Author: Pasi Nummisalo

Document: DCF

Date: 1/18/99

Version: 1.0

DCF

1 Introduction

The DCF (Distributed Component Framework) can be used to construct component based applications, which have model and GUI (control/view) parts. The model part is used to implement component's logic and the control/view part is used to present this component to a user. Model components can be connected and manipulated in a visual way to assemble applications. DCF components conforms to the JavaBeans design principles. See picture 2 at appendix 4 for overall architecture view.

2 Architecture

Typically the DCF model part is connected to external client with a developer defined interface. The application logic is constructed by connecting model components together. When the client interface receives a method call it must be translated to an internal transporter event. This event visits at connected components in particular order transferring the client data. Transporter event can carry any type of data. The data value is accessed from the TransporterEvent interface with its name (setProperty, getProperty). Appendix 3 presents TransporterEvent interface.

A ManagedFacade class is used to provide common access to model components from the GUI. The Managed interface is described in appendix 1. This interface is fixed and can be used to access different component properties and methods through dynamic action and property methods. Every method has an identifier parameter (object key) which is used to identify components. The GUI reads unique component identifiers at the connect phase. The model part can also make call backs to the GUI. The Manager interface is presented at appendix 2.

3 Implementation details

3.1 Packages

The common package provides interfaces for GUI and model parts. The dcf package includes model classes and the swinggui GUI classes. See picture 1 at appendix 2 for package dependencies (dashed arrows indicate relations between packages).

3.1 Classes

Every DCF model component must be derived from a ComponentImp abstract class. Components must implement accept method in where the component logic is added. Every outside accessible property must have public set and get methods. See picture 3 at appendix 4 for a UML class diagram.

3.2 Relationship to JavaBeans

DCF components obey JavaBeans 1.0 specification /1/. However some restrictions and additions have been made to support DCF functionality. Every DCF component must implement the ComponentIf interface, components are connected through TransporterEventListener interface, components can be context components (including other components), and they have to register themselves to the facade at loading or creation time.

3.3 Swing GUI

The GUI (view/control) part is provided as it is. The GUI is done merely as proof of concept and the design/implementation should be carefully reconsidered if used further in other projects. One possibility is to use some existing graphics package to do the GUI part and adapt it to use DCF interfaces and logic. Although the GUI part includes some advanced ideas the TOVE project doesn't at this time have resources to fully finish the GUI implementation.

4 Features implemented

DCF logic can be constructed in a visual manner and distributed to other DCF environments. Component properties can be shown and altered from GUI. DCF environment handles storage and retrieval of component's state (based on standard JavaBeans serialization file).

5 Future development

The model part is very easy and clean implementation for e.g. prototyping purposes. In the production environment things like performance, scalability and storage integration should be considered. Emerging standards like the Enterprise Java Beans should give a good support for extending ideas presented in this paper for production environment needs. The new JavaBeans specification (implemented in JDK 1.2) should be studied.

6 References

1. Hamilton,Graham (Editor). JavaBeans Specification 1.01. Sun Microsystems Inc.1997.

APPENDIX 1

```
/*
 * Managed.java
 *
 * Copyright 1999 Helsinki University of Technology
 * ALL RIGHTS RESERVED BETWEEN JANUARY 1996 AND JUNE 1999.
 */

package tove.dcf.common;

/**
 *  Interface for managed components.
 *
 *  @author   Pasi Nummisalo
 *  @version  1.0  (14.12.98)
 */

public interface Managed extends java.rmi.Remote
{
   int connectManager(Manager manager)
      throws java.rmi.RemoteException, dcfException;

   void disconnectManager(Manager manager)
      throws java.rmi.RemoteException, dcfException;

   int createObject(int contextKey, String className)
      throws java.rmi.RemoteException, dcfException;

   void deleteObject(int contextKey, int objectKey)
      throws java.rmi.RemoteException, dcfException;

   void addListener(int componentKey, int listenerKey)
      throws java.rmi.RemoteException, dcfException;

   void removeListener(int componentKey, int listenerKey)
      throws java.rmi.RemoteException, dcfException;

   Object action(int key, String command, Object[] parameters)
      throws java.rmi.RemoteException, dcfException;

   void setProperty(int objectKey, String name, Object parameter)
      throws java.rmi.RemoteException, dcfException;

   Object getProperty(int objectKey, String name)
      throws java.rmi.RemoteException, dcfException;

   void copy(int serviceKey, String destinationName)
      throws java.rmi.RemoteException, dcfException;

   void install(ComponentIf service)
      throws java.rmi.RemoteException, dcfException;
}
```

