The Evaluation of Adaptive Reflective Middlewares in Distributed Systems

Farahnaz Poorfard
AmirKabir University of Tehran
Computer Engineering Department

Advisor: Dr. Hossein Pedram
Examiner: Dr. Mahdi Dehghan

Oct. 2007
Outline

- Introduction
- Current Middlewares
- New requirements and new Ideas
- Reflective Component Model
- OpenCOM
- OpenORB
- CFs
- Evaluations
- Future Works
- References
What is a Distributed System?

A distributed system is a collection of independent computers that appears to its users as a single coherent system.
A middleware system constitutes a set of services that aim at facilitating the development of distributed applications in heterogeneous environments.
An Abstraction of Middleware in a Distributed System
Some Current Well-known Middlewares

- CORBA
- COM
- XPCOM
- Java RMI
- WSDL
But We Have Some New Requirements

- Multimedia
- Real-time applications
- Ad-hoc networks
- Mobile networks
- Embedded systems
- Critical applications
- 24x7 applications
- ...

21/10/2007
They Have Issued New & “Separate” Versions for New Requirements

- CORBA
  - Real-time CORBA
  - Minimum CORBA
  - Data Parallel CORBA
  - Fault-tolerance CORBA
  - CORBA/e
  - CORBA/IIOP

- COM
  - COM+, .NET, DCOM, COMERA

- Java RMI
  - J2SE, J2EE, J2ME, Java RMI-IIOP
We Need Some New Middlewares

- Light weight
- Configurable
- Reconfigurable
- Dynamic Reconfigurable
- Adaptable
- Scalable
- Meet the new requirements
- …
Some New Ideas

- Object Models
- Component Models
- Component-object Models
- Reflective-component Models
- Event-based Models
- Horizontal Decomposition Models
- ...

21/10/2007
OpenCOM: An Adaptive Reflective Component Middleware

It has built on top of COM, with saving some properties of COM:

- vtable
- IDL
- GUIDs
- IUnknown Interface

And adding some new concepts:

- Making Explicit the Dependencies of each Component,
- Adding Basic Mechanism-level Functionality for Reconfiguration,
- Supporting for Pre- & Post- Method Call Interception.
Components in OpenCOM …

Components are encapsulated units of functionality and deployment that interact with other components exclusively through “interfaces” and “receptacles”.

… and in the view of Reflection

In Reflective model, a component can reason about and alter itself, in terms of different situations.
Basic Concepts in OpenCOM

- **Interface:** A unit of service provision.
- **Receptacle:** A unit of service requirement.
- **Connection:** Binding a pair of interface and receptacle, for creating a configuration of components.
OpenCOM Architecture
Interceptors & Receptacles
Mechanisms
Component Frameworks (CFs)

In OpenORB, we add CFs.

Definition of CFs:

“Collections of rules and interfaces that govern the interaction of a set of Components plugged into them.”
Some Sample CFs in OpenORB

- **Binding Layer**
  - Binding CF
  - Protocols
  - BT implementations

- **Comms Layer**
  - Protocol CF
  - Multimedia Streaming CF

- **Resource Layer**
  - Buffer Mgt. CF
  - Transport Mgt. CF
  - Thread Mgt. CF

- **Middleware Top CF**
  - Buffer policies
  - Transport plug-ins
  - Schedulers
The Benefits of CFs

- Abstraction between the CF’s components and the other parts of the system,
  - Understanding of system
  - Maintainability of system
- Reusability
  - Simplicity for system developers
- Creating of light-weight components for initialization phase,
- Disallowing the plug-ins from directly depending on external components,
  - Security, Integration of management, Scalability, and coupling
- Preparing the “nesting”, …
A Simple Sample Configuration with CFs
createInstance(/*[in]*/ REFCLSID rclsid,
    /*[out]*/ IUnknown **RPC_FAR *RPC_FAR *ppIUnknown,
    /*[string][in]*/ const unsigned char **RPC_FAR *name);

deleteInstance(/*[in]*/ IUnknown **RPC_FAR *pIUnknown,
    /*[out]*/ IReceptacle **RPC_FAR *RPC_FAR ppOCM_IRecps[]),
    /*[out]*/ int **RPC_FAR *pcElems);

connect(/*[in]*/ IUnknown **RPC_FAR *pIUnkSource,
    /*[in]*/ IUnknown **RPC_FAR *pIUnkSink,
    /*[in]*/ REFIID iid,
    /*[in]*/ unsigned long recpID,
    /*[out]*/ unsigned long **RPC_FAR *pConnID);

