Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction

Patrick Eugster
Swiss Federal Institute of Technology
Lausanne

Rachid Guerraoui

Joe Sventek
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

춘 Collection
춘 A container abstraction used to store, retrieve and manipulate objects
춘 Represents group of related objects, e.g., set, list, queue

춘 Commonalities
춘 Add new elements
춘 Check if the collection contains specific elements
춘 Remove elements
춘 …

춘 Differences
춘 Element management
춘 Size
춘 …
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

 الغذائي Callback to application: push
غذي Notification mechanism
 غذي New element
 غذي Element has been removed
 غذي …
غذي Requires subscription
 غذي Observer design pattern: DAC is subject, client is observer
غذي Several subscribers and publishers
 غذي Publish/subscribe interaction scheme
 غذي 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

```java
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

<table>
<thead>
<tr>
<th>Collection</th>
<th>DACollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Storage order</td>
<td>✅ Delivery order</td>
</tr>
<tr>
<td>✅ Deterministic</td>
<td>✅ Delivery semantics</td>
</tr>
<tr>
<td>✅ None</td>
<td>✅ Unreliable</td>
</tr>
<tr>
<td>✅ Duplicates</td>
<td>✅ Reliable</td>
</tr>
<tr>
<td>✅ Insertion order</td>
<td>✅ Certified</td>
</tr>
<tr>
<td>✅ Explicit</td>
<td></td>
</tr>
<tr>
<td>✅ Implicit</td>
<td></td>
</tr>
<tr>
<td>✅ Extraction order</td>
<td>✅ Extraction order: pull</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lightweight proxies
Appears as local collections
Programming Example

Create a local DAC proxy

```java
daset myChat = new DASTrongSet("/Chat/Insomnia");
```

Insert new objects (publish)

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

Register interest in new objects (subscribe)

```java
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