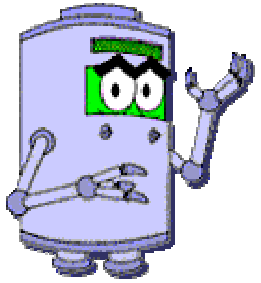
Current State of the OROCOS::SmartSoft Implementation

Christian Schlegel

FAW Ulm
Research Institute for Applied Knowledge Processing
Helmholtzstrasse 16
D-89081 Ulm
Germany

schlegel@faw.uni-ulm.de



Leuven
November 2002

OROCOS::SmartSoft

Current State

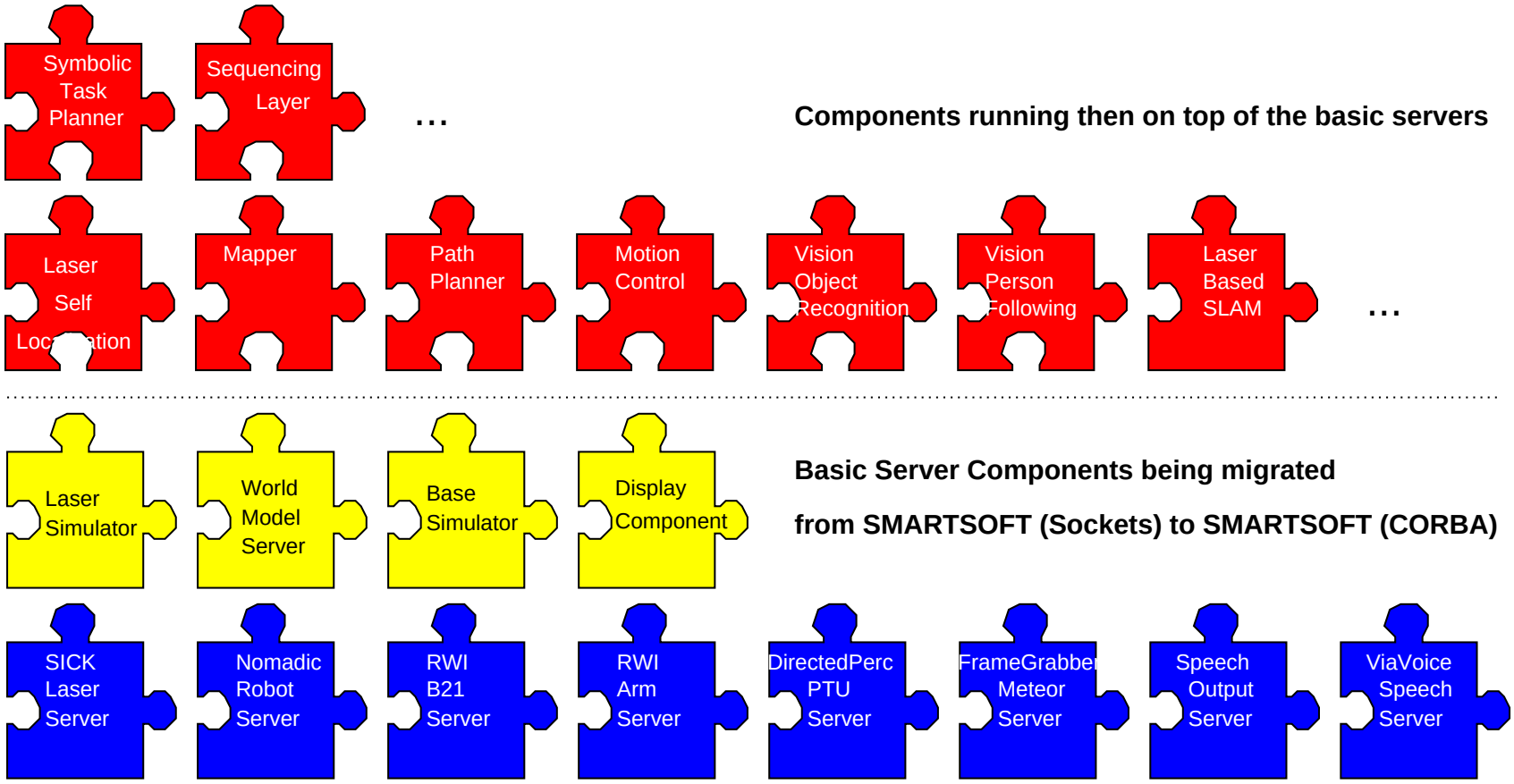
- Framework: *AVAILABLE*
  - ⋆ Communication Patterns
  - ⋆ Processing Patterns
  - ⋆ Examples and extensive Doxygen Documentation
  - ⋆ *Manuals to be extended*