Table 1. Managed interface

APPENDIX 2

```
package tove.dcf.common;

/**
 *  A callback interface for manager components.
 *
 *  @author   Pasi Nummisalo
 *  @version  1.0  (14.12.98)
 */

public interface Manager extends java.rmi.Remote
{
   void propertyChange(int objectKey, String name, Object newValue)
      throws java.rmi.RemoteException, dcfException;

   Object action(int objectKey, String command, Object[] parameters)
      throws java.rmi.RemoteException, dcfException;
}
```

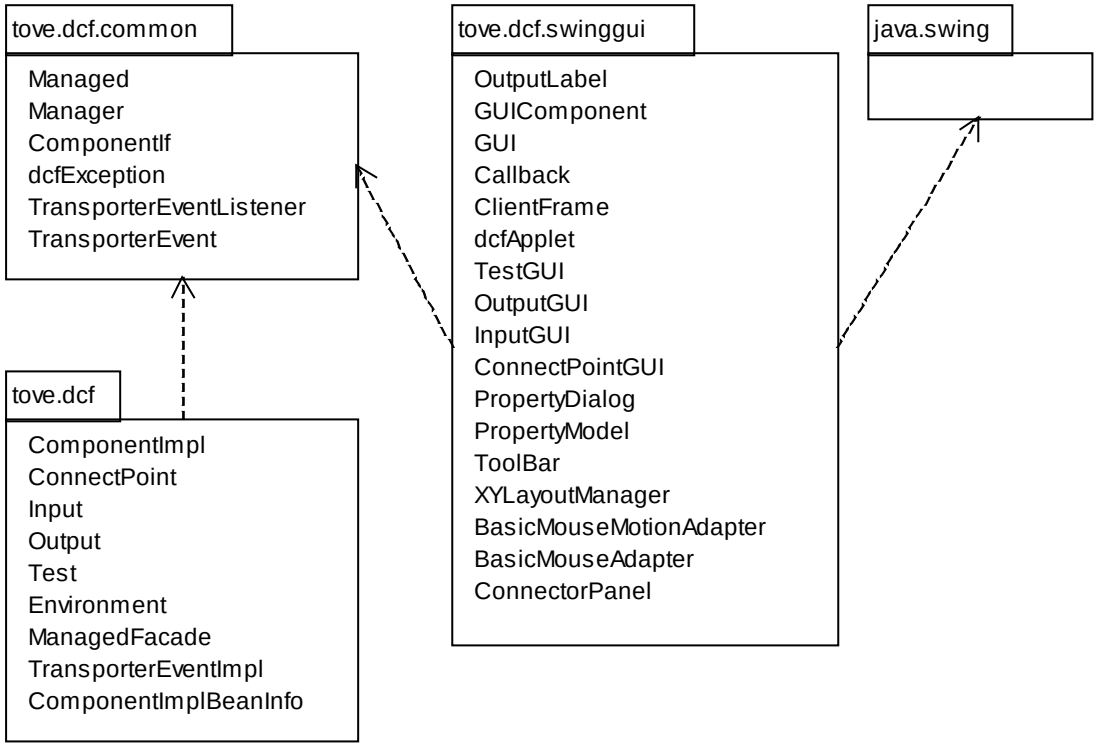


Table 2. Manager interface.

