APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

ANSA Phase III

The Design of a Persistence Service for Java

Scarlet Schwiderski

Abstract

This presentation introduces a persistence service for Java.

This persistence service is based on Reflective Java and exploits existing Java APIs, namely Object Serialization and Cryptography.

In order to provide the full functionality of a Persistent Store used concurrently by multiple applications, the issues of concurrency control, security, redundancy, and naming and locating of persistent objects need to be addressed.

The purpose of this project is twofold: to demonstrate and evaluate the benefits of Reflective Java, and to provide a persistence service for the Java programming language.

APM.1876.01	**Approved**	12th November 1996
	External Paper	

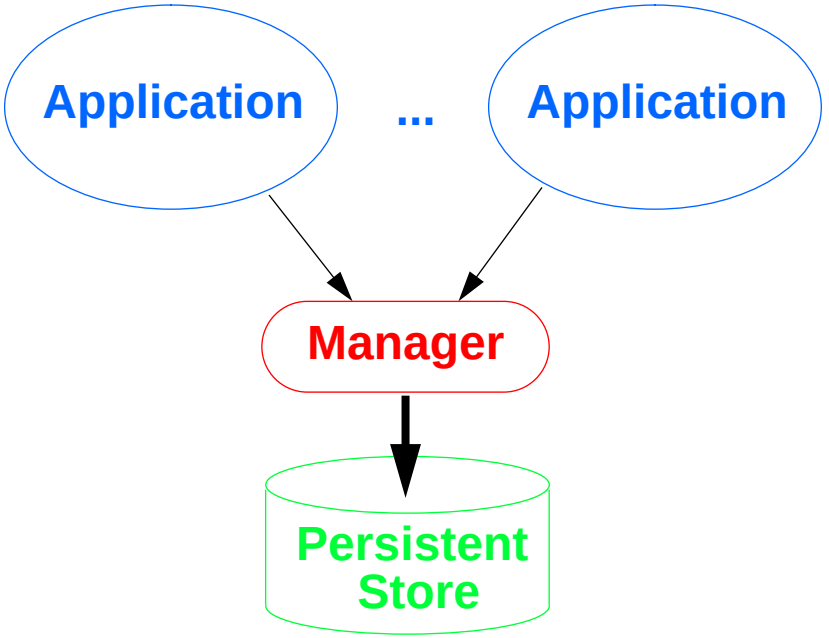
Distribution:
Supersedes:
Superseded by:

Design of a Persistence Service for Java

Scarlet Schwiderski
Zhixue Wu

Key Issue

- maintain a Persistent Store for Java objects in order to
  - make Java objects reusable by the same or by different applications
  - provide failure resilience to applications



Objectives

- take checkpoints of Java objects during and at the end of program execution
- retrieve Java objects from the Persistent Store as and when they are needed
- provide security measures in order to avoid forgery
- enable multiple applications to access the Persistent Store concurrently
- ensure the consistency of the stored Java objects
- make no changes to the Java language