Current Work

- Components: *in progress*
  - ⋆ SICK Laser Server
  - ⋆ B21 Base Server
  - ⋆ Nomad Base Server
  - ⋆ ...
- Communication Objects
  - ⋆ Currently migrated to OROCOS::SmartSoft

OROCOS::SmartSoft

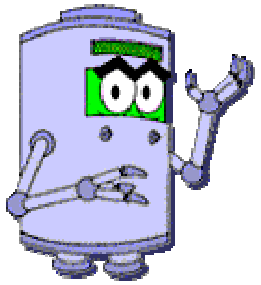
Components



OROCOS::SmartSoft

Talk

- Approach
  - ⋆ Components
  - ⋆ Communication Patterns
- Implementation
  - ⋆ CORBA / ACE / TAO
- Usage
  - ⋆ Examples



Basic Idea of OROCOS::SmartSoft

- Component based approach (NOT just distributed objects)

  Software Component A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be developed independently and is subject to composition by third parties. (Szyperski, WCOP, 1996)

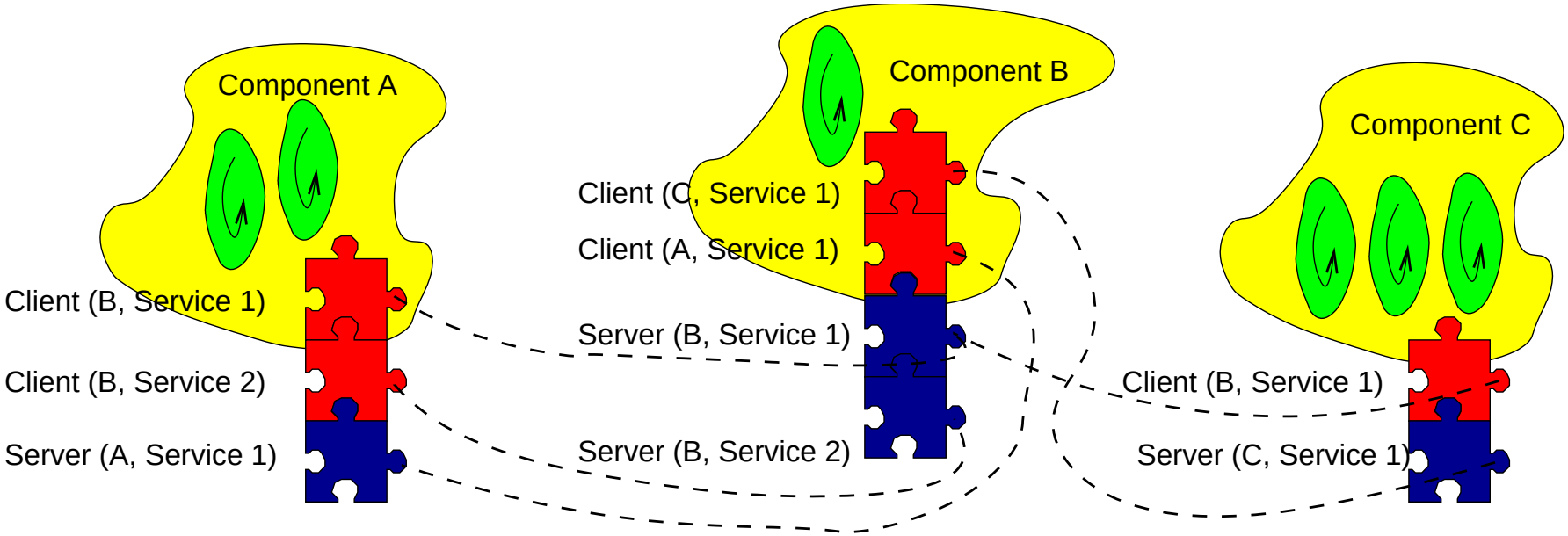
- Requirements
  - ⋆ Decoupling of Components
  - ⋆ Framework: Strict $\Longleftrightarrow$ Limiting ...
  - ⋆ End Users / Application Builders / Component Builders / Framework Builders
  - ⋆ ...

