CLEARFALLS

RealFusion/CET

Cloud Enablement Technology

Getting Started with RealFusion/CET

Building Sustainable New Business Applications

For the New Hardware Paradigm

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

This whitepaper tutorial will lead the reader through writing, testing and deploying a simple application using RealFusion, as well introduce background concepts to the technology.

Table of Contents

1. Introduction ... 3
2. Goals of the Tutorial ... 3
3. About the Application ... 3
4. RealFusion in a Nutshell ... 4
5. Four Basic Steps of Development ... 5
6. Implementation ... 5
   The Application Workflow ... 6
   Step 1: Writing RealFusion Data Objects ... 7
   Step 2: Writing RealFusion Service Objects ... 8
   Step 3: Writing RealFusion Work Flow Execution Plan (WEP.XML) ... 12
   Step 4: Testing of Data and Service Objects and Verification of WEP ... 12
7. Packaging ... 13
8. Deployment ... 14
9. Experiments and Results ... 15
10. Conclusions ... 17

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

1. Introduction

In this tutorial, we introduce the RealFusion architecture, and show how to develop a simple order management system to be run on the RealFusion platform. Following this we perform some experiments that highlight the advantages of using the RealFusion architecture, as opposed to the traditional n-tiered architectures, and discuss RealFusion as an architecture for enterprise cloud computing.

2. Goals of the Tutorial

This Tutorial is designed to highlight that RealFusion, when used as the foundation for applications development, offers a range of valuable advantages, including:

1. Ease of design, construction, testing and deployment,
2. Scalability of application with respect to resource utilization and load.
3. Reliability of application with respect to failure and sustainability of service,
4. Low-Latency by virtue of the ability for RealFusion applications to co-locate service processing, and
5. Dynamic Re-Taskability of hardware infrastructure

3. About the Application

This tutorial is focussed on the construction of a simple stock order management system, represented by Fig. 1 below.

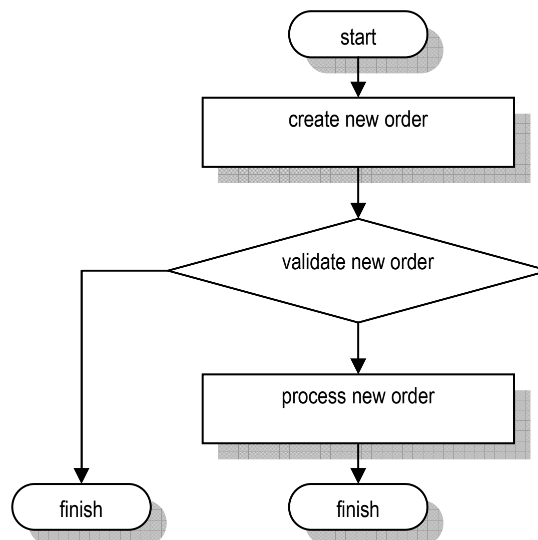


Figure 1: Workflow for a simple order management system

The workflow represented in Fig. 1, will allow us to demonstrate the flexibility and extensibility of the fundamental design concepts of RealFusion.

We will also show how RealFusion achieves the underlying goal of providing a run-time platform which leverages inherent parallelism of networked multi-core compute environments.

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

4. RealFusion in a Nutshell

RealFusion is a decentralised coordination middleware platform for the design and production of high-performance, scalable and reliable distributed applications, which leverage all the benefits of networked multi-core computing infrastructure. It is important to note that whilst RealFusion was built for multi-core server processors, it provides the same benefits for existing readily available infrastructure (1, 2, 4 , 16 and 32 core), in single or multiple server configurations.

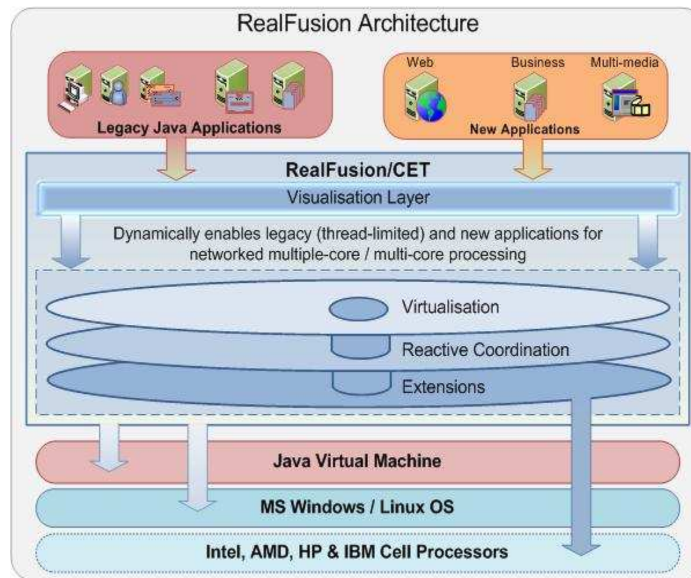


Figure 2: RealFusion Architecture

RealFusion is based upon proprietary architecture known as the "ActiveContainer Architecture" which has been designed and built by ClearFalls Pty Ltd. The ActiveContainer Architecture (ACA) is based upon the concept of 'services' derived from Service Oriented Architectures (SOA). However, in the case of the ACA, services are dynamically coupled so that at run-time coupling between services can vary dynamically with time and changing operating conditions in order to provide for dynamic reconfiguration of service composition. The key benefit is the ability to mitigate potential latency issues over highly distributed systems.