Starting Point

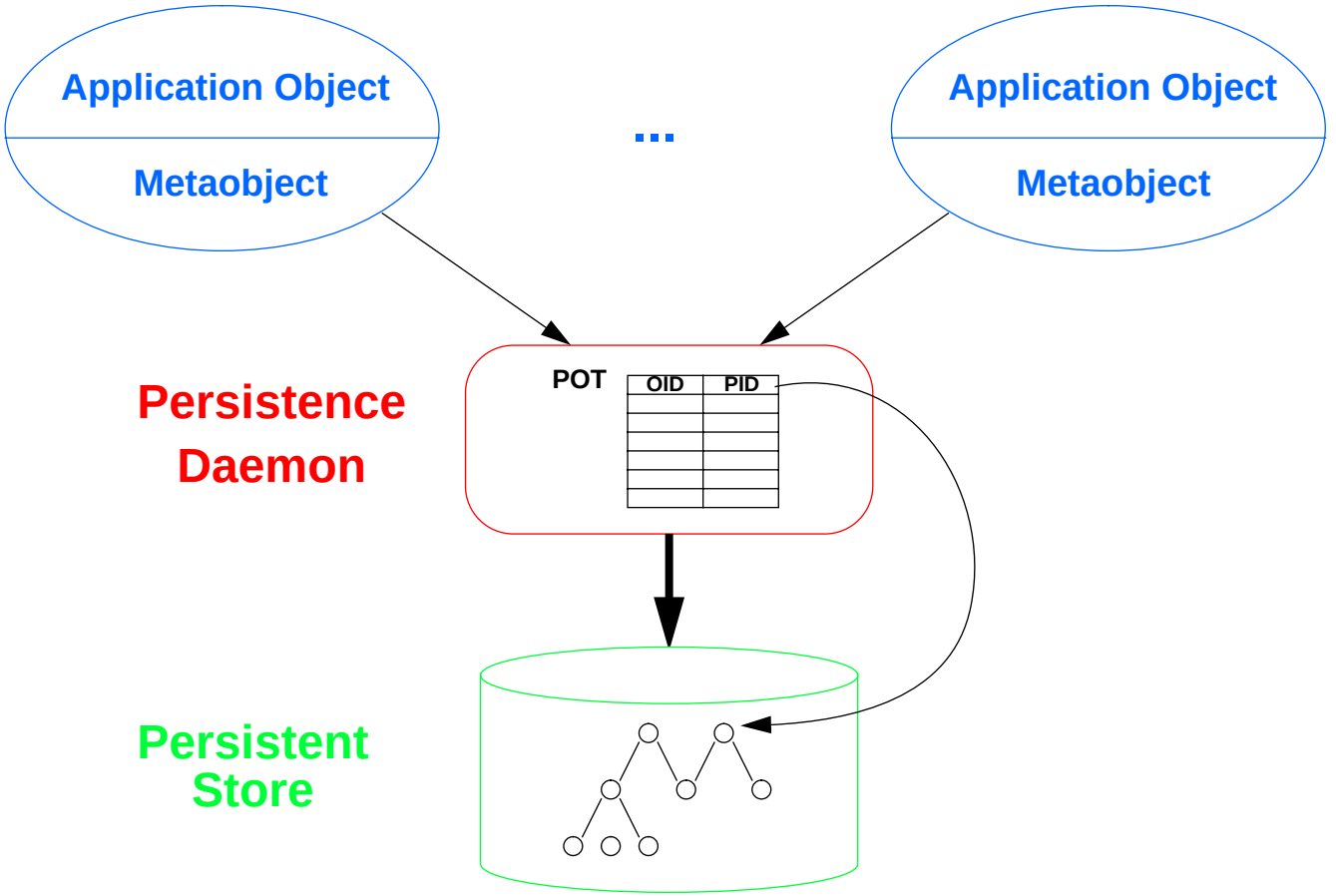
- **Exploit existing Java technology, i.e.**
  - **Sun's Object Serialization API**
  - **Systemics' Cryptography API**

- **Use reflection technology**
  - **to evaluate Reflective Java**
  - **to provide flexibility (for example, using a persistence metaobject in conjunction with a transaction metaobject)**

Java Object Serialization API

- Current version JDK 1.0.2
  - An object and all objects that are reachable from that object are stored in a flat file
  - Provides type checking
  - Does not provide concurrency control
  - Problem with redundancy
  - Problem with naming and locating serialized objects
  - Problem with security
  - Basic behaviour of object serialization can be enhanced

Persistence Model



Metaobject

- contacts the local Persistence Daemon in order to
  - store a corresponding Java object and all objects that are reachable from that object (passing an object reference)
  - load a corresponding Java object
    (passing an object identifier and an application identifier)
- may support other services, such as a transaction service

Persistence Daemon

- well-know central server at a site
- coordinates access to the Persistent Store
  - allocates disc space for objects in the Persistent Store
  - locates objects in the Persistent Store
  - manages multiple application objects at the same time
- maintains a Persistent Object Table (POT) in order to
  - keep track of activated (currently used) and deactivated (currently unused) Java objects
- provides garbage collection

Concurrency Control

- **Step 1: (simple version)**
  - **only one metaobject can use (that is, activate) a persistent object at any time**
  - **concurrent access to the POT is restricted**
- **Step 2: (advanced version)**
  - **concurrent access to the POT and the persistent objects is allowed by employing transaction metaobjects**

Loading Java Objects

- Metaobject contacts Persistence Daemon about loading a corresponding application object
- If the application object is deactivated
  - locate application object in Persistent Store
  - return it to metaobject
  - set entry in POT to *activated*
- If the application object is activated
  - block current thread until entry in POT is *deactivated* and continue as above or
  - return to metaobject and continue with an alternative execution or
  - allow the concurrent access to the application object through a transaction service
- If the application object is not existent in the Persistent Store, a corresponding message is sent to the metaobject and the application object is created

Storing Java Objects

- Metaobject contacts Persistence Daemon about storing a corresponding application object
  - Persistence Daemon allocates space for the application object and all objects that are reachable from that object
  - Persistence Daemon updates the POT (that is, an activated object becomes deactivated)
  - the application object and all objects that are reachable from that object are serialized

Summary

- Use Reflective Java to implement a Persistent Store for Java objects
- Exploit and enhance existing Java technology, i.e. the Object Serialization API
  - concurrency control
    will be provided by transaction metaobjects
  - security of Persistent Store
    will be realized by applying Cryptography API
  - avoiding redundancy
    will be realized by POT in Persistence Daemon
  - naming and locating objects
    will be realized in Persistence Daemon
- completed by the end of 1996