INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Projektarbeit

Development of a distributed architecture for controlling and monitoring REFLECT systems

Gernot Pointner

Aufgabensteller: Prof.Dr. Martin Wirsing
Betreuer: Christian Kroiss
Abgabetermin: 16. Dezember 2009

Hiermit versichere ich, dass ich die vorliegende Praktikumsarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 16. Dezember 2009

.....

(Unterschrift des Kandidaten)

Abstract

The REFLECT framework is used to create adaptive multimedia applications and is based on the OSGi framework. This thesis gives some information about customizing the existing REFLECT core in order to being able to access it remotely and the development of a GUI for remote monitoring and controlling a REFLECT application.

Contents

List of Figures

CHAPTER 1

Introduction

1. The REFLECT Framework

The REFLECT Framework is a Java framework based on OSGi (see 3.1) which can be used to create adaptive multimedia applications. It is developed at the "Lehrstuhl fuer Programmierung und Softwaretechnik" at the LMU, Munich. The framework provides the infrastructure for Component-Based Software Engineering. This approach offers several benefits, e.g. the possibility to reuse components in several software projects, being able to restrict communication between different components to those, which have explicitly been wired and the possibility to realize information hiding for components(communication between different components only takes place using declared interfaces).
There some parts of the framework that are worth further explanation: Components are essential parts of a REFLECT application. Each component has a well-defined function. There are three types of components:

- Passive Components: Offer functionality for other components (e.g. buffering and aggregating data, providing library functions...).
- Active Components: Running as a thread, continuously performing actions(e.g. server components waiting for requests ...)
- Composite Components: Can consist of several components of an arbitrary type (also Composite Components) in order to provide functionality using several components and introduce abstraction from the underlying architecture.

Components can have so-called "Properties". Those Properties are fields of the class determing the state of the Component. These Properties can either be modifiable (if a setter for this property is present) or not-modifiable (if only a getter is present), which is automatically determined by the framework. If a property of a component has changed, interested listeners get notified about the change.
Components can also have so-called Ports. Ports can either be declared as "Provided" or "Required" and also their connection multiplicity (n:m) can be given. Component A which offers a provided Port c can then be connected to Component B which requires Port c using so-called Connectors.

2. Motivation for the REFLECT Remote Control Center

The REFLECT Remote Control Center offers the possibility to monitor the properties of components of REFLECT application via remote access while it is running. Additionally it's also possible to change modifiable properties at runtime which is a big advantage over continuously changing hardcoded values within the source and restarting the application or using XML-files containing the property values. Especially if you want to experiment with different values, the ability to change values at runtime can save a huge amount of time.

3. Used Technologies

3.1. OSGi. The OSGi standard describes a dynamic module system for Java. It provides standardized primitives that allow applications to be constructed from small, reusable and collaborative modules. [**All09**] It follows the philosophy of a "SOA" , a service-oriented architecture. Related program logic can be encapsulated within so-called bundles and registered as a service, which itself can be accessed then by other services or bundles.
OSGi aims at reducing the complexity of software systems by using components which have a well-defined scope. For the REFLECT framework, Equinox is used as the implementation of the OSGi standard.

3.2. R-OSGi. R-OSGi is developed by the Swiss Federal Insitute of Technology Zurich. It's an OSGi-Bundle which allows transparent access to running OSGi-Bundles from remote machines and facilitates distribution for arbitrary OSGi framework implementations [**Rel09**] As the REFLECT framework bases on OSGi, this technology is used to establish the communication between the REFLECT Remote Control Center and the REFLECT application.

3.3. RCP. The Rich Client Platform can be used to build clients upon. It uses the same technology on which the Eclipse IDE is based on. The resulting rich applications are still based on a dynamic plug-in model, and the UI is built using the same toolkits and extension points. [**eW09**]. SWT [**eli09**] is used for building the GUI components on order to provide a native look and feel for all available platforms as SWT uses native widgets.
The Remote Control Center GUI was built upon RCP as a so-called "Fat Client".

4. An example usecase for the Remote Control Center

The REFLECT framework is used for an application, which receives data from a driving simulator, such as acceleration , velocity or angle of the steering wheel and tries to compute some information about the driver's current mental state and driving style. For this reason, the application consists of several components, each performing a defined task (receiving data from the simulator ,aggregating and buffering data ,performing mathematical operations on the data,sending response to the simulator...) . Especially those components, which are responsible for performing computations on the data and drawing conclusions from it have to be tested intensively in order to return the designated results, as this isn't a trivial task. For this reason, the settings, which determine the behavior of these components are made available as modifiable properties. Then the components can be fine tuned at runtime by accessing the application using the Remote Control Center and effects of the changes are experienced immediately. This approach saves a lot of time as the alternative way to achieve this would be to change the values directly in the source code and restarting the application or changing the predefined values within XML-files which have to be read in again. As it's possible that the properties have to be changed very often until the designated value has been found, it's likely that changing the values of the properties using the Control Center saves a lot of time towards changing values in XML-files and having to re-read them back in or even changing the values directly in the source code.