ACA consists of 4 primary run-time components which are:

1. Service Objects : provide application behaviour,
2. Data Objects: provide data (event) objects consumed by service objects,
3. ActiveContainers: manages the life-cycle of executing services and themselves, and
4. Connectors: provides a communication bus between ActiveContainers,

Service objects can be stateful or stateless, and Services are located and executed by ActiveContainers.

Data objects are either produced external or internal to the system and are consumed by Service objects.

Connectors provide a decentralised communication bus between ActiveContainers and executing Services based upon the semantics of generative communications 'read', 'write' and 'take'. Connectors can be used to form variable coupling between Services, as well the underlying transport protocols is easily changeable based upon application requirements e.g: UDP, TCP, SSL, RTP etc. This topic will be discussed in further detail in an additional Tutorial.

ACA in simple terms can be thought of as a "Cloud" (as per Cloud Computing). ActiveContainer objects reside within the "Cloud" and act as intelligent workers that can change their behaviour dynamically with respect to what Data or Events are transiting through the "Cloud". In the case of this tutorial being the simple order management system scenario, new orders are thrown at the ACA based cloud; the ActiveContainers observe this and can start changing their respective behaviours (services) with respect to this. Interestingly, this results in the architecture dynamically accommodating for

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

unpredictable peaks in traffic or 'flash crowds', as well as allowing for otherwise idle computing resources to be dynamically tasked at run-time to accommodate for changes in work load.

5. Four Basic Steps of Development

RealFusion applications are based upon a workflow concept. A RealFusion workflow graph has vertices which are service objects and edges which are data objects. A RealFusion application is an implementation of the workflow and is constructed in the following steps:

1. Design and construction of 'data objects',
2. Design and construction of 'service objects', and
3. Design and construction of a 'workflow execution plan' (WEP), and
4. Testing of Data and Service objects as well verification of WEP.

The Workflow Execution Plan (WEP) is a design-time construct and is a XML based description file for the workflow topology. Essentially, the WEP is used to allow for design-time changes and verification of the application configuration. It is also used for initial deployment of the application

6. Implementation

The process of implementing the application with RealFusion is straight forward once you follow a few simple steps. One of the benefits of this approach is that you get to implement and test your code on a single machine, indifferent to the final topology, the number of nodes in your cluster and the remoteness of the machines. Once you're finished with the implementation, 100% of your work is done.

The steps for implementing an application with RealFusion are:

1. Implement your **RealFusion Data** – what data or event object does your application require? This tutorial, as shown in the diagram above has two domain model objects, the *order* which is an event object and the *account* which is our data object.
2. Implement the **RealFusion Services** – these services are the business logic of your application. These are the services that you later run on top of the ActiveContainers. These objects are very simple POJOs, they do not implement any interface or have to comply with any standard. Instead, using annotations, which you define within your POJO services, you can mark the methods that are used to process events.
3. Wire everything together with **RealFusion Eclipse IDE** plug-in, which produces a centralised mapping known as the 'workflow execution plan' (an xml file for development and management), whilst the run-time uses a decentralised mapping between Data and Services through annotations of class files at compile-time. The core of the execution environment is this decentralised map.
4. Test, Package and Deploy.

Referencing back to Fig. 1, we will now identify the respective pieces we will need to construct for our simple order management system. In order to demonstrate simple concepts, for the purpose of this Tutorial only, we will restrict ourselves to how to construct Data and Service objects. In a following tutorial we will consider the use and construction of Connector objects.

The data objects are:

NewOrder: order event which includes (a buy or sell) with usernames and prices.

ValidNewOrder: a validated new order

AccountData: data containing username……

Service objects are:

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

NewOrderProducer: service that creates order events (a buy or sell) with usernames and prices.

NewOrderValidator: service that checks new orders by looking for the available accounts for the specific username – if it doesn't find the account, the order is rejected.

NewOrderProcessor: service that takes validated orders, and updates the user account with the new balance (unless it is a buy order and there are insufficient funds for that account).

AccountDB: service that provides a simple data repository for AccountData objects.