Basic Idea of OROCOS::SmartSoft

- Master Component Interactions but allow arbitrary Component internal Architecture
- Provide Communication Patterns

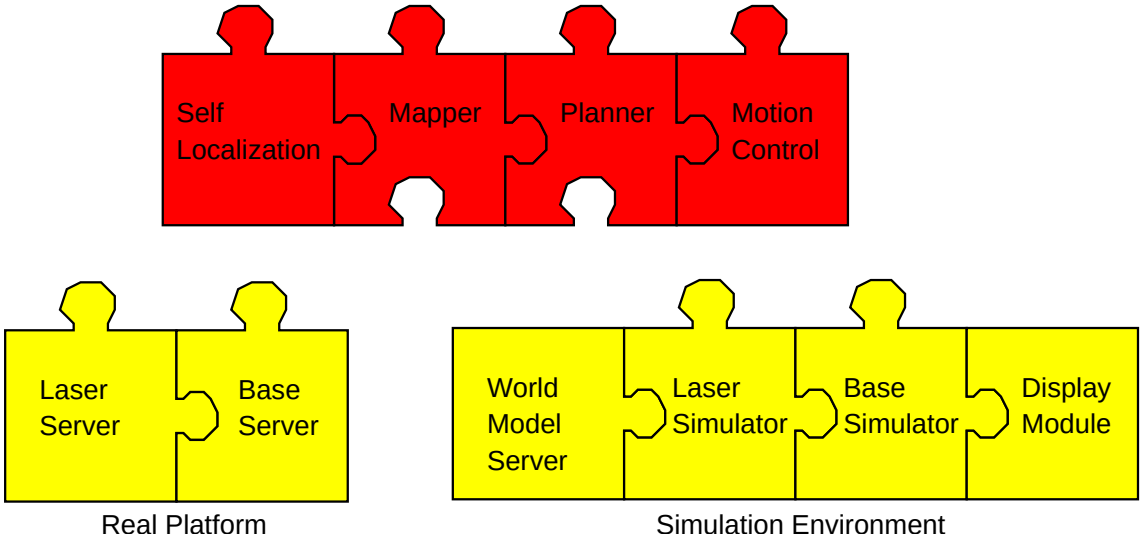
$\Longrightarrow$ Service based Approach: Map all Services on fixed communication patterns

Service = Mode (communication pattern) + Content (communication object)



Basic Idea of OROCOS::SmartSoft

- plug-and-play
- clear interface semantic
- different component internal architectures
- ...