CHAPTER 2

Usage of the Remote Control Center

1. Basic overview of the GUI

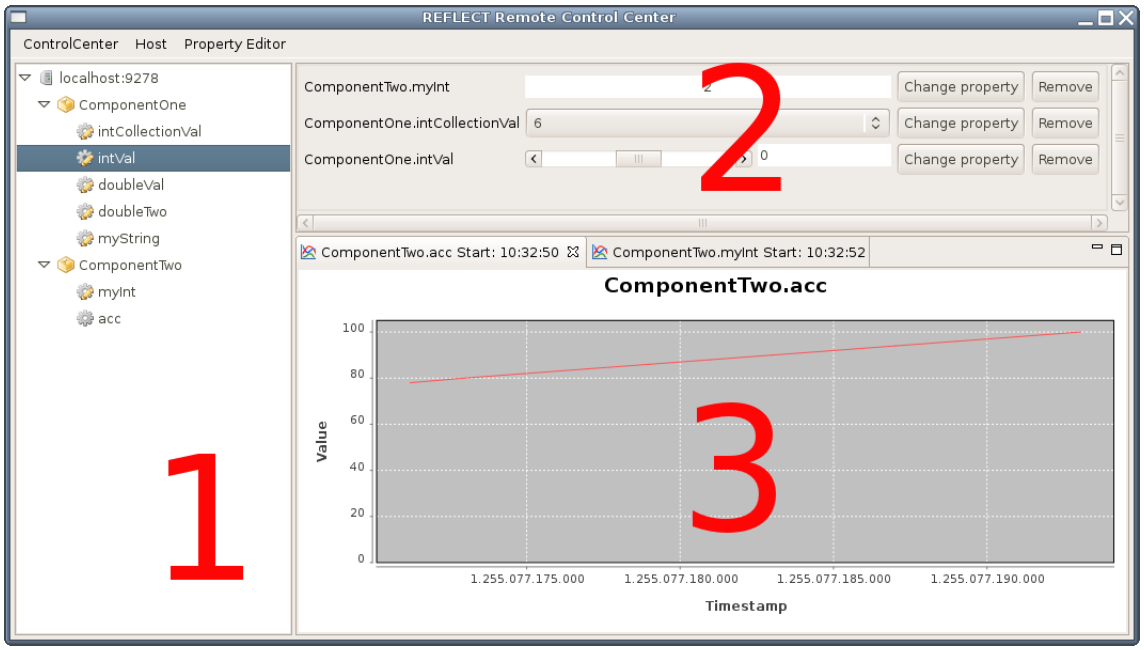


FIGURE 2.1. The Remote Control Center GUI

2.1 shows the GUI of the Remote Control Center. The Component Browser (**1**) can be used to navigate through the available components and properties. Within the browser you can select specific properties and choose to edit them (if they are modifiable) or to monitor them (if they are of a type that is monitorable.) The Property Editor (**2**) can be used to edit properties within the ranges of their associated Property Domain. Property Domains can be assigned to a property in order to constraint the range of values that can be set for this property. Finally the Property Monitor (**3**) offers the possibility to monitor properties of certain types (currently *Integer* and *Double* are supported) and save the monitored data afterwards.

2. Use cases

2.2 shows a basic overview of the possible Use Cases for the Control Center.

2.1. Connect to a Reflect application. Select the "Connect" menu item from the "Host" menu, enter the designated host and port and press "Connect". An alternative way would be to right-click on the "Not connected" - Node within the Component Browser.