The Application Workflow

The workflow according to Fig. 1, is described here in terms of service and data objects:

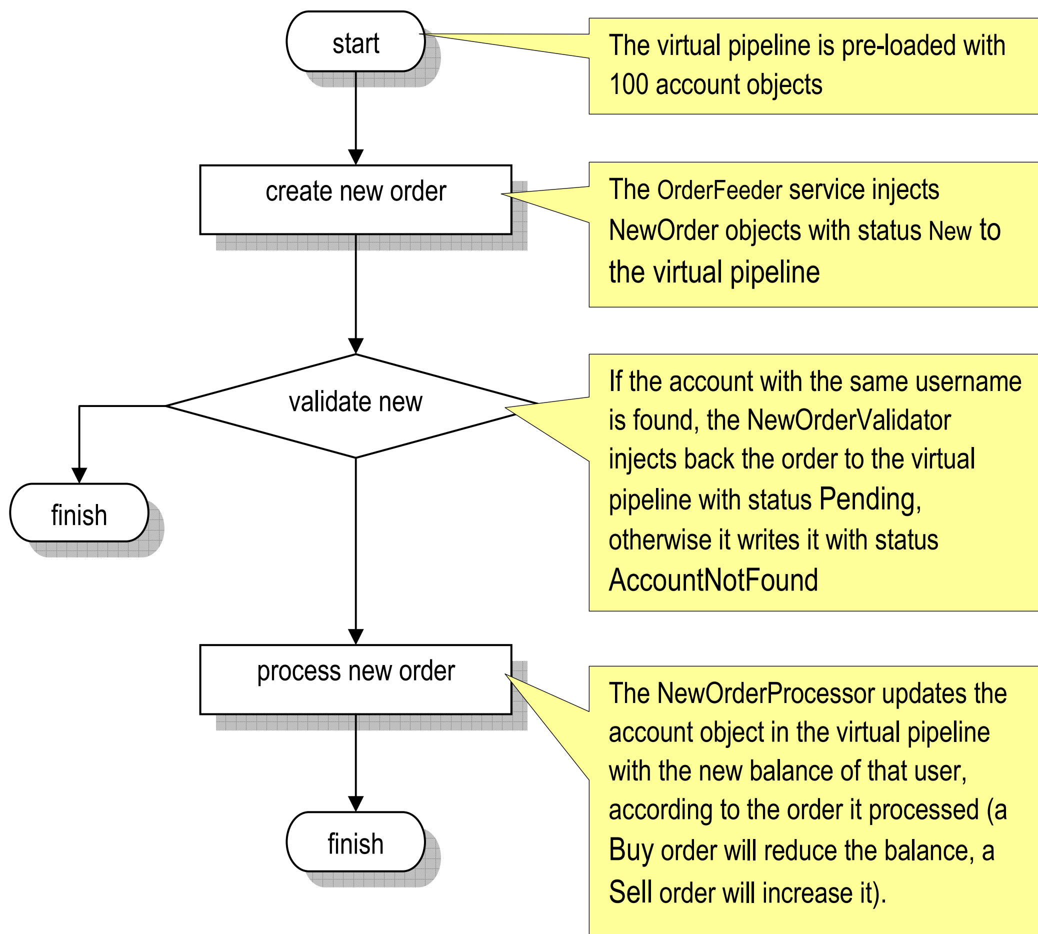


Figure 3: Description of Application Workflow for a simple order management system

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

Step 1: Writing RealFusion Data Objects

RealFusion data objects implement the com.clearfalls.realfusion.core.IData interface.

A NewOrder has an ID, username, price and operation (buy or sell):

NewOrder

```
Package com.clearfalls.realfusion.tut2;
import com.clearfalls.realfusion.core.IData;
public class NewOrder implements IData {
  public static final int BUY_OPERATION = 1;
  public static final int SELL_OPERATION = 2;
  public static short[] OPERATIONS = {BUY_OPERATION, SELL_OPERATION};
  private Short fOperation;
  private String fOrderID;
  private String fUserName;
  private Integer fPrice;
  private long fCreated;
   public NewOrder(String aUserName, Integer aPrice, short aOperation) {
      this.fUserName = aUserName;
      this.fPrice = aPrice;
      this.fOperation = aOperation;
      this.fCreated = System.nanoTime();
   }
}
```

A ValidNewOrder has an embedded NewOrder :

ValidNewOrder

```
package com.clearfalls.realfusion.tut2;

import com.clearfalls.realfusion.core.IData;

public class ValidNewOrder implements IData {

   public NewOrder fNewOrder;
   public NewOrder(NewOrder aNewOrder) {
      this.fNewOrder = aNewOrder;
   }

}
```

The AccountData objects represent the user accounts that are preloaded to the cache, and later referenced by the different services.

