


| | | |
|---|------------------|---|
|  | Location: | N/A |
| | Date: | September 2004 |
| | Responsible: | Luís Ferreira Pires, University of Twente, pires@cs.utwente.nl |
| | Confidentiality: | MODA-TEL Consortium confidential |
| | Distribution: | MODA-TEL Consortium |
| MODA-TEL IST-2001-37785 | Document Ref.: | /private/D7.3 |

MODA-TEL

D7.3: Innovation prototype demonstrator

Editor: Luís Ferreira Pires, University of Twente

Suggested readers

MODA-TEL consortium partners

European Commission Project Officer

Abstract

This document reports on a prototype built to demonstrate the concept of abstract platform and show how it can be supported with modelling and transformation tools, and middleware technology.

Disclaimer

This document contains material, which is copyright of certain MODA-TEL consortium parties and may not be reproduced or copied without permission. The information contained in this document is the proprietary confidential information of certain MODA-TEL consortium parties and may not be disclosed except in accordance with Section 12 of the consortium agreement.

The commercial use of any information in this document may require a licence from the proprietor of that information.

Neither the MODA-TEL consortium as a whole, nor a certain party of the MODA-TEL consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using the information.

List of Authors

| Name | Company | e-mail |
|----------------------------|----------------------|--|
| Luís Ferreira Pires | University of Twente | pires@cs.utwente.nl |
| João Paulo Andrade Almeida | University of Twente | alme@cs.utwente.nl |

Table of Contents

| | |
|--------------------------------|----|
| List of Authors..... | 3 |
| List of Figures..... | 4 |
| 1 Introduction | 5 |
| 2 Overview | 5 |
| 3 Preparation phase..... | 6 |
| 4 Execution phase..... | 9 |
| 5 Deployment | 9 |
| 6 Running the application..... | 10 |
| 7 Conclusions | 11 |
| References | 12 |

List of Figures

| | |
|---|----|
| Figure 1 Preparation phase steps demonstrated by the prototype | 5 |
| Figure 2 Execution phase steps demonstrated | 5 |
| Figure 3 PIM metamodel for the prototype | 6 |
| Figure 4 Simple Server example..... | 6 |
| Figure 5 Snapshot of UML profile project | 8 |
| Figure 6 Reconfiguration module and its commands | 8 |
| Figure 7 Using the Generate command to perform a transformation | 9 |
| Figure 8 Prototype demo configuration | 10 |
| Figure 9 Location manager user interface | 10 |
| Figure 10 User interface of the client | 11 |

1 Introduction

MODA-TEL WP7 has produced a collection of papers on the MDA concepts and challenges. In particular the concept of abstract platform has been introduced in [1] as a means to help developers setting up the PIM metamodel, the transformations and the underlying platform. In [2] we illustrate the concept of abstract platform, by discussing how CORBA extended with the DRS (Dynamic Reconfigurable Service) can be used as a target concrete platform that allows designers to ignore reconfiguration while producing the PIMs, yet obtaining a final realisation that supports reconfiguration. This document reports on a prototype built to demonstrate the claims we have made in [2], namely that by using the concept of abstract platform one can ignore some implementation issues at the PIM level and concentrate on the application logic, leaving these implementation issues to be handled by (automated) transformations.

This document is further structured as follows: section 2 gives an overview of the development steps demonstrated with our prototype, section 3 discusses the preparation activities, section 4 discusses the execution activities, section 5 discusses deployment issues, section 6 discusses what can be shown when running the prototype demonstrator and section 7 gives some final conclusions.

2 Overview

The prototype demonstrator presented in this document shows an abstract platform that is implemented by a middleware platform with reconfiguration mechanisms. In this way a designer can reason about the application logic and design the application objects and their interactions at the PIM level without having to consider the reconfiguration functionality that may be necessary to fulfil some availability requirements. A transformation has been defined and implemented using Objecteering, which allows one to obtain the implementation of factories for the objects (classes) defined at the PIM level automatically. The designer has to define the body of the operations of each class by hand (in Java), and the transformation packs all the generated code in an executable `jar` file ready for deployment. The prototype demonstrator also shows how deployment can be started from Objecteering.

Figure 1 shows the steps of the preparation phase demonstrated by this prototype.