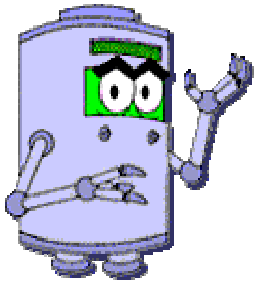


Communication Patterns

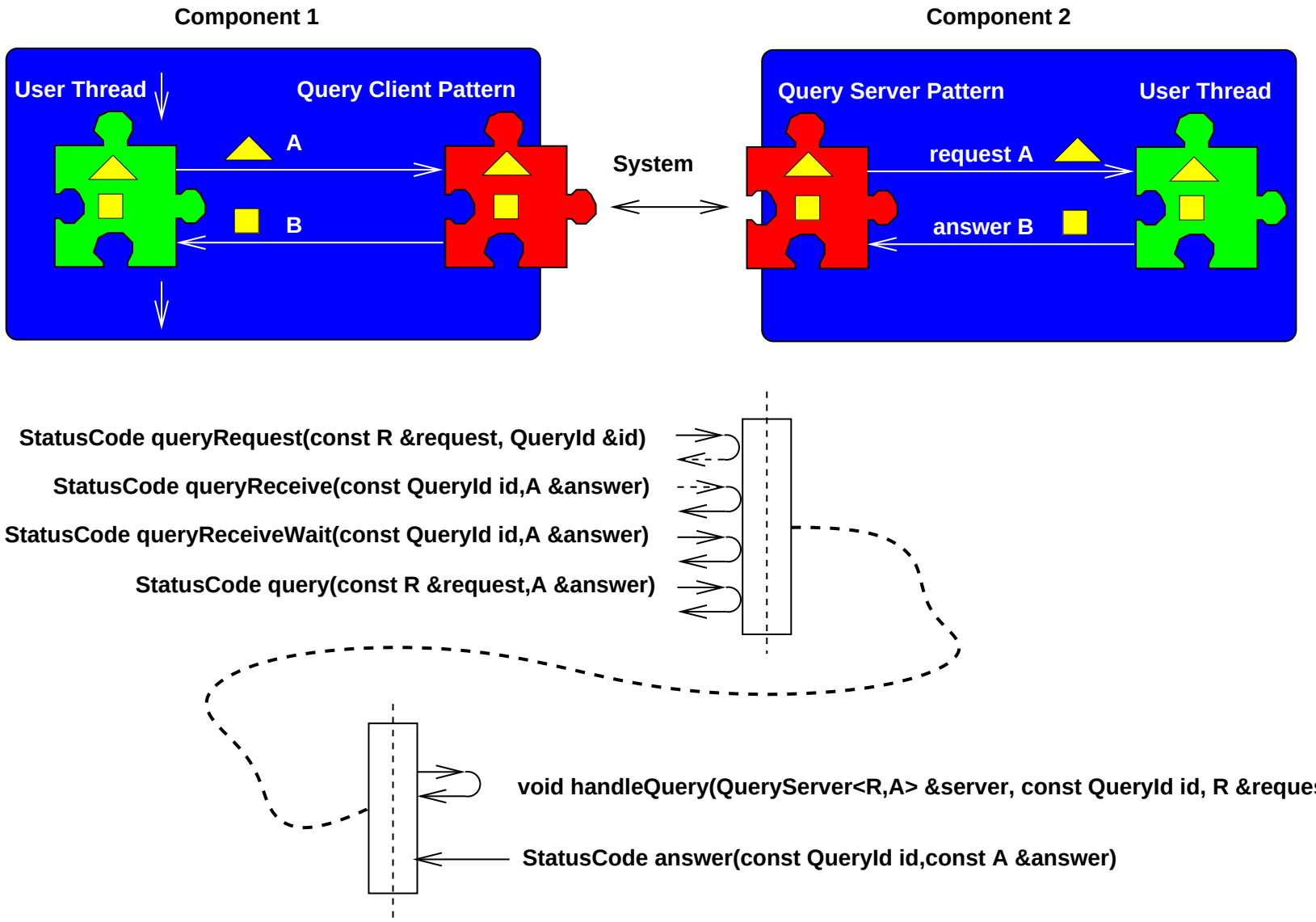
- Communication Patterns
- Processing Patterns
  - ⋆ Active Handler Objects
- Component Organization
  - ⋆ Component Wiring Pattern (included December 2002)

- Send
- Query
- Push
  - ⋆ Push Newest
  - ⋆ Push Timed
- Event
- State

(see design document or online documentation for detailed description)