disconnect(/*[in]*/ unsigned long connID,
    /*[out]*/ IReceptacle **RPC_FAR * RPC_FAR * ppOCM_IRecp);

cgetConnectionInfo(/*[in]*/ unsigned long connID,
    /*[out]*/ OCM_ConnInfo_t **RPC_FAR * RPC_FAR * ppConnInfo);

cfreeConnectionInfo(/*[in]*/ OCM_ConnInfo_t **RPC_FAR * pConnInfo);

cenumComponents(/*[out]*/ IUnknown **RPC_FAR * RPC_FAR * RPC_FAR * ppComps[],
    /*[out]*/ int **RPC_FAR * pcElems);

cGetComponentName(/*[in]*/ IUnknown **RPC_FAR *pIUnknown,
    /*[out]*/ unsigned char **RPC_FAR * ppName);

cGetComponentPIUnknown(/*[string][in]*/ const unsigned char **RPC_FAR *name,
    /*[out]*/ IUnknown **RPC_FAR * RPC_FAR * ppIUnknown);

cGetComponentCLSID(/*[in]*/ IUnknown **RPC_FAR *pIUnknown,
    /*[out]*/ CLSID **RPC_FAR *pclsid);
... and in OpenORB

// ICFMetaArchitecture
get_internal_components(/* [out] */ IUnknown __RPC_FAR *__RPC_FAR *__RPC_FAR FAR ppComps[ ], /* [out] */ int __RPC_FAR *__RPC_FAR *pcElems);
get_bound_components(/* [in] */ IUnknown __RPC_FAR *comp, /* [out] */ ConnectedComponent __RPC_FAR *__RPC_FAR ppConnections[ ], /* [out] */ int __RPC_FAR *__RPC_FAR *pConnectedElements);
get_internalBindings(/* [out] */ unsigned long __RPC_FAR *__RPC_FAR ppConnIDs[ ], /* [out] */ int __RPC_FAR *__RPC_FAR *pcElems);
local_bind(/* [in] */ IUnknown __RPC_FAR *pIUnkSource, /* [in] */ IUnknown __RPC_FAR *pIUnkSink, /* [in] */ REFIID iid, /* [out] */ unsigned long __RPC_FAR *__RPC_FAR *pConnID);
break_local_bind(/* [in] */ unsigned long connID);
create_component(/* [in] */ CLSID clsid, /* [string][in] */ const unsigned char __RPC_FAR *name, /* [out] */ IUnknown __RPC_FAR *__RPC_FAR *ppIUnkUnknown);
insert_component(/* [in] */ IUnknown __RPC_FAR *pIUnkUnknown);
remove_component(/* [in] */ IUnknown __RPC_FAR *pIUnkUnknown);
delete_component(/* [in] */ IUnknown __RPC_FAR *pIUnkUnknown);
newexpose_interface(/* [in] */ IID rintf, /* [in] */ IUnknown __RPC_FAR *pComp);
init_arch_transaction( void );
commit_arch_transaction( void );
rollback_arch_transaction( void );
access_CF_graph_lock(/* [in] */ int index);
release_CF_graph_lock(/* [in] */ int index);
update_readers_count(/* [in] */ int Value);
unexpose_interface(/* [in] */ IID rintf, /* [in] */ IUnknown __RPC_FAR *pComp);
expose_receptacle(/* [in] */ IID rintf, /* [in] */ IUnknown __RPC_FAR *pComp,
/* [in] */ RecptType_t recptType);
unexpose_receptacle(/* [in] */ IID rintf, /* [in] */ IUnknown __RPC_FAR *pComp);
unexpose_all_interfaces(/* [out] */ int __RPC_FAR *__RPC_FAR *piElements);
unexpose_all_receptacles(/* [out] */ int __RPC_FAR *__RPC_FAR *piElements);
“Accept” Component in OpenORB