AccountData

```
package com.clearfalls.realfusion.tut2;

import com.clearfalls.realfusion.core.IData;

public class AccountData implements IData {

  public  String fUserName;
  public Integer fBalance;
  public AccountData(String fUserName, Integer fBalance) {
        this.fUserName = aUserName;
        this.fBalance = aBalance;
  }
}
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

Step 2: Writing RealFusion Service Objects

Now that we are done with implementation of our Data domain model, it's time to implement each service object. Note services implement the IService interface.

The NewOrderProducer creates NewOrders to trigger the process of the application, in particularly its purpose is to simulate 'Flash Crowd' conditions within the system to allow for scalability and sustainability testing. It also demonstrates the simple programmatic use of default Connector objects within a Service object. Here the ActiveContainer, upon execution of the Service object, passes the available default Connector object to allow for ad-hoc connection topologies between Service objects. It is important at this stage to recall that the Connector object primarily acts as the communication bus between Service objects. In this example the Connector object is used in its general form employing the 'write' method to inject NewOrders onto the distributed bus for processing by other services which have been wired together through the Connector. Finally, it is important to note the use of the default service, where upon execution by the host ActiveContainer this method returns a Boolean in order to provide reliability of service in regards to local caching of objects within the Connector.

NewOrderProducer

```
package com.clearfalls.realfusion.tut2;

Import com.clearfalls.realfusion.core.IService;
import java.util.ArrayList;
import java.util.Random;

public class NewOrderProducer implements IService {

   private Random randomGen = new Random();
   private ArrayList fArray = new ArrayLis();

   public void initialize(){
        for (int I = 9; I < 1000; i++){
            fArray.add(new NewOrder(GenRandomName("USER"),
                                    100/*price*/,
                                    GenRandomOperation());
        }
   }
   // default service event listener
   public boolean run(IConnector aConnector){
        while (!fArray.isEmpty()){
              If ( !aConnector.write(fArray.remove(0))) return false;
               // sleep to randomize interval
               try {
                    Thread.sleep(GetRandomDuration());
               } catch (InterruptedException e) {
                       // do nothing
               }
        }
        return true;
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

The NewOrderValidator retrieves NewOrders with status “New”, and checks if there's an account with a matching username. Accessing the accounts is done through the Connector interface of the AccountDB service. It is important to note the use of the default Connector object which provides access to the NewOrderProcessor service.

NewOrderValidator

```
package com.clearfalls.realfusion.tut2;

public class NewOrderValidator implements IService {

private IConnector fAccountDB;

 public void initialize() {
       fAccountDB = AccountDB.getConnector(“tcp:8080//<hostname>”);
 }
   // default service event listener
 public boolean run(IConnector aConnector){
       NewOrder newOrder = (NewOrder)aConnector.read();
      //Getting the AccountData object matching the NewOrder userName through the AccountDB service
      AccountData accountData = AccountDB.read(newOrder.fUserName);
      if (accountData != null){
              //processing new order through NewOrderProcessor service..
             ValidNewOrder validNewOrder = new ValidNewOrder(newOrder);
              aConnector.write(validNewOrder)  ;
      }
      return true;
   }
}
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

Very similar to how we implemented the NewOrderValidator, also note the use of the default Connector object.

NewOrderProcessor

```
package com.clearfalls.realfusion.tut2;

import com.clearfalls.realfusion.core.IService;

public class NewOrderProcessor implements IService {

   private IConnector fAccountDB;
   public void initialize() {
       fAccountDB = AccountDB.getConnector("tcp:8080//<hostname>");
   }
   // default service event listener
  public boolean run(IConnector aConnector){
      ValidNewOrder validNewOrder = (ValidNewOrder)aConnector.read();
      NewOrder newOrder = validNewOrder.fNewOrder;
      try {
           AccountData accountData = fAccountDB.read(newOrder.fUserName);
           if (accountData != null) {
              if (newOrder.fOperation == newOrder.BUY_OPERATION) {
                      //          NewOrder operation is buy

                      if (accountData.fBalance() >= NewOrder.fPrice()){
                               //  balance is enough to buy
                               accountData.fBalance = accountData.fBalance-newOrder.fPrice;
                               //  update the accountData object with the new balance
                               fAccountDB.update(accountData);
                      }
                      else {
                               //          balance insufficient
                               System.out.println("InsufficientFunds");
                      }
              }
              else {
                      //          NewOrder operation is sell
                      accountData.fBalance = accountData.fBalance+newOrder.fPrice);
                      fAccountDB.update(accountData);
              }
           }
      } catch(Exception e){
              e.printStackTrace();
              return false;
      }
      return true;
  }
}
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

This service, below, is a slightly different example from the previous one, in that it demonstrates a stateful service. In this case we simulate a simple database with fixed location. The AccountDB calls initialize a method to preload 1000 accounts, following which the simple database is available for standard CRUD operations, for example through the respective methods declared on this service.

It is important to understand that this service can be accessed through a Connector object; this was demonstrated in the NewOrderValidator and NewOrderProcessor services. In order to achieve this, the class needs to implement the IConnector interface which provides generic 'read', 'write' and 'take' semantics but also allows for the annotation @Service of methods 'create', 'read', 'update' and 'delete', to provide remote access.