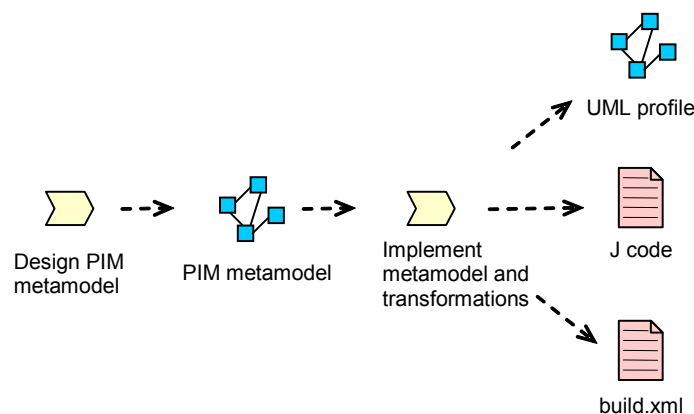


Figure 1 Preparation phase steps demonstrated by the prototype

Figure 2 shows the steps of the execution phase demonstrated by the prototype.

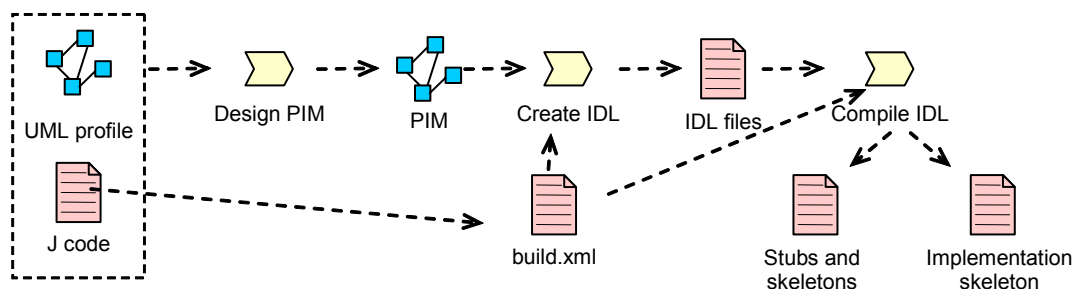


Figure 2 Execution phase steps demonstrated

The UML profile and the J code necessary to perform the transformation from PIM to IDL (CORBA PSM) are managed by Objecteering. The ant script `build.xml` defines the tasks that should be performed in order to compile the IDL files, compile the resulting Java code and packing the resulting classes in a deployable (`jar`) file.

The preparation phase is discussed in more detail in section 3. The execution phase is discussed in more detail in section 4, while deployment issues are discussed in more detail in section 5.

3 Preparation phase

The preparation phase consists of defining the PIM metamodel and the transformations, and implementing them. For our prototype demonstrator we have implemented the PIM metamodel and the transformations as an UML profile using the Objecteering UML profile modeller tool.

Figure 3 shows the PIM metamodel defined for this prototype.

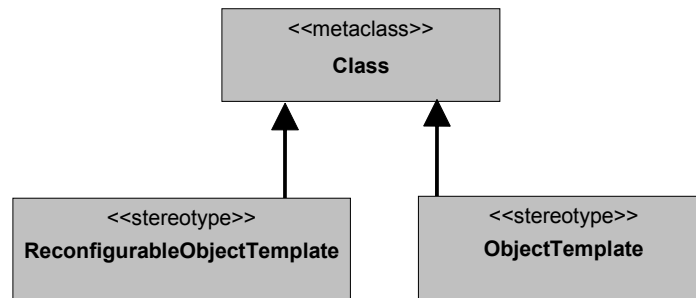


Figure 3 PIM metamodel for the prototype

This PIM metamodel makes it possible for a developer to define a class of objects as reconfigurable or not, by simply assigning the stereotype `ReconfigurableObjectTemplate` or `ObjectTemplate` to this class, respectively.

We define a transformation from PIMs to IDL, which roughly corresponds to a CORBA PSM if we take into account the UML profile for CORBA [4]. This transformation has the following rules:

1. Each class of the PIM is transformed into a single IDL file defining an interface with the same name as the class;
2. IDL interfaces of classes with stereotype `ReconfigurableObjectTemplate` are defined as extensions of the `CosDRS::ReconfigurableObject` interface
3. Interfaces of classes with stereotype `ObjectTemplate` are defined without extending any particular interface;
4. In the IDL file of a class with stereotype `ReconfigurableObjectTemplate` a factory interface is defined, which extends the `CosDRS::ReconfigurableObjectFactory` interface; no factory interface is generated for classes with stereotype `ObjectTemplate`;
5. Each operation of the class is transformed into an operation of the interface of this class.