Communication Patterns

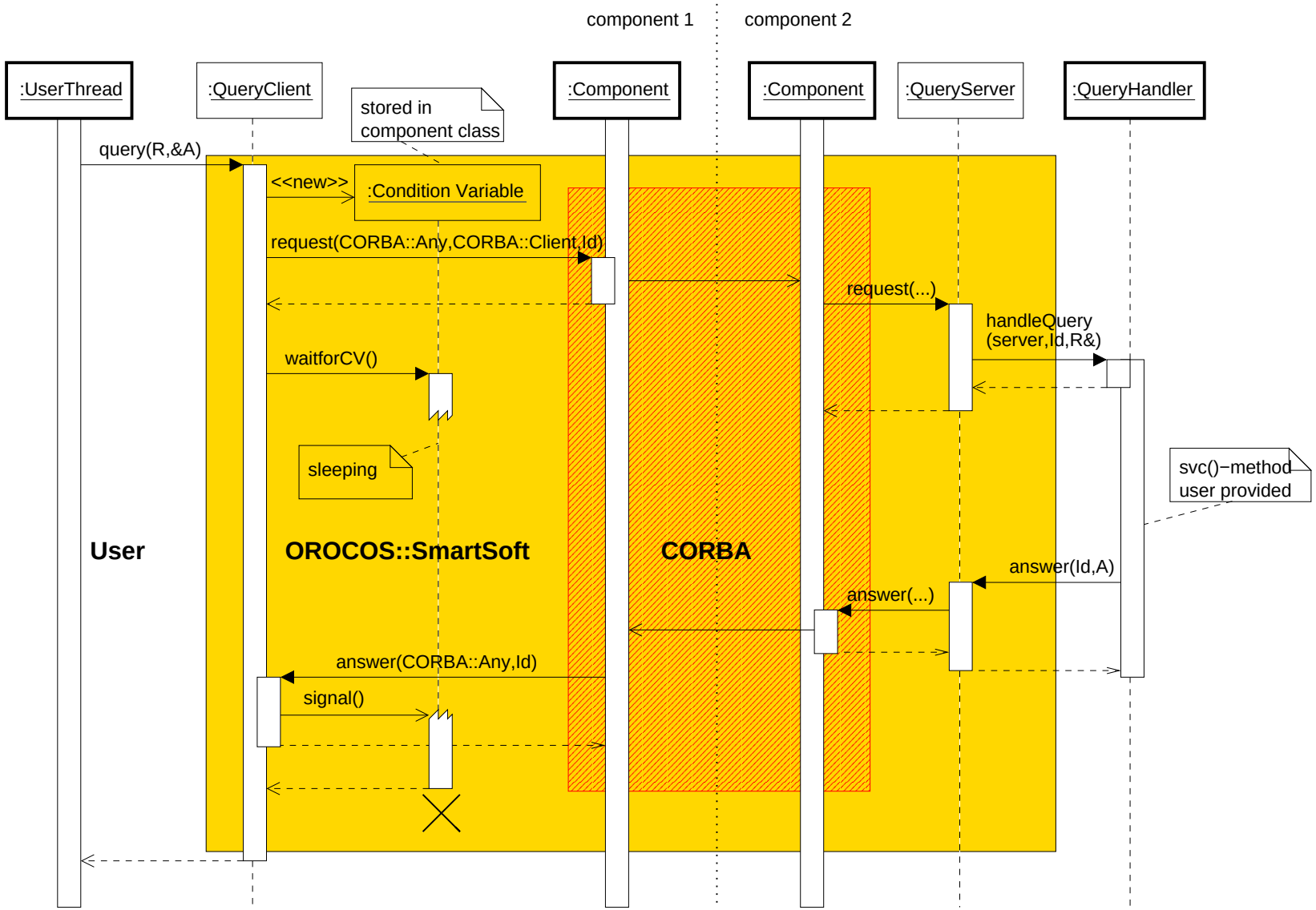


Communication Object

Communication Object Objects which parameterize the communication pattern templates. They contain both, the data structure to be transmitted and the access methods.