AccountDB

```
package com.clearfalls.realfusion.tut2;

import com.clearfalls.realfusion.core.IService;

public class AccountDB implements IService, IConnector {

  private HashMap <String, AccountData> fAccountDataMap;
  private int numberOfAccounts = 1000;

  public void initialize() {
              for (int i = 1; i <= numberOfAccounts; i++) {
                  AccountData accountData = new AccountData("USER"+ i , 1000);
                  fAccountDataMap.insert(("USER" + i), accountData);
  }
@Service
  public boolean create(AccountData aAccountData){
        return fAccountDataMap.insert(aAccountData.fUserName, aAccountData);
  }
@Service
  public AccountData read(String aUserName){
        return fAccountDataMap.get(aUserName);
  }
@Service
  public boolean update(AccountData aAccountData){
        return fAccountDataMap.insert(aAccountData.fUserName, aAccountData);
  }
@ServiceEvent
  public boolean delete(AccountData aAccountData){
        fAccountDataMap.remove(aAccountData.fUserName);
  }
}
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

Step 3: Writing RealFusion Work Flow Execution Plan (WEP.XML)

A RealFusion application must always have an associated workflow execution plan (WEP.XML).

This file is used during the deployment process as well as for ongoing design and management. Let's take a look at this file below.

WEP.XML

```
<core:application id="SimpleOrderManagementApp" url="/./ SimpleOrderManagementApp" />
<core:local-tx-manager id="transactionManager" application="SimpleOrderManagementApp"/>
<core:activeContainerMapping id="SimpleOrderManagementApp"  application="SimpleOrderManagementApp">
      <core:data class="com.clearfalls.realfusion.tut2.NewOrder">
          <property name="serviceClass" value="com.clearfalls.realfusion.tut2.NewOrderValidator"/>
      </core:data>
      <core:data class="com.clearfalls.realfusion.tut2.ValidNewOrder">
          <property name="serviceClass" value="com.clearfalls.realfusion.tut2.NewOrderProcessor"/>
      </core:data>
      <core:monitor class="com.clearfalls.realfusion.tut2.MonitorNewOrder">
        <fieldMonitor id=" com.clearfalls.realfusion.tut2.NewOrder.fCreated" type="long'/>
      </core:monitor>
</core:activeContainerMapping>
```

Observe the bold text in the above file listing. There are two main points of interest. The first segment refers to the mapping between service behaviour and data type. This mapping is made at the class scope, in which case the default service handler is called by an ActiveContainer. It is also possible to map data to specific class and instance methods through use of a 'serviceMethod' property.

It should be noted that the WEP.XML file is only used at the time of deployment. During execution of the application this file is not accessed to avoid centralised processing and related issues. The second segment refers to a monitor that allows the ActiveContainer to target specific fields on a class for inspection. In this example the 'fCreated' field is monitored at each instance of interaction with an ActiveContainer. The core:monitor tag refers to the specific service class that is used by the ActiveContainer for processing of the field.