The Accept Component checks the matching between a reconfiguration with determined rules.
An Example of xml Files

```xml
<ReMMoC_Configuration>
  <Interfaces>
    <Interface>{4525012C-47A6-465C-9F15-81B4035DD329}</Interface>
  </Interfaces>
  <Components>
    <Component>
      <Name>Adder</Name>
      <ID>{DA47A723-8CEA-4D4A-9551-C3714DC83664}</ID>
    </Component>
    <Component>
      <Name>Multiplier</Name>
      <ID>{90755991-4921-494E-A726-F2FC78A33BB0}</ID>
    </Component>
    <Component>
      <Name>Calculator</Name>
      <ID>{1C224E5B-4CCC-4AEC-B9C6-2FB82FC4D0B9}</ID>
      <Connections>
        <Interface>{1EA921FB-5FAB-4172-8206-1BA7880551C0}</Interface>
        <Interface>{37DCF638-2DC6-4E7A-914C-5708BD25A0D6}</Interface>
      </Connections>
    </Component>
  </Components>
</ReMMoC_Configuration>
```
## Results for OpenCOM

<table>
<thead>
<tr>
<th>command</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoInitialize</td>
<td>15 (once)</td>
</tr>
<tr>
<td>CoCreateInstance</td>
<td>31 (once for runtime component)</td>
</tr>
<tr>
<td>createInstance</td>
<td>32 (once for each component)</td>
</tr>
<tr>
<td>startup</td>
<td>0</td>
</tr>
<tr>
<td>connection (connect)</td>
<td>0</td>
</tr>
<tr>
<td>operation (method call)</td>
<td>0</td>
</tr>
<tr>
<td>interception</td>
<td>46 (once for each proposed interface)</td>
</tr>
</tbody>
</table>
Results for OpenORB

<table>
<thead>
<tr>
<th>command</th>
<th>time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoInitialize</td>
<td>15 (once)</td>
</tr>
<tr>
<td>CoCreateInstance</td>
<td>31 (once for runtime component)</td>
</tr>
<tr>
<td>createInstance</td>
<td>32 (once for each CF)</td>
</tr>
<tr>
<td>create_component</td>
<td>32 (once for each component inside CF)</td>
</tr>
<tr>
<td>startup</td>
<td>0</td>
</tr>
<tr>
<td>connection (connect/local_bind)</td>
<td>0</td>
</tr>
<tr>
<td>operation (method call)</td>
<td>0</td>
</tr>
<tr>
<td>interception</td>
<td>46 (once for each proposed interface)</td>
</tr>
<tr>
<td>addConfiguration</td>
<td>0</td>
</tr>
<tr>
<td>expose_interface</td>
<td>47</td>
</tr>
<tr>
<td>init_arch_transaction</td>
<td>0</td>
</tr>
<tr>
<td>commit_arch_transaction</td>
<td>16</td>
</tr>
</tbody>
</table>
Test1: Evaluation of Time vs number of Components

![Bar Chart]

- **X-axis**: Number of Components
- **Y-axis**: Time (ms)

Series 1:
- 0 components: 0 ms
- 1 component: 50 ms
- 2 components: 100 ms
- 3 components: 150 ms
- 4 components: 200 ms
- 5 components: 250 ms
- 6 components: 300 ms
Test2: Comparison between “Simple” and “Intercepted” Components

![Graph showing comparison between Simple and Intercepted Components]
Test3: Evaluation of Time vs number of Interfaces

![Bar chart showing time (ms) vs number of components for one and two interfaces.]

- With One Interface
- With Two Interfaces
Test4: Evaluation of Time with adding CFs

![Graph](image)

- **x-axis**: number of components
- **y-axis**: time (ms)

Graph comparison between OpenORB and OpenCOM.
Test5: Evaluation of Time with adding CFs (Complete Operations)

![Graph showing the relationship between the number of components and time for OpenCOM and OpenORB. The graph shows a linear increase in time with an increase in the number of components.]
Test5: Evaluation of Time with adding CFs (Complete Operations)

![Bar chart showing time (ms) vs number of components for OpenCOM and OpenORB](chart)

- **OpenCOM**
- **OpenORB**
Conclusion

- Fix time for creating each new component
- Fix time for each intercepting
  - Overhead, but instead reconfiguration operation
- Number of interfaces
- Connection/disconnection
- Starting a component
- Calling methods
- Fix time for each CF
- Deletion phase
RUNES Middleware

The Whole Scenario for Road Tunnels:
GridKit Middleware

Suitable for Flooding Scenarios:
Future Works …

- Evaluating in remote mode,
- Using new services (like multimedia),
- Defining filter plug-ins in multimedia CF,
- Building some hierarchies from CFs,
- Installing the GridKit, Genie, …
- MPI?
- …
References (1)


References (4)

References (5)