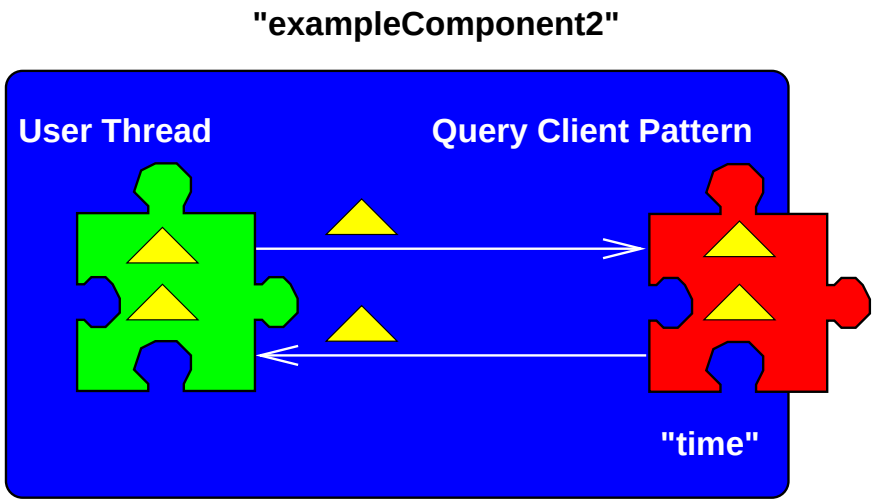
- ⋆ object transferred
- ⋆ object locally available
- ⋆ reduce network traffic
- ⋆ arguments in methods not restricted to CORBA types
  since methods are not part of IDL description
  (e.g. STL classes ...)
- ⋆ easily extensible (Decorator-Pattern) without CORBA knowledge
- ⋆ ...