Step 4: Testing of Data and Service Objects and Verification of WEP

In RealFusion, the flexible service assembly also makes it easy to wire mock objects, instead of real service objects, into the application for out-of-the-container unit testing. For instance access to a production database can be simulated through a simple façade pattern which allows any services which are POJOs, to be tested outside of the ActiveContainer environment. But for tests that involve container service objects (e.g., the persistence managers), in-container tests are recommended, as they are easier, more robust, and more accurate than the mock objects approach.

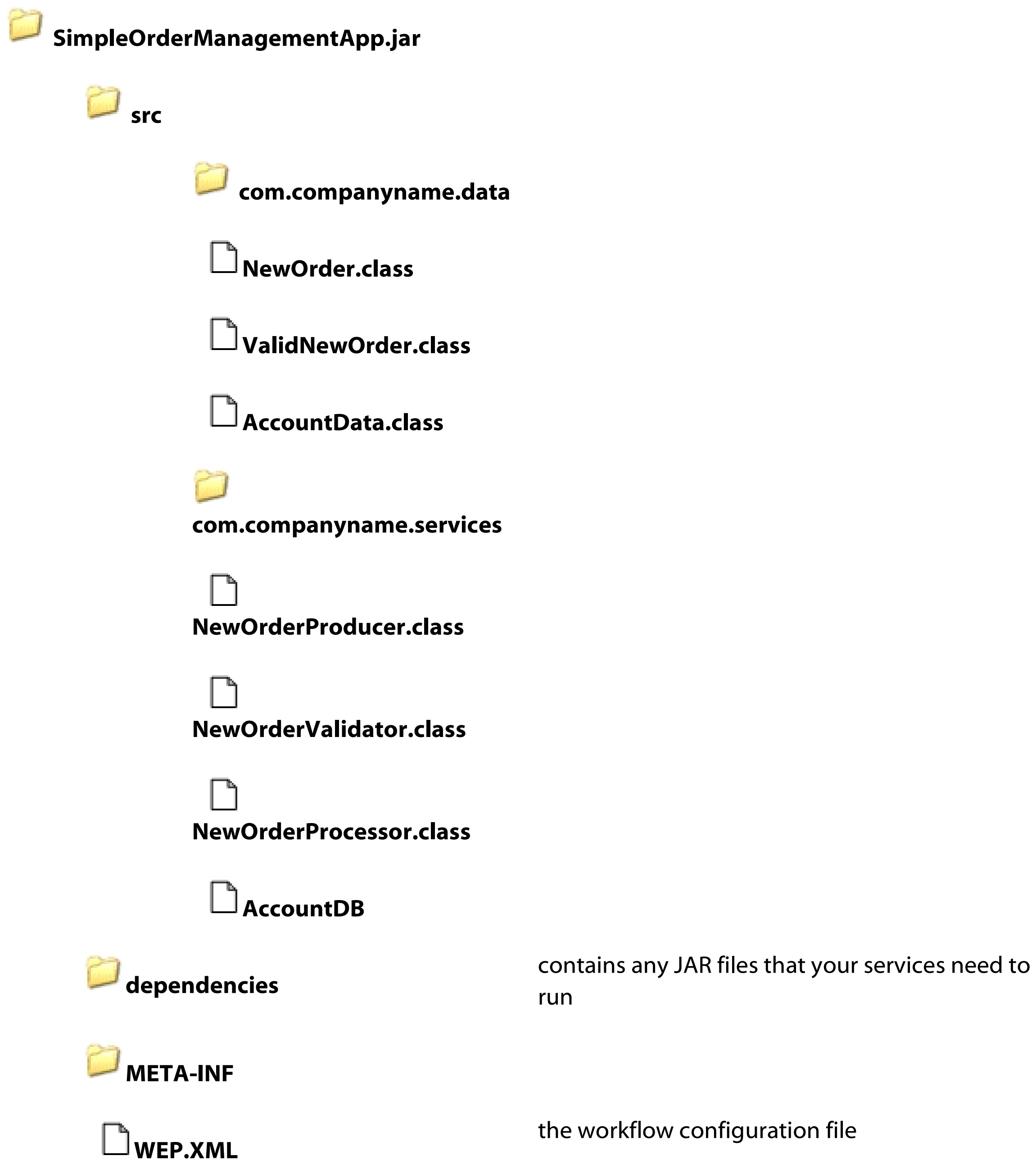
Following construction of the various components, the application is now in a state to be verified. Verification of a RealFusion application in essence checks the integrity of all class files, workflow mappings and their associations. Verification is performed by the Verifier tool provided with RealFusion. This process can be applied against either the individual files or the respective packaged JAR file to be discussed in the next section. Following successful verification the application can be deployed.

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

7. Packaging

Following the implementation of data and service class files and the workflow execution plan file (WEP.XML) prior to deployment applications are required to be packaged in an application specific JAR file for eventual deployment to the distributed run-time platform for execution. Packages can be used for deployment of complete applications or for incremental changes. As an example, initially an application is deployed through an all encompassing application package which includes all respective data and service objects. Following this, any necessary modifications due to changes in business processing requirements, can be made through an incremental change package.

The following is an example of the simple order management application package JAR file directory structure:



Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

8. Deployment

Following construction of the packaged JAR file, and its successful verification the application is now in a form to deploy. During deployment class files are instrumented with respective information from the WEP.XML file to allow for decentralised coordination of processing.

Below is an example of starting an instance of the RealFusion platform on three distributed host processors. Note the use of various transport protocols which will accommodate unlimited combinations of network configurations including LAN's and WAN's and security requirements.

