
Linguistic Support for Large Scale Distributed Programming

Patrick Th. Eugster *Rachid Guerraoui*











Swiss Federal Institute of Technology

Lausanne



Lessons Learned

Publish/subscribe paradigm

-  Decoupling of information producers/consumers in
 -  Time
 -  Space
 -  Flow
-  Enforces scalability by removing direct dependencies
-  Static vs. dynamic schemes
 -  Both advantages and shortcomings
 -  Combination most effective
-  Not a solution to everything
 -  But more effective in many cases than RMI

✍ **Can be implemented in OO such that**

- ✍ Type safety is ensured
- ✍ Encapsulation is preserved
- ✍ Efficient

?

✍ **Generic Distributed Asynchronous Collections**

- ✍ Type-based publish/subscribe enforces type safety
- ✍ « Reflection-based » publish/subscribe enforces encapsulation, optimizations without subscription language
- ✍ Combination enforces ease of use, efficiency

Language Integration

Paradigms/abstractions

1. Libraries
2. Libraries become part of language environment
3. Integrated into language semantics

E.g., monitor

1. As external concurrency control library
2. Added as control structure to Portal
3. Every object is monitor in Java

✍ Requirements to ensure

- ✍ Type safety avoiding type checks and casts
- ✍ Accessibility of subscription patterns and encapsulation of message objects in content-based subscription without subscription language
- ✍ Expressing different QoS


?

P/SLang

Publishing?

 Language primitive, e.g., new

```
publish p;
```

 Creates instance (copy) of `p` in every subscribed process

Subscribing?

 Language primitive, e.g., instance of

```
s subscribe T;
```

 Subscribes `s` to type `T`

✍ Callback?

- ✍ Every object `s` can override a method, e.g, `equals()`
`notify(Object o)`

✍ Strong typing?

- ✍ Every object can implement several `notify()` clauses
- ✍ With different argument types: dynamic overriding

✍ Dispatching types?

- ✍ Every incoming object is an `Object`
- ✍ Requires *dynamic dispatching*

✍ Expressing content-based subscription?

✍ Such that

- ✍ Encapsulation is preserved
- ✍ Pattern is transparent
- ✍ No subscription language is required

✍ Use language to express query

✍ Defer code evaluation

- ✍ *Multistage programming* (two levels are sufficient)
s subscribe T [T t | return t.equals(t1)] ;
- ✍ Anonymous classes (methods)?

✍ Anonymous methods (functions)

✍ Similar to Smalltalk blocks

✍ Deferred evaluation

✍ No name

✍ Not associated with a type

✍ Parameters

✍ Exceptions

✍ Poss. return value

✍ **Of type `java.lang.reflect.Method`, e.g.**

Method m =

```
void (String s, int i) throws Exception {...};
```

✍ **General**

```
s subscribe T boolean (T t) {...return ...; };
```

✍ **Or abbreviation**


```
s subscribe (T t) {return ...; };
```

✍ **Similar constraints as anonymous classes**

✍ Only `final` or block-local variables are used: shipping of code possible

✍ Otherwise no shipping of code (local filtering)

QoS?

 Messages use multiple subtyping to inherit behavior of predefined message classes, e.g,


 Unreliable/reliable/certified...

 Priorities

 Persistence

 ...



Removes ambiguities possible with DACs

 If different objects connect to the same type, but with different QoS, e.g., «unreliable» publisher - «reliable» subscriber?




RMI-Based Programming

Two types of objects

1. Remotely accessible objects

-  Have remote interface, can be remotely invoked
-  Passed by reference

2. Local objects

-  No remote interface
-  Can be used as invocation arguments/return values
-  Passed by value

One interaction style

-  Objects interact locally and remotely through invocations

Message-Oriented Distr. Progr.

One type of objects

 Pass by value

 No remote interfaces

Two interaction styles

 Locally through method invocations

 Remotely/locally by exchanging (message) objects

 Objects are published

 Subscribing to types of objects (with predicates)