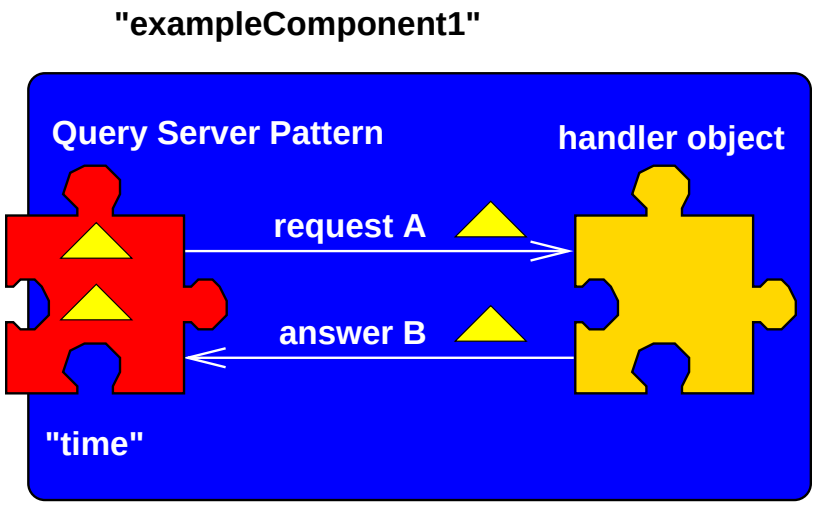
Query Pattern



Source Code Example

- unique component name
- unique service name within component

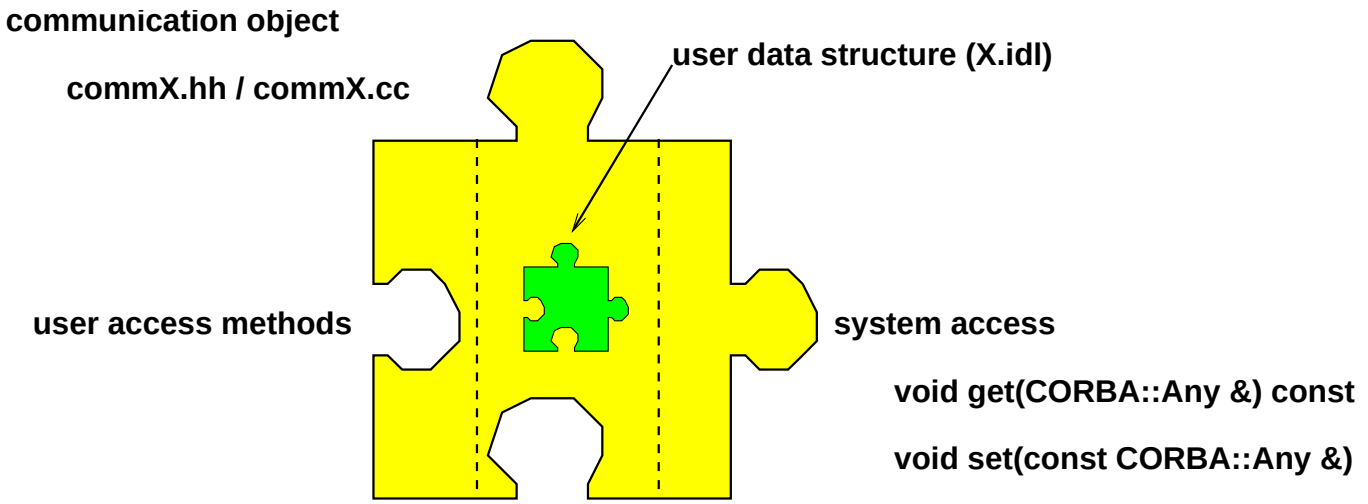




queryClient<...,...> (component, "time", "exampleComponent1")

Example Communication Object

- Communication Objects: Laserscan, Gridmap, Polygonal goal region, ...
- Example: Transmit time



To be provided by user:

- `timeOfDay.idl`
- `commTimeOfDay.hh`
- `commTimeOfDay.cc`

```
struct TimeOfDay {
  short hour;
  short minute;
  short second;
};
```

Example Communication Object

```
#ifndef COMM_TIME_OF_DAY_HH
#define COMM_TIME_OF_DAY_HH

#include <iostream>
#include "timeOfDayC.hh"

class CommTimeOfDay {
  protected:
    TimeOfDay time;
  public:
    CommTimeOfDay();
    virtual ~CommTimeOfDay();

    // SmartSoft interface
    void get(CORBA::Any &) const;
    void set(const CORBA::Any &);

    // user interface (no CORBA types here !)
    void get(int&,int&,int&);
    void set(int,int,int);
    void print(ostream &os = cout) const;
};
#endif
```

SmartSoft Interface Implementation:

```
void CommTimeOfDay::get(CORBA::Any &a) const
{
   a <<= time;
}

void CommTimeOfDay::set(const CORBA::Any &a)
{
   TimeOfDay *t;

   a >>= t;

   time = *t;
}
```

Example Client (Header)

```
#include "smartSoft.hh"

#include "smartExampleComponent1Client.hh"

CHS::SmartComponent *component;

CHS::QueryClient<CommTimeOfDay,CommTimeOfDay> *timeClient;

class UserThreadA : public CHS::SmartTask {
  public:
    UserThreadA() {};
    ~UserThreadA() {};
    int svc(void);
};
```

Example Client (Task)

```
int UserThreadA::svc(void) {
  CHS::QueryId    id1, id2;
  CHS::StatusCode status1, status2;
  CommTimeOfDay   q1,q2,a1,a2;

  time_t time_now;
  struct tm *time_p;

  while(1) {
    time_now = time(0);time_p = gmtime(&time_now);
    q1.set(time_p->tm_hour,time_p->tm_min,time_p->tm_sec);
    status1 = timeClient->queryRequest(q1,id1);

    time_now = time(0);time_p = gmtime(&time_now);
    q2.set(time_p->tm_hour,time_p->tm_min,time_p->tm_sec);
    status2 = timeClient->queryRequest(q2,id2);

    status1 = timeClient->queryReceiveWait(id1,a1);
    status2 = timeClient->queryReceiveWait(id2,a2);
  }
  return 0;
};
```

Example Client (Main)

```
int main (int argc, char *argv[]) {
  try {
    CHS::SmartThreadManager *threadManager = CHS::SmartThreadManager::instance();

    component  = new CHS::SmartComponent("exampleComponent2",argc,argv);
    timeClient = new CHS::QueryClient<CommTimeOfDay,CommTimeOfDay>
                                  (component,"exampleComponent1","time");

    UserThreadA user1;
    user1.open();

    component->run();
    threadManager->wait();
  } catch  (const CORBA::Exception &) {
    cerr << "Uncaught CORBA exception" << endl;
    return 1;
  } catch (...) {
    cerr << "Uncaught exception" << endl;
    return 1;
  }
  return 0;
}
```