In this prototype, attributes are ignored for the sake of simplicity.

Figure 4 shows a class `Server` that implements operations `hello()` and `stats()`, which return a string and the number of times the server has been invoked so far, respectively.

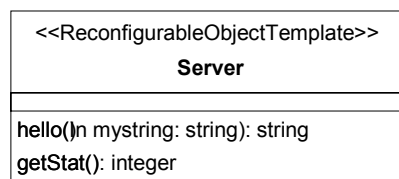


Figure 4 Simple Server example

The IDL file generated for this class is given below.

```
#include <OB/CosNaming.idl>
#include <org/omg/CosDRS/CosDRS.idl>
```

```
#pragma prefix "Module_Server"
interface Server : CosDRS::ReconfigurableObject
{
    string hello(in string mystring);
    long getStat();
};
interface ServerFactory : CosDRS::ReconfigurableObjectFactory
{
};
```

Since the `Server` class has stereotype `ReconfigurableObjectTemplate`, the transformation defines the IDL interface for this class as an extension of `CosDRS::ReconfigurableObject`, and defines a `ServerFactory` interface to manage instances of this class during runtime. This factory has a role similar to the role of home interfaces in component-based middleware such as CCM and EJB, namely to manage instances of its class. Reconfiguration mechanisms can be associated with a class during deployment exactly because we create a factory for this class and define its interface as extension of `ReconfigurableObject`. These are namely the requirements imposed by the DRS to support the reconfiguration of objects.

The PIM metamodel and the transformation have been implemented using Objectteering in an UML profile called *DRS UML profile*. This profile has been created as a child of the `#default#external#Code` profile, which is an Objectteering built-in profile for code generation.

Figure 5 shows a snapshot of the UML profile for this PIM metamodel as defined in the Objectteering UML Profile Builder.

In order to execute the transformation defined above, we have defined (meta-)operations called `generate()` for the IDL product, `Class` and `Operation` metaclasses, and we have defined a `Generate` command that calls the IDL product `generate()` operation. The IDL product `generate()` operation calls the `Class generate()`, which starts creating the IDL file, and calls the `Operation generate()` operation to fill in the operations in the file accordingly, whenever an operation is found in the class being transformed. After the IDL file is ready, the `ant` facility is called to execute a `build.xml` script that compiles the IDL to generate stubs and skeletons. In case no implementation file exists (file named `<interface>_impl.java`) a skeleton of this file is generated. This skeleton has the form of a class with empty operation bodies to be filled in; the `editCode()` (meta-)operation of the `Class` metaclass allows the designer to edit this implementation file. The `build.xml` script also compiles all the Java files generated and creates a `.jar` file ready for deployment.

We have also defined a `deploy()` (meta-)operation in the `Class` metaclass that allows the factories of the classes to be started directly from the model. This (meta-)operation is useful for debugging and demos, but we doubt whether it would be useful in production.