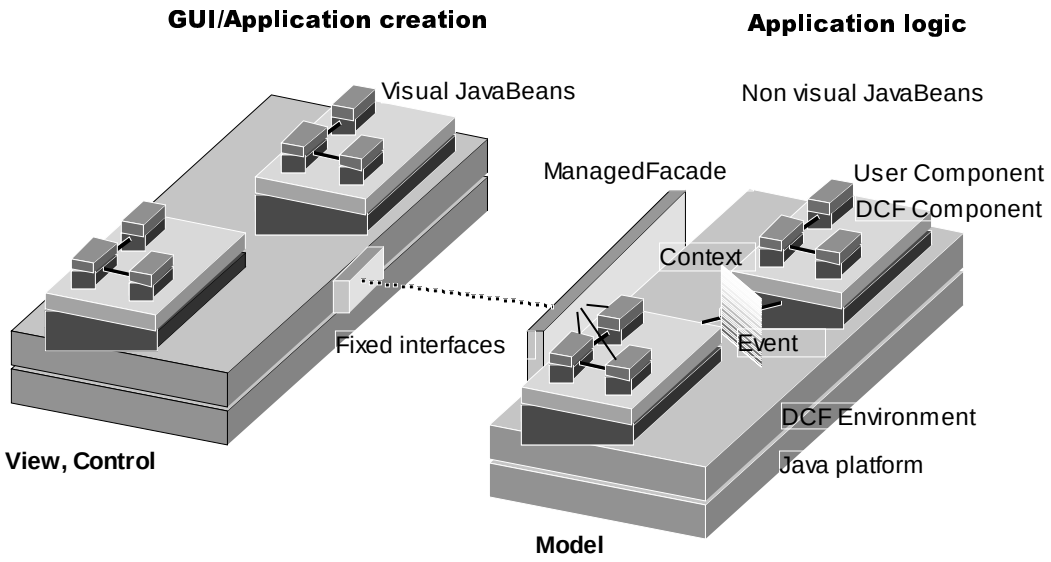
Picture 1. Package dependencies.

APPENDIX 3

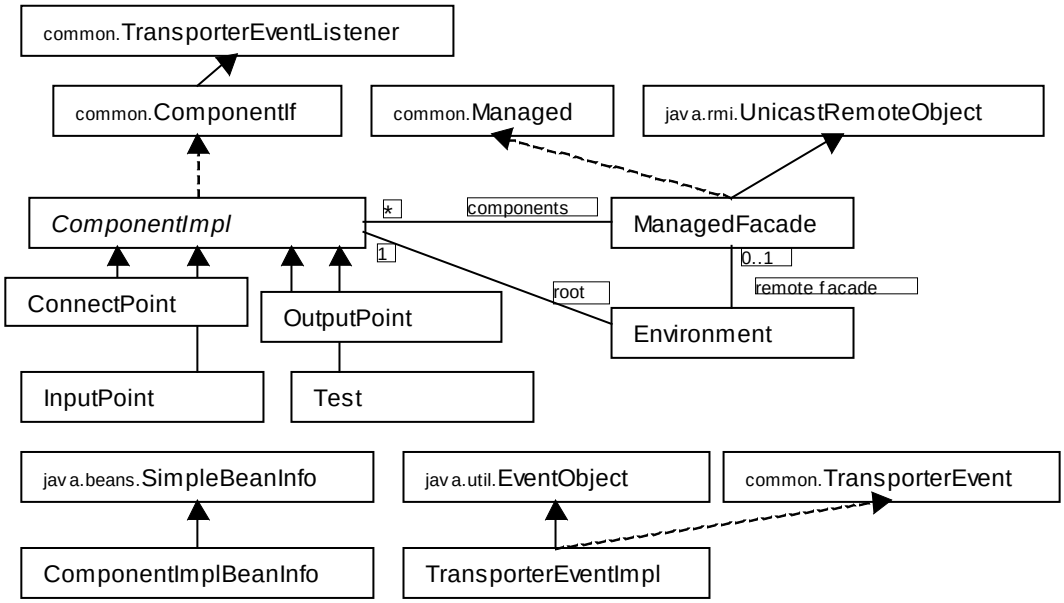
```
/*
 * TransporterEvent.java
 *
 * Copyright 1999 Helsinki University of Technology
 * ALL RIGHTS RESERVED BETWEEN JANUARY 1996 AND JUNE 1999.
 *
 */

package tove.dcf.common;

import java.util.*;

/**
 *  Interface for transporter event.
 *
 *  @author   Pasi Nummisalo
 *  @version  1.0  (14.12.98)
 */

public interface TransporterEvent
{
   public void send();
   public void setListeners(Vector listeners);
   public void setListener(TransporterEventListener listener);
   public void continueExecution();
   public void setProperty(String name, Object value);
   public Object getProperty(String name);
   public Hashtable getProperties();
   public void setCriteriaMet();
   public boolean isCriteriaMet();
   public void setState(TransporterEventListener listener);
}
```

Table 3. Transporter Event interface.

APPENDIX 4



Picture 2. Overall architecture.



Picture 3. Class hierarchy.