**The POP programming model**

**ACOMP 2007**

16th of March 2007

Ho Chi Minh City, Vietnam

Prof. Pierre Kuonen, Dr. Tuan Anh Nguyen

Communication and Information Technologies Dpt.

GRID & Ubiquitous Computing Group

University of Applied Science Western Switzerland

Pierre.Kuonen@elf.ch

Tél.: ++41 (0)26 429 65 65

Fax: ++41 (0)26 429 66 00

**GRID**

**Introduction**

- GRID environments are by essence
  - Heterogeneous
  - Dynamic
  - Decentralised
  - ...

- These characteristics are very often presented as disadvantages or as difficulties in terms of programming activity

  **BUT.....**

- Current and future distributed HPC applications also have these characteristics
  - code coupling ....

- The main problem becomes from the fact that current HPC programming models are not adapted to such situations
  - MPI, SPMD approach ...

**Active vs inactive objects**

- Active objects
  - Objects have an intrinsic activity
    - Parallelism is realized thank to internal activities of the objects

- Inactive objects
  - Objects are active only when an method is invoked
    - Closer to the sequential semantic but...
      - Parallelism must be realized thank to methods invocations

- Need for asynchronous method invocation
Parallelism thank to asynchronism

- Two ways to call a method
  - Synchronous
    - Method returns when the execution is finished
    - Same semantic than sequential invocation
  - Asynchronous
    - Method returns immediately
    - Allows parallelism but.. no returned value

Order and atomicity control

- The POP model defines three ways to execute a method
  - Concurrent
    - No order no atomicity
  - Sequential
    - Ordered and atomic regarding others sequential methods invocations
  - Mutex
    - Totally ordered and atomic

Combination of synchronous/asynchronous and concurrent/sequential/mutex allows for six different semantics of methods invocations

POP: Methods invocations, recipient side

- We transformed a «parallelism» problem in a «concurrency» problem
  - Order and atomicity

Semantic : example
Method call semantics: definition

1. An arriving **concurrent** call can be executed concurrently (time sharing) when it arrives, except if mutex calls are pending or executing. In the later case he is executed after completion of all mutex calls previously arrived.

2. An arriving **sequential** call is executed after completion of all sequential and mutex calls previously arrived.

3. An arriving **mutex** call is executed after completion of all calls previously arrived.

Separation of caller and called side

- This approach clearly separates the semantic of the call for:
  - The object who calls the method
    - synchronous / asynchronous
  - The object who executes the method
    - concurrent / sequential / mutex

- As a consequence **six** semantics for methods calls are defined
  - synchronous concurrent
  - synchronous sequential
  - synchronous mutex
  - asynchronous sequential
  - asynchronous sequential
  - asynchronous mutex

POP-C++

- POP-C++ is an implementation of the parallel object model as an extension of C++
  - Syntax: six new key words have been added to the language
    - **parclass**: to declare parallel objects
    - **async**: asynchronous method call
    - **sync**: synchronous method call
    - **conc**: concurrent method execution
    - **seq**: sequential method execution
    - **mutex**: mutex method execution

Example: Integer Class

**File: integer.h**

```cpp
1: class Integer {
2:   public:
3:   Integer();
4:   void Set(int val);
5:   int Get();
6:   void Add(Integer &other);
7:   private:
8:   int data;
9:   }
```

Printed with FinePrint - purchase at www.fineprint.com
### Integer: Implementation

**File: integer.cc**

```cpp
#include "integer.h"

Integer::Integer(){}  
int Integer::Get(){return data;}  
void Integer::Add(Integer &other){data+=other.Get();}
```

### Integer: The main program

**File: main.cc**

```cpp
#include "integer.h"

int main(){  
    Integer o1, o2;  
    o1.Set(1); o2.Set(2);  
    o1.Add(o2);  
    printf("Value=%d\n",o1.Get());  
}
```

### Syntax (POP-C++ = C++ extension)

**File: integer.ph**

```cpp
parclass Integer
public:
    Integer();
    seq async void Set(int val);
    conc int Get();
    mutex void Add(Integer &other);
private:
    int data;
};
```

### Execution

```cpp
#include "integer.ph"

int main(){
    Integer o1, o2;
    o1.Set(1); o2.Set(2);
    o1.Add(o2);
    printf("Value=%d\n",o1.Get());
}
```
Question!

- Can the programmer influence:
  - The chosen machine to run a parallel object
  - The protocol to use to connect parallel objects
  - The data encoding to use to transmit data to objects
  - ….

- The answer is YES…
  - Thanks to « object description »

Object description

- Parallel objects can express the needed resources to run thanks to a so-called object description (OD).
- OD is a part of the POP-C++ language.
- Each constructor of a parallel class can be associated with an OD which contains a high level expression that can be parameterized with actual inputs of the constructor.
- These requirements will be evaluated at run-time to generate the resource requirement specification which in turn will be used by the runtime system to perform the resource discovery for that specific parallel object.