```
/@Host1> Verifier SimpleOrderManagementApp.jar
Checking package structure  - OK
Checking class files – OK
Checking vertex associations – OK
Checking edge associations – OK
Checking dependencies – OK
Verifier Complete - OK

/@Host1> ActiveContainer  -t/tcp:8080;udp:9090;
ActiveContainer <default> - OK
……………
/@Host2> ActiveContainer -t/ssl:8080;udp:9090;
ActiveContainer <default> - OK
……………
@/Host3> ActiveContainer -t/tcp:8080;ssl:9090;
ActiveContainer <default> - OK
……………
@/Host1> Deployer –h/tcp:8080://Host1/  -p/SimpleOrderManagementApp.jar
……………
```

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

9. Experiments and Results

The experiments were carried out on three hosts networked through 10MBps Ethernet LAN. Host1 was an Acer Quad-Core, Host2 was a Dell Dual-Core, and Host3 was an Acer Single-Core Laptop.

The experiments employed the monitor process, discussed previously which inspected the 'fCreated' field and logged the current time interval at the time of processing. The data from these logs were later manually processed.

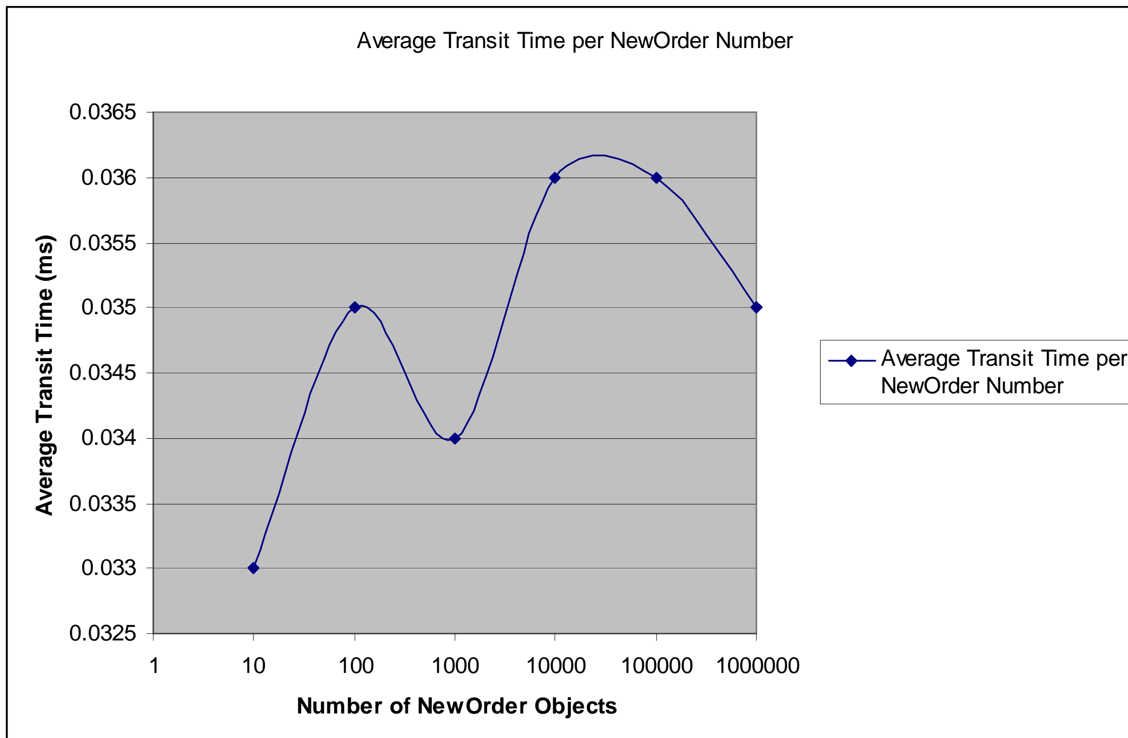
The following experiment scenarios of NewOrder production were used to simulate both ideal load conditions and unpredictable load conditions:

1. Average transit time thru system of NewOrder objects with respect to constant production rate.
2. Average transit time thru system of NewOrder objects with respect to sinusoidal varying production rate.

Results of Experiment 1:

The rate of production of NewOrder objects is assumed to be constant.

NewOrder Objects	Average Transit Time (milliseconds)
10	0.033
100	0.035
1000	0.034
10,000	0.036
100,000	0.036
1,000,000	0.035



From the table and graph above of results for experiment 1, it can be seen that transit times are scalable with respect production of NewOrder objects. The variation of 4.5% between results could be accounted for by considering that processing of services is spread across multiple hosts, which have differing CPU types and speeds, as well as memory availability. As well production rate of NewOrder objects has potential minor variations in rate, due to JVM and operating system influences. This variation can therefore be attributed to a standard deviation of results due to test bed differences.

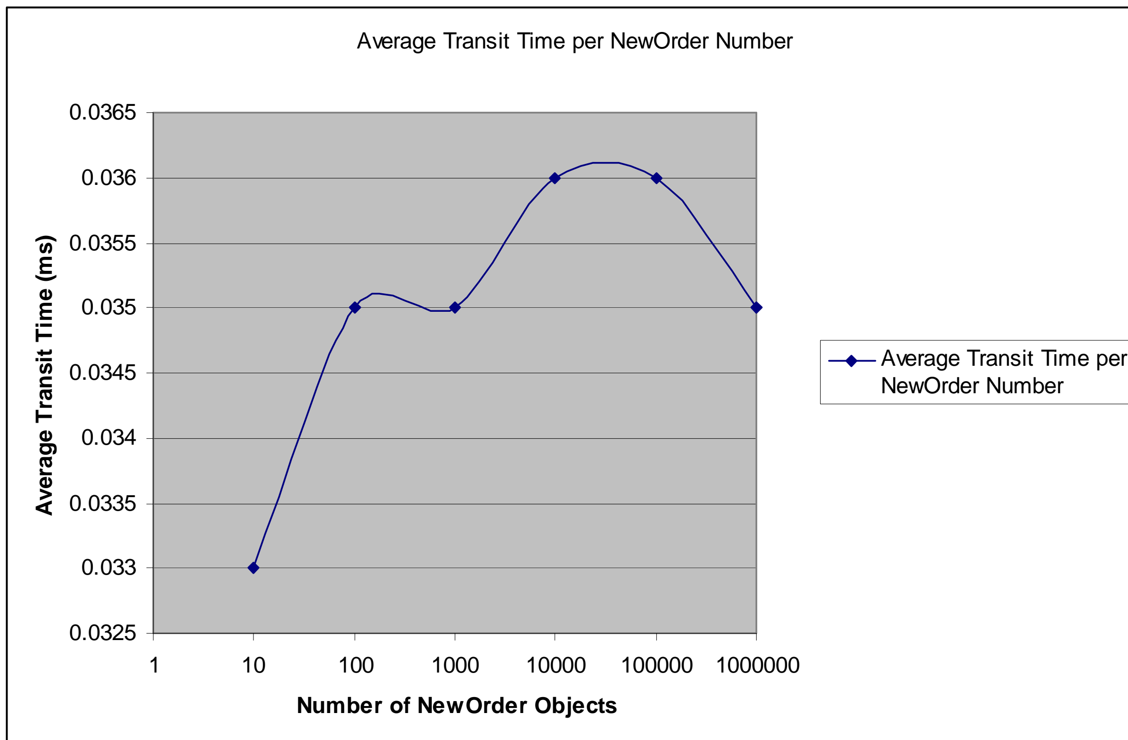
Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

Results of Experiment 2:

The rate of production of NewOrder objects is made to vary sinusoidally with time. This is a relatively simple simulation of variation of traffic through the system in order to test system response to variability.

NewOrder Objects	Average Transit Time (milli-seconds)
10	0.033
100	0.035
1000	0.035
10,000	0.036
100,000	0.036
1,000,000	0.035



From the table and graph above of results for experiment 2, it can be seen that transit times are scalable with respect to production of NewOrder objects. Although it is to be expected following from 'experiment 1' it however demonstrates that the system can stay within normal functioning boundaries even during variation of load which simulates for instance 'Flash Crowds'. Once again the variation of 4.5% between results could be accounted for by considering that processing of services is spread across multiple hosts, which have differing CPU types and speeds, as well as memory availability. As well the variable production rate of NewOrder objects has potential minor variations in rate, due to JVM and operating system influences. This variation can therefore be attributed to a standard deviation of results due to test bed differences.

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.

CLEARFALLS

10.Conclusions

Now we can examine whether our initial goals were met for this tutorial.

1. **Ease of design, construction, testing and deployment,**