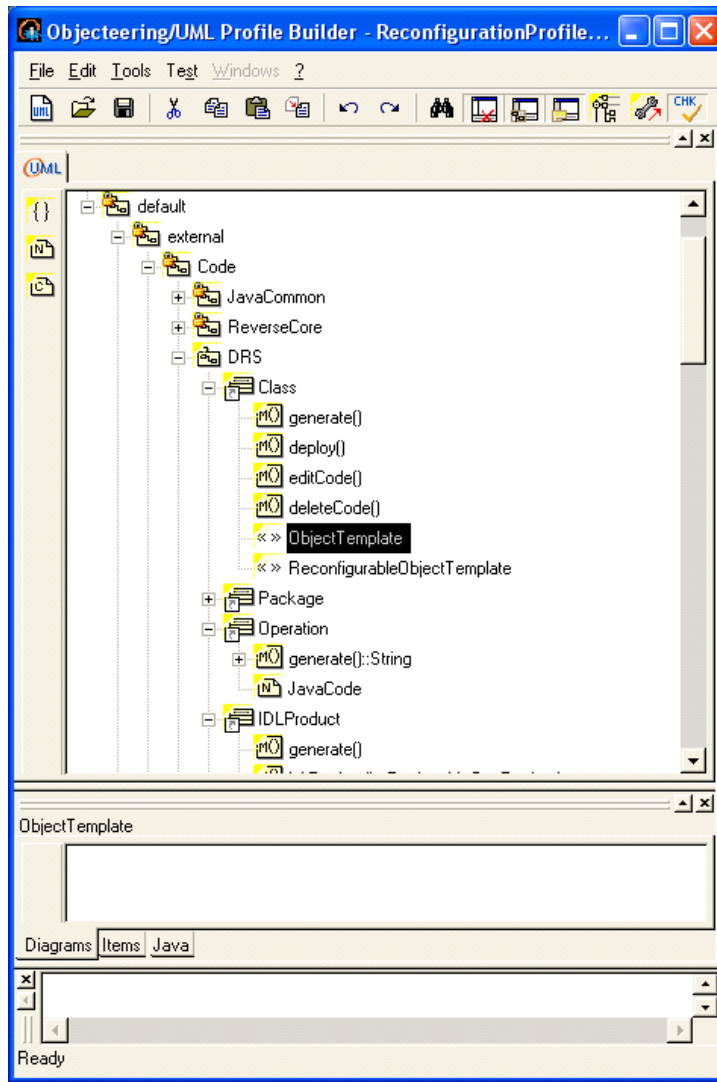


Figure 5 Snapshot of UML profile project

Figure 6 shows the module that supports the DRS UML profile and the commands defined for this module.

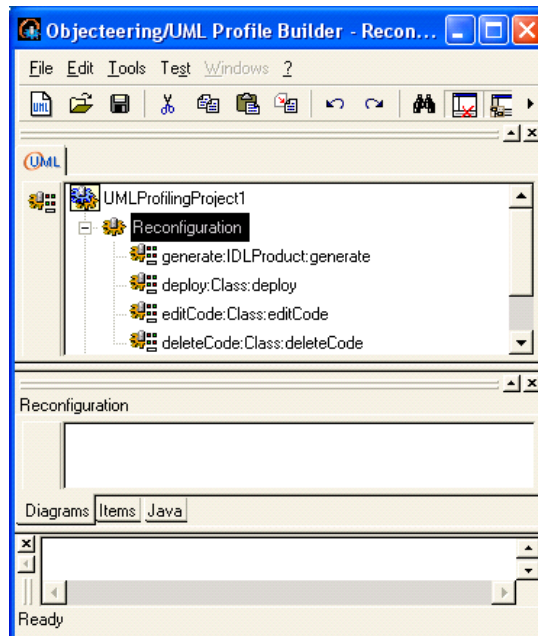


Figure 6 Reconfiguration module and its commands

4 Execution phase

During the execution phase the designer creates a project and associates this project to a module that supports the UML profile. By doing this, the commands defined in the module can be called on instances of the metaclasses. In our demo, we associate a project to the Reconfiguration module so that the commands Generate, deploy, editCode and deleteCode can be called on instances of the IDL product and the Class metaclass accordingly.

In order to create the IDL file we have to create an IDL product for the class we want to transform. By calling the Generate command on this IDL product we can generate the IDL file and call the `build.xml` script as described before.

Figure 7 shows a snapshot of the Objectteering UML modeller for a simple UML project.

