Managed by

European Research Consortium for Informatics and Mathematics
ERCIM

CoreGRID: European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

EIA-FR contribution to WP3

CoreGRID Pierre Kuonen, EIA-FR

http://www.coregrid.net

Pierre.kuonen@eif.ch

GRID & Ubiquitous Computing Group

Information Society Technologies

ParoC++

An Object-oriented model for HPC on the GRID (p2p)

Programming model level

- Parallel object model
  - High level abstraction: to escape from send/receive paradigm

Programming tool level

- ParoC++ programming system (C++ extension)
- Developing and deploying Grid applications and components

The Parallel object model

Generalization of sequential objects (passive)

- Objects are distributed on the GRID but….
  - *As close as possible from the semantic of sequential model !*

The “good” proprieties of OO programming paradigm must be conserved

- Interaction between objects by method invocations
- Encapsulation
- Inheritance
- Polymorphism
- …

Parallel object

- Various method invocation semantics
- Transparent and dynamic object allocation guided by the object resources need.
- Shareable, “transmissible”
- No explicit send/receive

Parallelism support

- Inter-object parallelism
  - Asynchronous invocations
  - Dynamic parallel object creation/destruction
  - Passing parallel objects as arguments
  - Control: Synchronous/Mutex method invocations
- Intra-object parallelism
  - Concurrent method invocations
  - Synchronization : block mutex and event raise/wait

Methods invocations semantic

Caller side

- Synchronous invocation
  - Return when finished
- Asynchronous invocation
  - Return immediately

Object side

- Sequential
  - Partial serialization of invocations
- Mutex
  - Full serialization of invocations
- Concurrent
  - Concurrent execution

Example : Integer Class

File: integer.h

```
1: class Integer {
2: public:
3:    Integer(int wanted, int minp);
4:    Integer(char *machine);
5:    void Set(int val);
6:    int Get();
7:    void Add(Integer &other);
8: private:
9:    int data;
10: };
```

Example: Implementation

File: integer.cc

```
1 : #include "integer.h"
2 : Integer::Integer(int wanted, int minp)
3 : {}
4 : Integer::Integer(char* machine)
5 : {}
6 : void Integer::Set(int val) {data=val;}
7 :   {data=val;}
8 : int Integer::Get()
9 :   {return data;}
10: void Integer::Add(Integer &other)
11:   {data=other.Get();}
```

Example: The main program

File: main.cc

```
1 : #include "integer.ph"
2 : int main(int argc, char **argv) {
3 :  try { Integer o1(100,80), o2("localhost");
4 :   o1.Set(1); o2.Set(2);
5 :   o1.Add(o2);
6 :   cout<<"Value="<<o1.Get();
7 :  }
8 :  catch (paroc exception *e) {
9 :   cout<<"Object creation failure";
10:   return -1;
11:  }
12:  return 0;
13: }
```

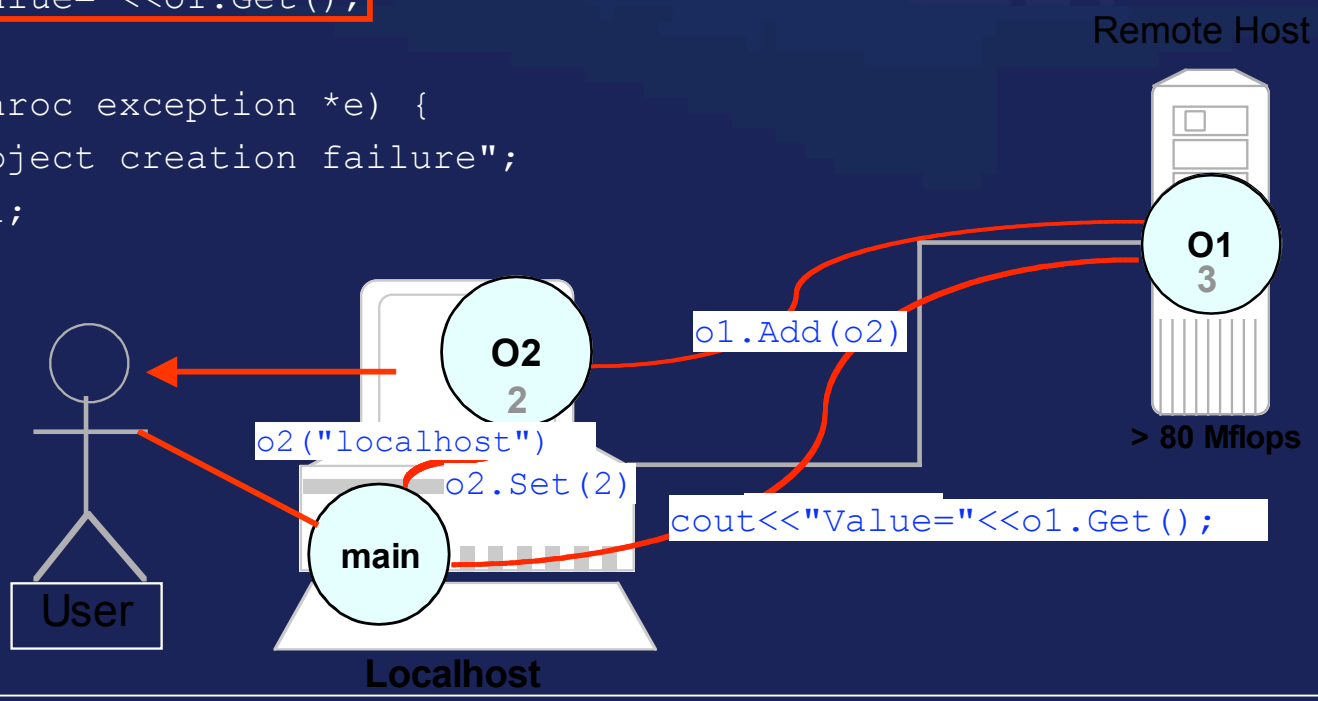
Syntax (ParoC++ = C++ extension)

File: integer.h

```
1: parclass Integer {
2: public:
3:    Integer(int wanted, int mini) @{power>=wanted?: mini;}
4:    Integer(char *machine) @{host=machine;};
5:     seq async void Set(int val);
6:     conc int Get();
7:     mutex void Add(Integer &other);
8: private:
9:    int data;
10: };
```

Execution

```
1 : #include "integer.ph"
2 : int main(int argc, char **argv) {
3 : try { Integer o1(100,80), o2("localhost");
4 :   o1.Set(1); o2.Set(2);
5 :   o1.Add(o2);
6 :   cout<<"Value="<<o1.Get();
7 :   }
8 :   catch (paroc exception *e) {
9 :   cout<<"Object creation failure";
10:   return -1;
11:   }
12: return 0;
13: }
```



Test case 2: Hydro@Alpine3D project a collaboration with SLF/Davos