Example Server (Header, Handler)

```
#include "smartSoft.hh"

#include "commTimeOfDay.hh"

CHS::SmartComponent *component;

class TimeQueryHandler : public CHS::QueryServerHandler<CommTimeOfDay,CommTimeOfDay> {
  public:
    void handleQuery(CHS::QueryServer<CommTimeOfDay, CommTimeOfDay> & server,
                     const CHS::QueryId id,
                     CommTimeOfDay& r)
    {
      CommTimeOfDay a;

      time_t time_now = time(0);
      struct tm *time_p = gmtime(&time_now);

      a.set(time_p->tm_hour,time_p->tm_min,time_p->tm_sec);

      server.answer(id,a);
    }
};
```

Example Server (Main)

```
int main (int argc, char *argv[])
{
  try {
    component = new CHS::SmartComponent("exampleComponent1",argc,argv);

    // Create time query and its handler
    TimeQueryHandler timeHandler;
    CHS::QueryServer<CommTimeOfDay,CommTimeOfDay> timeServant(component,"time",timeHandler);

    component->run();
  } catch (const CORBA::Exception &) {
    cerr << "Uncaught CORBA exception" << endl;
    return 1;
  } catch (...) {
    cerr << "Uncaught exception" << endl;
    return 1;
  }

  delete component;

  return 0;
}
```

Example

```
$ cd $SMART_ROOT
$ $TAO_ROOT/orbsvcs/Naming_Service/Naming_Service -m 0 -ORBEndpoint iiop://localhost:9999
$ bin/smartExampleComponent1 -ORBInitRef NameService=corbaloc:iiop:localhost:9999/NameService
$ bin/smartExampleComponent2 -ORBInitRef NameService=corbaloc:iiop:localhost:9999/NameService
```

Examples in Distribution

Example	Description
Example 1	query communication pattern, thread management, handling of slow and/or blocking queries
Example 2	push newest communication pattern, thread management
Example 3	query communication pattern, state management pattern, thread management
Example 4	query communication pattern, STL in communication objects, thread management
Example 5	event communication pattern, thread management
Example 6	send communication pattern, thread management, timer
Example 7	parameter management
Example 8	push timed communication pattern
Example 9	wiring pattern

Summary

- Summary
  - ⋆ Communication Patterns to structure Component Interactions
  - ⋆ CORBA based Implementation available
  - ⋆ ACE to hide OS specifics
  - ⋆ Server Components for RWI/B21, SICK/LMS,PLS etc. follow very soon
- How to get it ?
  - ⋆ http://www.orocos.org/ $\Longrightarrow$ OROCOS@FAW
  - ⋆ CVS server
  - ⋆ Detailed instructions
- Requirements
  - ⋆ TAO (ACE 5.2 / TAO 1.2 and newer)
  - ⋆ Tested on Linux
    (SuSE 7.0 and newer [gcc 2.95.3], Red Hat 7.1 [gcc 2.96.85]).
  - ⋆ Not yet compiled on SuSE 8.1 !

More Complex Communication Object

```
#ifndef _COMM_CALC_VALUES_HH
#define _COMM_CALC_VALUES_HH

#include <list>

#include "calcValuesC.hh"

class CommCalcValues {
  protected:
    list<int> values;

  public:
    CommCalcValues();
    virtual ~CommCalcValues();

    void get(CORBA::Any &) const;
    void set(const CORBA::Any &);

    void set(list<int>);
    void get(list<int>&);
};
#endif
```

```
typedef sequence<long> CalcValues;
```

More Complex Communication Object

```
void CommCalcValues::get(CORBA::Any &a) const {
  CalcValues v;
  int j=0;

  v.length(values.size());
  for (list<int>::const_iterator i=values.begin();i!=values.end();++i) {
    v[j++] = *i;
  }

  a <<= v;
}
```