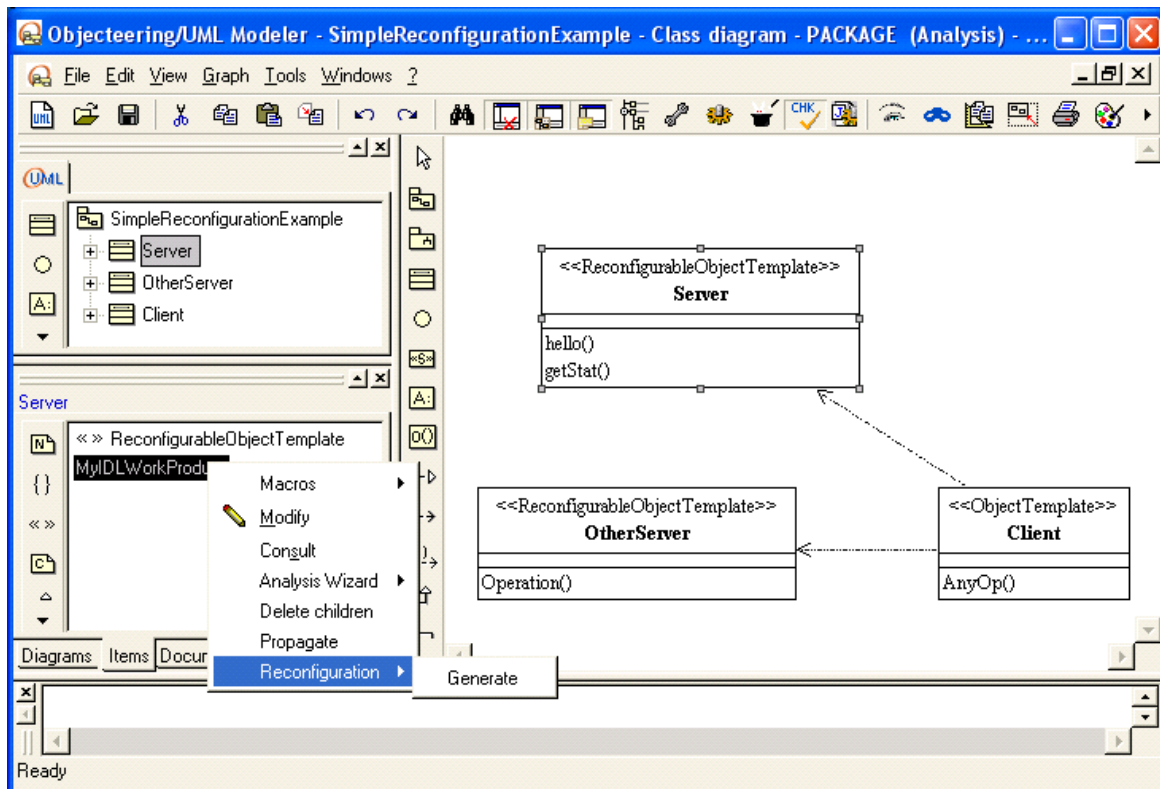


Figure 7 Using the Generate command to perform a transformation

Figure 7 shows that the Reconfiguration -> Generate command can be applied on the MyIDLProduct instance of the IDL Product metaclass. MyIDLProduct has been assigned by the designer to the Server class. After the Generate command is executed, an IDL file is generated and compiled, and the generated Java files are compiled and packed in a `.jar` file ready for deployment.

5 Deployment

After deployment the prototype shows that the instances of the classes assigned as reconfigurable can indeed be reconfigured. This has been shown by starting the Reconfiguration Manager server, which is responsible for coordinating the reconfiguration step, and creating two factories at two different physical locations (two different computers). The Reconfiguration Manager depends on a Location Agent server to perform these tasks. A Location Manager object acts as a client for the reconfiguration manager, offering a convenient user interface to request and visualise reconfiguration steps. The prototype only supports the migration of a single object at a time, for the sake of simplicity.

Figure 8 shows the configuration of the prototype demonstration.

