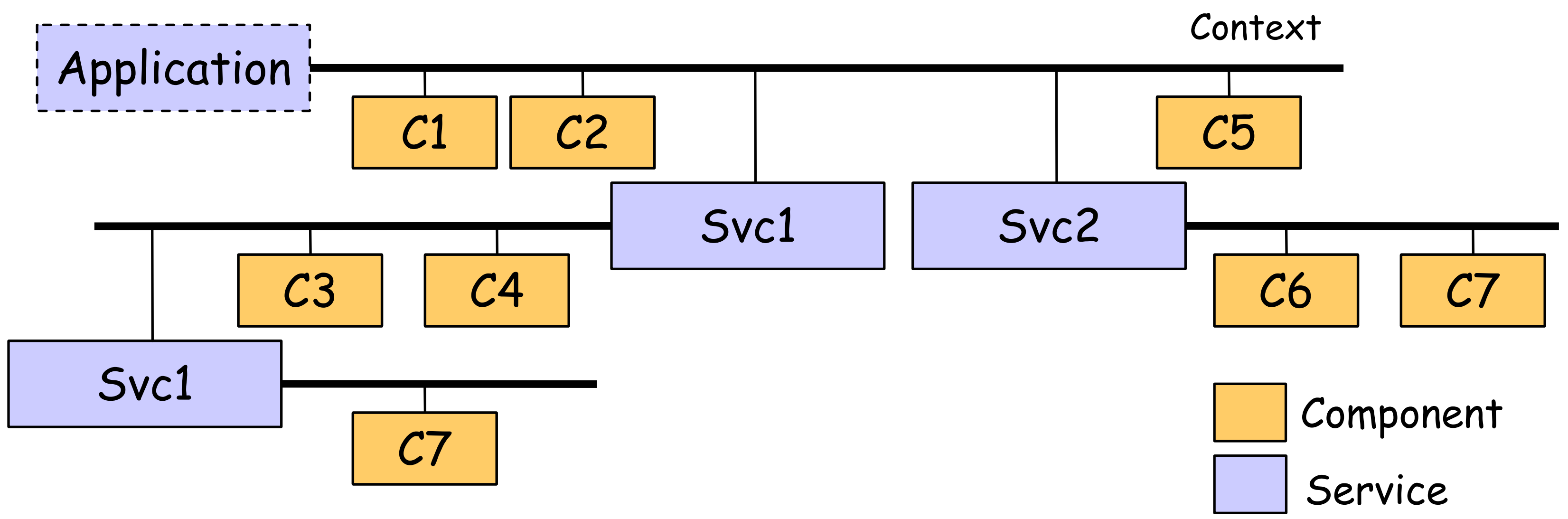
- Baseline taken from Iguana component system
  - Re-design followed to make it independent
- The design we agreed was first prototyped in Python
  - Focus on functionality & component collaboration apects
- 2 quick independent C++ prototypes followed
  - The first goal to achieve the Python prototype look-alike
  - The second to achieve minimal amount of code needed to build and use components in a C++ application
- Internal review of the prototypes
  - Benchmark, C++ implementations comparison, identifying performance bottle-necks and points of inflexibility

Component Model

- Designed as hierarchy of bases classes to support the component model
- Each *Component* lives in a *Context*
- *Contexts form their own hierarchy*
- Support for component look-up
  - If not in local context, look in parent
- A *Service* provides its own local *Context*
  - Components of a *Service* live in the scope defined by its own *Context*
- User classes inherit from *Component* or *Service*
  - Plug-in functionality for free



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 recursively

Component Model: How-To (1)

MyComponent.h

```
#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();
};
```

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

Look-up by type

```
#include "SealKernel/ComponentLoader.h"
#include "MyComponent.h"

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

OtherComponent1.cpp

Look-up by key

```
#include "SealKernel/ComponentLoader.h"
#include "MyComponent.h"

Handle<MyComponent> handle = component<MyComponent>(
                                        "seal/example/MyComponent2"
                                        );
handle->doSomething();
```

Component Model: How-To (4)

IFace.h

```
class IFace {
  public: virtual void ifaceCall(void)=0;
          virtual ~Iface(){}
};
```

IFaceComponent.cpp

```
class IFaceComponent : public IFace, public Component {
  public:
    DECLARE_SEAL_COMPONENT;
    IFaceComponent( Context* c ) : Component( c, IFaceComponent::classContextKey() ) {}
    IFaceComponent( Context* c, const ContextKey& key ) : Component( c, key ) {}
    IFaceComponent( Context* c, const std::string& key ) : Component( c, key ) {}
    virtual ~IFaceComponent() {}
  public: // IFace implementation
    virtual void ifaceCall(void);
};
DEFINE_SEAL_COMPONENT(IFaceComponent,"seal/example/IFaceComponent")
void IFaceComponent::ifaceCall(void) {
  MessageStream optimist( this, "Optimistic" );
  optimist << "I live!" << flush;
}
```

Using abstract interface

Main.cpp