OD syntax

- OD expressions are declared for each constructor of a parallel class directly after the constructor declaration and before the end-of-instruction symbol “;”.
- The syntax of OD is the following:
  - @{resources requirement expression}
- The current implementation allows indicating resources requirement in terms of:
  - Computing power (Mflops), keyword power
  - Memory size (MB), keyword memory
  - Bandwidth (Mb/s), keyword network
  - Location (host name or IP address), keyword url
  - Protocol (socket or http), keyword protocol
  - Data encoding (raw, xdr, raw-zlib, xdr-zlib), keyword encoding

Formal syntax

- object description :=
  @{resources requirement expression}
- resources requirement expression :=
  od.<resource type1> (<numexpr>) | od.<resource type1> (<numexpr>, <numexpr>) | od.<resource type2> (<strexpr>)
- resource type1 := power | memory | network | walltime
- resource type2 := protocol | encoding | url
- numexpr := a real number expression
- strexpr := a null-terminated string expression
Example:

```
parclass MyObj {
  public:
    MyObj(float P) {
      od.power(P); od.memory(100,60);
      od.protocol("socket http");
    }
    ...
};
```

**Interpretation:**
- To run an instance of `MyObj` we need:
  - At least `P` Mflops (available power)
  - Preferably 100 MB of ram but at least 60MB
- The protocol to use to communicate with this object is `socket` or `http`

---

**Example: Integer Class**

```
class Integer {
  public:
    Integer(int wanted, int minp);
    Integer(char *machine);
    void Set(int val);
    int Get();
    void Add(Integer &other);
  private:
    int data;
};
```

**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. }

---

**Integer: Implementation**

```
#include "integer.h"

Integer::Integer(int wanted, int minp) 
{} 

Integer::Integer(char* machine) 
{}

void Integer::Set(int val)
{
  data=val;
}

int Integer::Get()
{
  return data;
}

void Integer::Add(Integer &other)
{
  data=data+other.Get();
}
```

---

**Integer: The main program**

```
#include "integer.h"

int main()
{
  Integer o1(100,80), o2("localhost");
  o1.Set(1); o2.Set(2);
  o1.Add(o2);
  cout<<"Value="<<o1.Get();
  return 0;
}
```
Syntax (POP-C++ = C++ extension)

- File: integer.ph

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

POP-C++ implements the parallel object model

- A (minimal) extension of C++

```cpp
class Foo { ...
    Foo(...);
    void MyClass(...);
};
```

```cpp
parclass Foo { ...
    Foo(...) @{ power=100; }
    async conc void MyClass(...);
};
```

Shared implementation

Exception handling

- Exceptions are propagated through the network
  - Exception thrown in a remote object can be caught in the local object

```cpp
try
    Object::Funct(...)
    ...
    O.Funct(...) ...
    catch(int x)
    {
        handle x here ...
    }
    throw x;
```
Coupling POP-C++ with MPI code

- **Why?**
  - There exists a lot of codes already written in MPI (library,...)
  - In some situations MPI could be a better choice than POP-C++
- **Provided in the standard POP-C++ library:**

  ```cpp
  template<class T> class POPMPI
  {
  public:
    ...;
    POPMPI(int np); // Create np MPI process of type T
    ...;
    bool ExecuteMPI(); // Execute this method on all processes
  };
  ```

Coupling POP-C++ with MPI code

- **Put your MPI code in a POP-C++ object (**parclass**)
  - **Example:** the **TestMPI** parclass

  ```cpp
  parclass TestMPI
  {
  public:
    TestMPI();
    async void ExecuteMPI(); // Mandatory: Here, your MPI code
    async void Set(int v); // Optional, others methods
    sync void Get(); // ...
    ...;
  }
  private:
    int val;
  };
  ```

How to use:

```cpp
#include <popc_mpi.h>

int main(int argc, char **argv)
{
  ...
  POPMPI<TestMPI> mpi(2); // Create 2 MPI processes
  mpi.ExecuteMPI(); // Call ExecuteMPI method on all processes
  ... // MPI processes
  mpi[0].Set(mpi[1].Get()); // Individual access to MPI process
  ...}
```
Interoperable Architecture

Compiler and runtime routines

- interoperable distributed POP-C++ application
- POP-C++ language library
- POP services
- adapter
- GRID middleware / services
- operating system / virtual machine

POP-C++ is an on-going activity

- Available at: www.eif.ch/gridgroup/popc
  - Porting on Windows
  - POP-java
  - Tracing debugging, editing,…
  - Compiling, run-time, pop-services (middleware),…
  - …

- Contacts:
  - Prof. Pierre Kuonen: pierre.kuonen@eif.ch, EIA-FR, Switzerland
  - Dr. Tuan Anh Nguyen: tuananh.nguyen@hcmut.edu.vn, HCMUT, Vietnam

Thank you for your attention!!