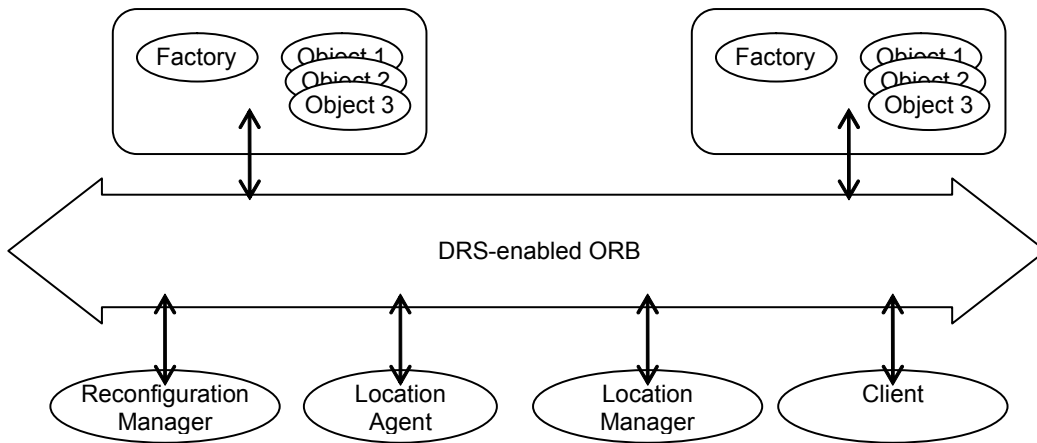


Figure 8 Prototype demo configuration

In this prototype demo, we deploy a factory for a generated server at two different locations directly from the Objecteering UML Modeller. Deploying code on the machine in which Objecteering is running is trivial, i.e., by running the generated `.jar` file. In order to deploy the code on a remote machine we have written a `build.xml` script that waits for a `run.properties` file to be written at a certain directory and runs the code after this file becomes available. The `run.properties` file contains the name of the `jar` file to be executed and the arguments of the program. The `build.xml` script triggered by Objecteering writes the `jar` file, a file with the IOR of the Reconfiguration Manager and the `run.properties` file to both the local and remote locations. This requires a disk accessible for the remote machine to be mounted on the machine running Objecteering. In this way we have been able to start a factory for an arbitrary class in two locations (almost) simultaneously.

6 Running the application

After starting the Location Agent, the Reconfiguration Manager and deploying the factories, we start the Location Manager and a simple client. This simple client requests an object to be created and starts executing `hello()` and `stats()` on the server continuously. While the client is executing, we can use the Location Manager interface to migrate an object to the other location. After migration, we notice that the `stats()` results are reset, since we have not used the persistence facilities of DRS, actually on purpose to show that another non-persistent instance of the object is created after migration.

Figure 9 shows the user interface of the location manager.

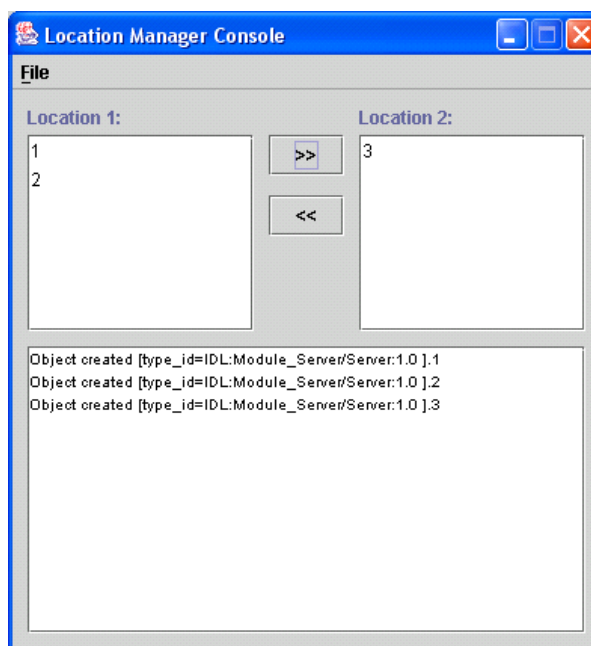


Figure 9 Location manager user interface

By selecting an object in the interface shown in Figure 9 and choosing << or >>, we can migrate an object to Location 1 or Location 2, respectively. In this prototype demo, these locations correspond to the two factories that are deployed directly from Objecteering.

Figure 10 shows the user interface of the simple client. This user interface allows its users to start or stop a sequence of hello() and stats(), or to perform a single hello() and stats() step. The interface also shows the response time of the invocations, which were originally meant to get an indication of the additional delays caused by the reconfiguration mechanisms.