```
#include "IFace.h"
int main()
{ Application theApp;
  Handle<ComponentLoader> loader = theApp.context()->component<ComponentLoader>();
  loader->load("seal/example/IFaceComponent");
  IHandle<IFace> ifacehandle =
    theApp.context()->query<IFace>("seal/example/IFaceComponent");
  if( ifacehandle )
    ifacehandle->ifaceCall();
  return 0;}
```

Basic Framework Services

- The first set of Basic Services came with 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
  - Closely related to MessageStream
- Configuration Service
  - Management of *Component* properties and loading configurations
  - Multiple backends foreseen:
    - » Gaudi style options, .INI style, CMS style, XML, ...

Future development

- New Services
  - *Whiteboard* service
    - » Design started and discussed with the interested experiments
    - » Waiting for feedback from experiments
    - » Implementation plan similar to that of component model
  - *DictionaryService* - loading of dictionary libraries on-demand
    - » Design and prototyping in progress
- More ConfigurationService back-ends
- Get feedback (from experiments + POOL+…) about Component model and Framework services
  - Corrections and re-designs are foreseen and possible

Summary

- Component model & basic services came with SEAL 1.0.0
  - Combining existing designs into a "common" one is not trivial
  - Base classes to support the model provided
  - First set of basic Framework services
- Ready to be used (tested) by experiments frameworks, however:
  - POOL is waiting for the component model to stabilize
  - LHCb may jump on it after POOL integration in Gaudi is O.K.
  - CMS similar attitude as LHCb, going for minimalist approach
  - ATLAS not there yet

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
```

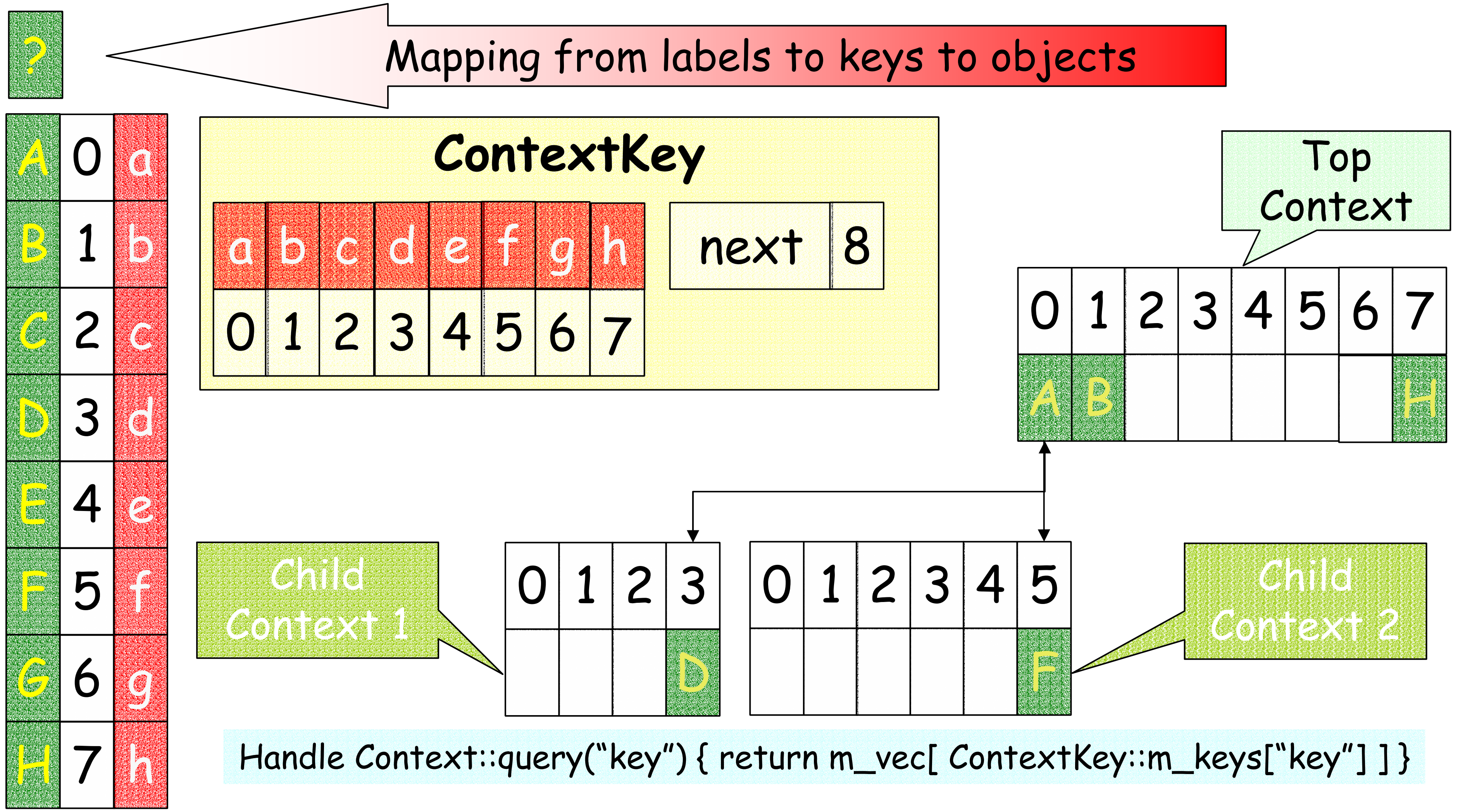
Configuration Service

- 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);
```

Single Context Look-up object model



```
Handle Context::query("key") { return m_vec[ ContextKey::m_keys["key"] ] }
```

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, Scripting)
◆ 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)