# Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction

*Patrick Eugster* *Rachid Guerraoui* *Joe Sventek*

Swiss Federal Institute of Technology Lausanne

Agilent Laboratories Scotland Edinburgh

{Patrick.Eugster, Rachid.Guerraoui}@epfl.ch sventek@labs.agilent.com

# Roadmap

- **Distributed Asynchronous Collections (DACs)**
  - Reminder: Collections
  - Distributed Collections
  - Distributed Asynchronous Collections
- **The DAC Framework**
  - Interfaces
  - Classes
  - Characteristics
  - Implementation
- **Programming Example**
- **Future Work & Conclusions**

# Reminder: Collections

- **Collection**
  - A container abstraction used to store, retrieve and manipulate objects
  - Represents group of related objects, e.g., set, list, queue
- **Commonalities**
  - **Add** new elements
  - Check if the collection **contains** specific elements
  - **Remove** elements
  - …
- **Differences**
  - Element management
  - Size
  - …

# Distributed Collections

- **Accessible from various nodes**
- **Pull**
- **Similar to shared memory**
  - Participants can share information
- **Centralized**
  - Accessed through remote invocations
  - Single point of failure
- **Or not centralized: DACs**
  - Increased availability

# Distributed Asynchronous Collection

- **Callback to application: push**
- **Notification mechanism**
  - New element
  - Element has been removed
  - …
- **Requires subscription**
  - Observer design pattern: DAC is subject, client is observer
- **Several subscribers and publishers**
  - Publish/subscribe interaction scheme
  - DACs like event channels, topics, message queues, etc.

# DAC Framework

- **Collection frameworks**
  - Unify different semantics
  - Integrated with certain languages
    - Smalltalk
    - Java
  - Additional libraries
    - E.g. STL for C++
- **Java DACs**
  - Extension of `java.util` collections

```
public interface DACollection
        extends java.util.Collection {…}
```

# DAC Interfaces

- **Callback interface**

```
public interface Notifiable {
  public void notify(Object m, String DACName);
}
```

- **Subscribe (all-of-n)**
  - Without subtopics: `contains(Notifiable n);`
  - With subtopics: `containsAll(Notifiable n);`
- **Subscribe (one-of-n)**
  - Without subtopics: `remove(Notifiable n);`
  - With subtopics: `removeAll(Notifiable n);`

# DAC Classes

- **Different DAC types**
  - Different interaction styles
  - Push vs. pull, one-for-each vs. one-for-all
- **Different DAC classes**
  - For semantics not visible in interfaces
  - Duplicate elements
  - Reliability
  - …
- **Own classes for specific requirements**

# Characteristics of DACs

**Collection**

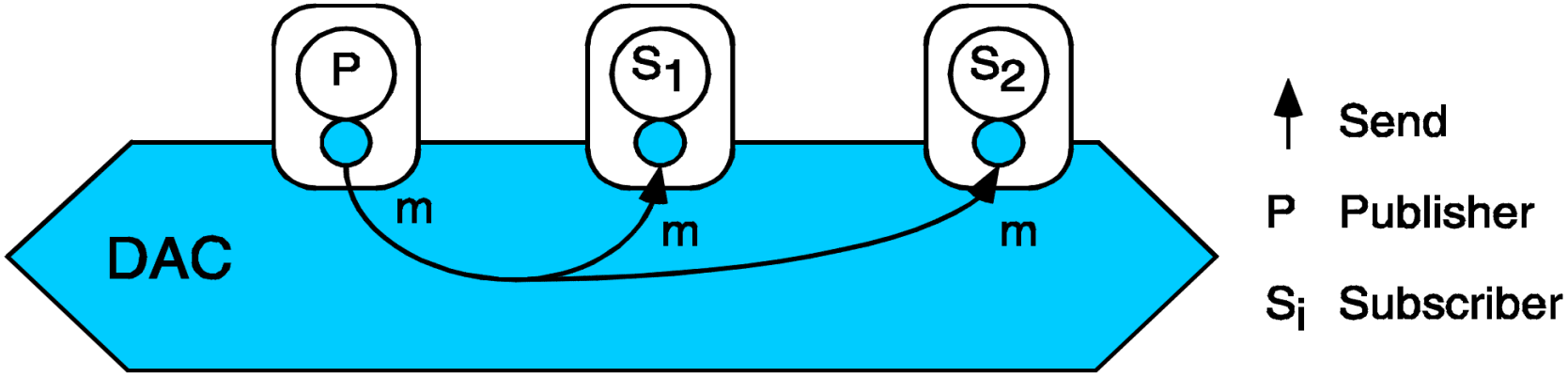
- **Storage order**
  - Deterministic
  - None
- **Duplicates**
- **Insertion order**
  - Explicit
  - Implicit
- **Extraction order**

**DACollection**

- **Delivery order**
- **Delivery semantics**
  - Unreliable
  - Reliable
  - Certified
- **Duplicates**
  - Elements
  - Delivery
- **Extraction order: pull**

# DAC Implementation

- **Lightweight proxies**
  - Appear as local collections



# Programming Example

- **Create a local DAC proxy**

```
DASet myChat = new DAStrongSet("/Chat/Insomnia");
```

- **Insert new objects (publish)**

```
myChat.add(new String("Hi from Bob"));
```

- **Register interest in new objects (subscribe)**

```
public class ChatNotifiable implements Notifiable {
  public void notify(Object m, String DACName) {
    System.out.println((String)m); }
}
myChat.contains(new ChatNotifiable());
```

# Future Work and Conclusions

- **Content-based publish/subscribe with DACs**
  - Static and dynamic classification schemes
  - Reflection for
    - Encapsulation
    - Avoiding subscription grammar
- **Type-based publish/subscribe**
  - Use type scheme as natural classification scheme of messages
  - Integration of language with middleware
  - Parametric polymorphism for DACs: generic DACs
- **DAC express several messaging styles and QoS**
  - One basic abstraction, different flavors
  - Framework can easily be extended