On Objects and Events

P. Th. Eugster [1], R. Guerraoui [1], C. H. Damm[2]

[1] Swiss Federal Institute of Technology, Lausanne

[2] University of Aarhus, Denmark

Roadmap

- **Introduction**
  - Context
  - Background
- **Type-Based Publish/Subscribe**
- **Java$_{PS}$**
  - Syntax
  - Implementation
- **Conclusions**
- **Future Work**

Context

- **Large-scale distributed object-based computing**
  - e-business, banking, finance, telecommunications, ...
- **Ubiquitous, pervasive, peer-to-peer, ... computing**
  - Decentralized
  - Decoupling of participants
- **Need for**
  - Algorithms
  - *Abstractions*

Abstractions for Distr. Programming

- **Have been „integrated“ in many languages**
- **Remote procedure call (RPC) et al.**
  - Apply nicely to object settings
  - Argus, CLU, Modula 3, Obliq, Java RMI, ...
  - Type safety and encapsulation (also of distribution details: transparency)
  - Application-defined types
- **Distributed shared memory (DSM) et al.**
  - Tuple space in Linda
  - „Object“ spaces in Objective Linda, Smalltalk, C++, Java, ...
  - Message queues

Publish/Subscribe

- **Close to shared space**
  - Global „event bus“
  - Publish events
  - Subscribe to events
- **Decoupling of publishers and subscribers**
  - In time: do not have to be up at the same time
  - In space: do not have to know each other
  - In flow: asynchronous sending and receiving of events
  - Removes dependencies, thereby enforcing scalability

Topic-Based and Content-Based

- **cf. groups**
- **Explicit addressing scheme**
  - Given by topics
  - Motivated by interoperability
- **Hierarchical disposition of topics**
  - Wildcards
  - Aliases
- **Predefined event types**
  - „Self-describing“ events
- **Single event space**
- **Implicit addressing scheme**
  - Given by properties of events
  - Comes closer to tuple spaces
- **Events are viewed as attribute sets**
- **Subscriptions expressed**
  - Necessarily on these attributes
  - Query languages, e.g., SQL
  - Templates

Type-Based Publish/Subscribe

- **High-level variant of publish/subscribe**
  - cf. RPC
- **Events are objects (*obvents*)**
  - Instances of application-defined types
- **Publishing obvents**
  - Similar to a distributed `new`
  - Similar to a distributed `clone()`
- **Subscribing to obvent types**
  - Including „content“-based queries expressed on public members
- **Emphasis: type safety and encapsulation**
  - „Open“ subscription patterns, QoS

In Java

- **Obvents**
  - Easily transferrable
  - Defined by the application as specific classes (and interfaces)
- **Java inherently provides *serialization***
  - Default behavior by subtyping `java.io.Serializable`
  - `Obvent extends Serializable`
- **Generic Distributed Asynchronous Collections**
  - Library approach
  - Genericity for type safety
  - Structural reflection for encapsulation-preserving subscriptions

Java$_{PS}$ Syntax

- **Two primitives added**
- **Publish statements**
  - Publishing obvents
  - E.g., an obvent `t` of an arbitrary type `T`

```
publish t;
```

- **Subscription expressions**
  - Subscribing to obvent type
    - Including filters
  - E.g., an obvent type `T`

```
subscribe(T t) { /* filter */ } { /* handler */ }
```

  - Returns a subscription handle

Subscriptions

- **Obvent handlers**
  - *Closures*
    - Describing the handling of obvents
  - Motivation: type safety, regrouping of all code related to a subscription
- **Filters**
  - Specific closures
    - Describing the filtering of obvents
    - *Deferred evaluation*
    - Code is potentially transferred to enable optimizations
  - Motivation: as above, however by revealing filter semantics at compilation

Example: Stock Trade

```
public class StockQuote ... {
 private String company;
 private float value;
 private int amount;
 public String getCompany()
   { return company; }
 public float getValue() {...}
 public int getAmount() {...}
 public StockQuote(String c,
                   float v,
                   int a) {
   company = c;
   value = v;
   amount = a;
 }
}
```

Publishing stock quotes

```
StockQuote q = new
 StockQuote("Telco", 100.0, 25);
publish q;
```

Subscribing to stock quotes

```
Subscription s =
 subscribe(StockQuote q)
{
 return
  q.getCompany().equals("Telco");
}
{
  System.out.println(q.getPrice());
}
s.activate();
```

Qualities of Service

- **Increased importance in a distributed setting**
  - Local context: usually exactly-once of operations
  - Asynchronous distributed systems
  - E.g., unreliable, reliable, certified, ...
- **Associated with events**
  - Like a context: ensures a „correct“ handling along path
  - Part of events through *subtyping* : `Stockquote extends ...`
    - E.g., `ReliableObvent, CertifiedObvent`
    - Also more refined properties, e.g., `PriorityObvent`
    - Or more specific algorithms

Implementation

- **„Heterogenous“ translation at compilation**
  - Type-specific adapters are created for every obvent type
    - Adapters are similar in nature to proxies for RPC
    - `TAdapter` for every type `T`
    - Driven by class-based dissemination of events
  - Also „homogenous“ translation for „single“ event bus
    - `ObventAdapter`
- **Method added to every obvent class for publishing**
  - `publish()`method
  - Automatically sent to the right adapter, e.g.,
  - `publish t ↴ t.publish()↴ TAdapter.publish(t)`

Implementation (cont´d)

- **Subscription to a type `T` is transformed**
  - `subscribe()` method call to `TAdapter`, e.g.,
  - `subscribe(T t){…}{…}` ↴ `TAdapter.subscribe(…, …)`
- **Handlers**
  - Mapped to Java anonymous classes
- **Filters**
  - If not „easily transferrable“, then mapped to anonymous classes
  - Otherwise, intermediate representation is generated
    - Method invocation tree
    - Predicate tree

Conclusions

- **Object-oriented publish/subscribe does make sense**
  - Experiences in telecommunications and banking
  - Type-based publish/subscribe represents alternative to RMI, not replacement
- **Type-based publish/subscribe can implement**
  - Content-based publish/subscribe,
  - Topic-based publish/subscribe, or
  - any mixture of these
- **Type information**
  - Enables type checks at compilation
  - Enables performance optimizations at runtime

Future Work

- **„Language“ issues**
  - Filter semantics
  - Languages (mechanisms) for „clean“ library implementations of type-based p/s (and other distributed programming abstractions)
  - Interoperability
    - Language-independent *event definition language* (EDL), e.g., XML, (subset of) IDL, to define events as objects, i.e., with methods
    - cf. CORBA value types
    - Structural conformance
- **„Implementation“ issues**
  - Highly scalable multicast algorithms
  - Multi-level filtering

Related Work

- ***Events + Constraints + Objects* (ECO)**
  - Filters
    - Based on event properties, or
    - Predicates based on (local) constraints
  - Events
    - First class, specific constructs, yet
    - Viewed as sets of attributes
- ***Cambridge Event Architecture* (CEA)**
  - Interoperability
    - Java, C++
    - ODL, XML also mentioned as EDL
  - Events are viewed as sets of attributes