   It has been demonstrated that the RealFusion application design process is based upon easy and familiar concepts which leverage the advantages of a plain old Java objects (POJO's) design methodology. As well the usual difficulties testing in container-based systems development have been mitigated through the use of a simple object model which allows for ease of testing using familiar tools such as JUnit at the developer's workstation.

2. **Scalability of application with respect to Resource Utilization and Load.**

   The experiments and their results described in this tutorial, although limited in their extent, firmly demonstrate that a RealFusion based application is scalable with respect to handling unpredictable loads. Although not discussed in the above experimental section it was also evident and determined that all computing resources participated in the order management system application. This is relevant in that other solutions statically assign resources to be used for application processing with the disadvantage of under or over estimating resources, as well having resources sitting idle for long periods. It was demonstrated that the RealFusion based application was able to treat all resources as available and therefore could make full use of them.

3. **Sustainability of Service,**

   In particular Experiment 2 demonstrated service sustainability; in that under variable load the application was able to still maintain ideal processing parameters.

4. **Reliability of applications**

   The fact that the simple order management system was able to leverage all available resources demonstrated two important features of RealFusion:

   1) That services are 'NOT' fixed to one particular resource and such if a computing resource becomes unavailable the other resources can take-over, and

   2) The data objects being separated from the service objects allows for fail-over of processing requests to other services and therefore other computing resources.

5. **Low-Latency by virtue of the ability for RealFusion applications to co-locate service processing**

   This is an architectural aspect of the ACA underlying RealFusion. In that the ability to aggregate multiple differing services (co-locate) within an ActiveContainer allows for sub-millisecond response times. Another aspect which is derived from the ability of services to co-locate within ActiveContainers and therefore contributes towards low-latency is that co-location within RealFusion can move services closer to their eventual point of access and execution. For instance this is a valuable feature where computing networks include slow links also WN links, whereby it is possible to perform processing on stateless services closer to the end of the link where the processing is needed, rather than having to cross slow and possibly unreliable links many times.

6. **Dynamic ReTaskability of hardware infrastructure**

   The ACA is inherently a dynamically reconfigurable middleware. There exists only a loose coupling between services and underlying hardware. This allows for the ability to use all available computing resources at all times, rather than having specialised hardware units which may lie idle for long periods of time.

For further information, please contact:
ClearFalls Pty Ltd
Level 1, 80 Jephson Street
Toowong, QLD 4066
Australia
Phone: (07) 3327 9810
Email: enquiries@clearfalls.com

Authors:
Doug Van Gelder
Martin Graham
Date: 19/05/2009

Copyright © 2009, ClearFalls Pty Ltd. All rights reserved. No part of this publication may be reproduced in any form without the prior written consent of ClearFalls. ClearFalls, RealFusion, RealFusion/CET and Cloud Enablement Technology are trademarks of ClearFalls in the United States and other countries. All other names and logos are the property of their respective owners.