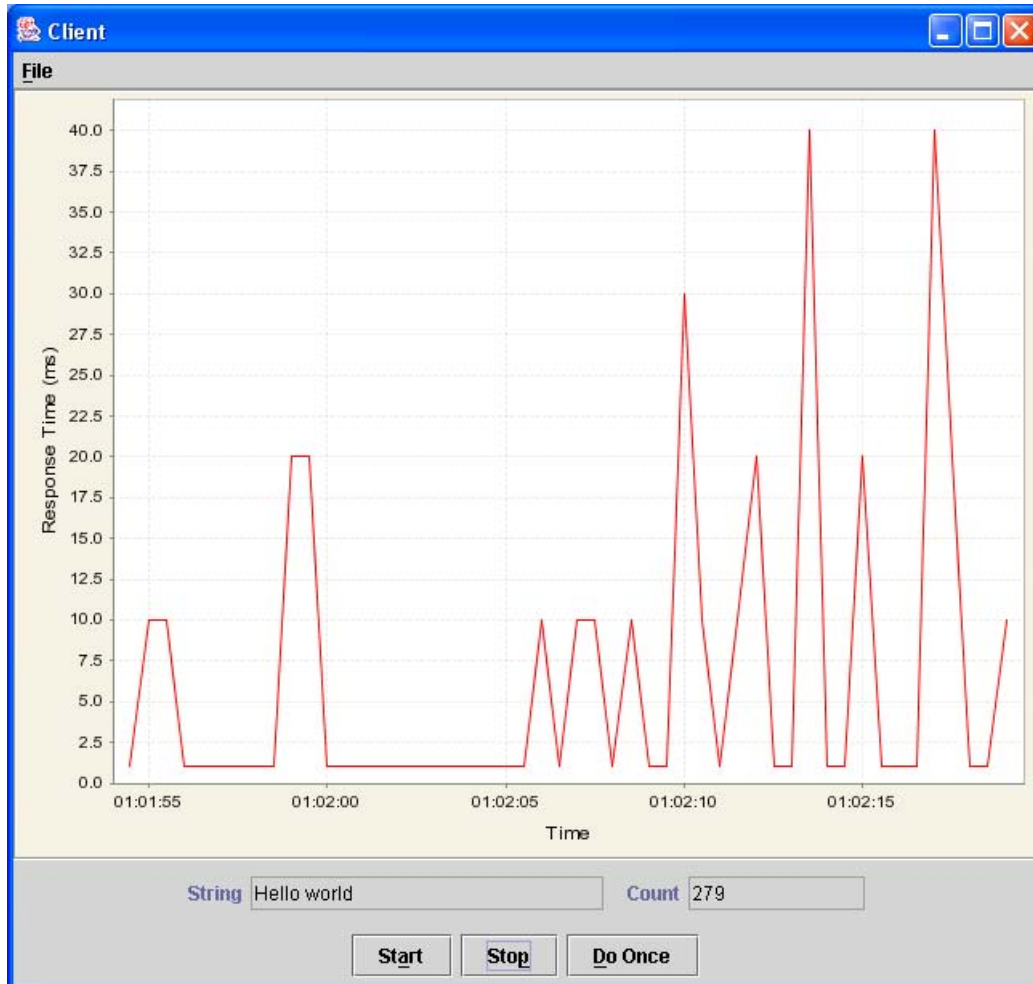


Figure 10 User interface of the client

7 Conclusions

This prototype demonstrator shows that by applying the concept of abstract platform one can reason about the application logic at the PIM level without having to be bothered by issues like reconfiguration. The demonstrator also shows that transformations can be defined and implemented in order to cope with these issues (almost) automatically.

In [1] we state that an abstract platform is defined in terms of a collection of concepts and is supported by a collection of components. In our prototype, the concepts are the stereotypes defined to represent reconfigurable and non-reconfigurable objects, while the components are the Reconfiguration Manager (with the Location Agent) and the DRS classes that have to be attached to the reconfigurable objects for reconfiguration to be supported.

References

- [1] J.P.A. Almeida, R. Dijkman, M. van Sinderen and L. Ferreira Pires. On the Notion of Abstract Platform in MDA Development. In: *Proceedings Eighth IEEE International Conference on Enterprise Distributed Object Computing* (EDOC 2004), California, USA, Sept. 2004, to appear.
- [2] J.P.A. Almeida, M. van Sinderen, L. Ferreira Pires and M. Wegdam. Platform-independent Dynamic Reconfiguration of Distributed Applications. In: *Proceedings IEEE 10th International Workshop on Future Trends in Distributed Computing Systems* (FTDCS 2004), Suzhou, China, May 26-28, 2004, pp. 286-291.
- [3] J.P.A. Almeida, M. van Sinderen, L. Ferreira Pires and M. Wegdam. Handling QoS in MDA: a discussion on availability and dynamic reconfiguration. In: *Proceedings of the Workshop on Model Driven Architecture: Foundations and Application* (MDAFA) 2003, CTIT Technical Report TR-CTIT-03-27, University of Twente, the Netherlands, June 26-27, 2003, pp. 91-96.
- [4] OMG. *UML Profile for CORBA Specification*. Version 1.0 (formal/02-04-01). April 2002.