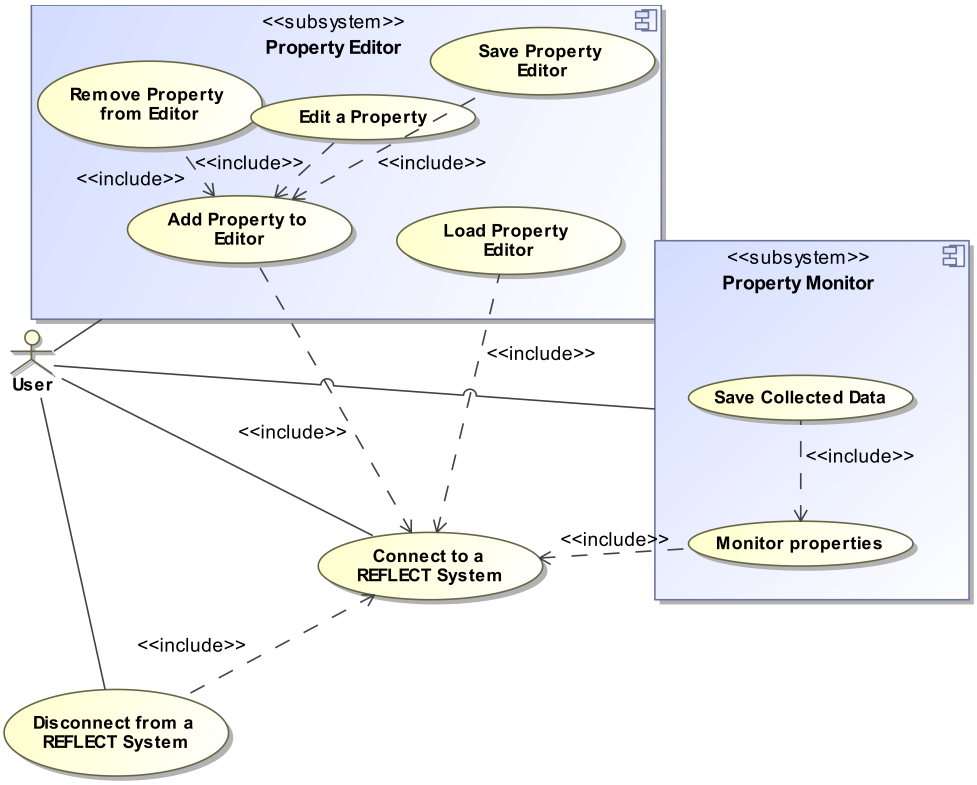


FIGURE 2.2. Use case overview

2.2. Add a Property to the Property Editor. After you have connected to a Reflect application, right-click on the designated modifiable property within the Component Browser (modifiable properties are marked by a pencil) and choose the "Edit this Property" item.

2.3. Edit a Property. Properties can be edited within the Property Editor by using appropiate input methods (sliders, combo boxes, textfields. . .) .

2.4. Remove a property from the property Editor. Previously added Properties can also be removed again from the Editor.

2.5. Save a Property Editor Configuration. Select the "Save Property Editor" option from the "Property Editor" menu.

2.6. Load a Property Editor Configuration. After you have connected to a Reflect application, select the "Load Property Editor" option from the "Property Editor" menu. Properties that are not available in the Reflect application, to which you are currently connected to, won't be loaded.

2.7. Monitor a property. After you have connected to a Reflect application, right-click on the designated property within the Component Browser and choose the "Monitor this property" option. A new Property Monitor view will be opened.

2.8. Save collected data. You are asked to save collected Data if you either close the corresponding property monitor or or disconnect from the Reflect Application.

2.9. Disconnect. Select the “Disconnect” option from the host menu.

CHAPTER 3

Architecture of the Remote Control Center

1. General Overview

1.1. Description of the Java Packages. The Remote Control Center is based on Eclipse RCP technology and subdivided into several packages. 3.1 shows the most important ones. In the following a more or less detailed description of the different parts is given.

- `de.lmu.ifi.pst.reflect.remote.control.center`
  Contains the Activator class and RCP specific classes for creating the workbench and menu.
- `de.lmu.ifi.pst.reflect.remote.control.center.actions`
  Contains actions that can be triggered by the users by using the Graphical User Interface.
- `de.lmu.ifi.pst.Reflect.remote.control.center.component.browser`
  Logic and rendering of the component browser
- `de.lmu.ifi.pst.reflect.remote.control.center.controller`
  GUI and Remote Controller
- `de.lmu.ifi.pst.reflect.remote.control.center.dialogs` Non-standard dialogs can be found here. At the moment only the dialog for connecting to a Reflect application exists.
- `de.lmu.ifi.pst.reflect.remote.control.center.logging`
  Logging functionality
- `de.lmu.ifi.pst.reflect.remote.control.center.property.editor`
  Logic and rendering of the property editor plus loading and saving functionality.
- `de.lmu.ifi.pst.reflect.remote.control.center.property.monitor`
  Functionality for drawing plots of monitored components and collecting and saving data.
- `de.lmu.ifi.pst.reflect.remote.control.center.remote.listener`
  Management of client side property listeners and establishing an answer channel for the Control Center.
- `de.lmu.ifi.pst.reflect.remote.control.center.splashHandlers`
  Display of the splash screen, was created during the branding process.
- `de.lmu.ifi.pst.reflect.remote.control.center.views`
  Views for Component Browser, Property Editor and Monitor

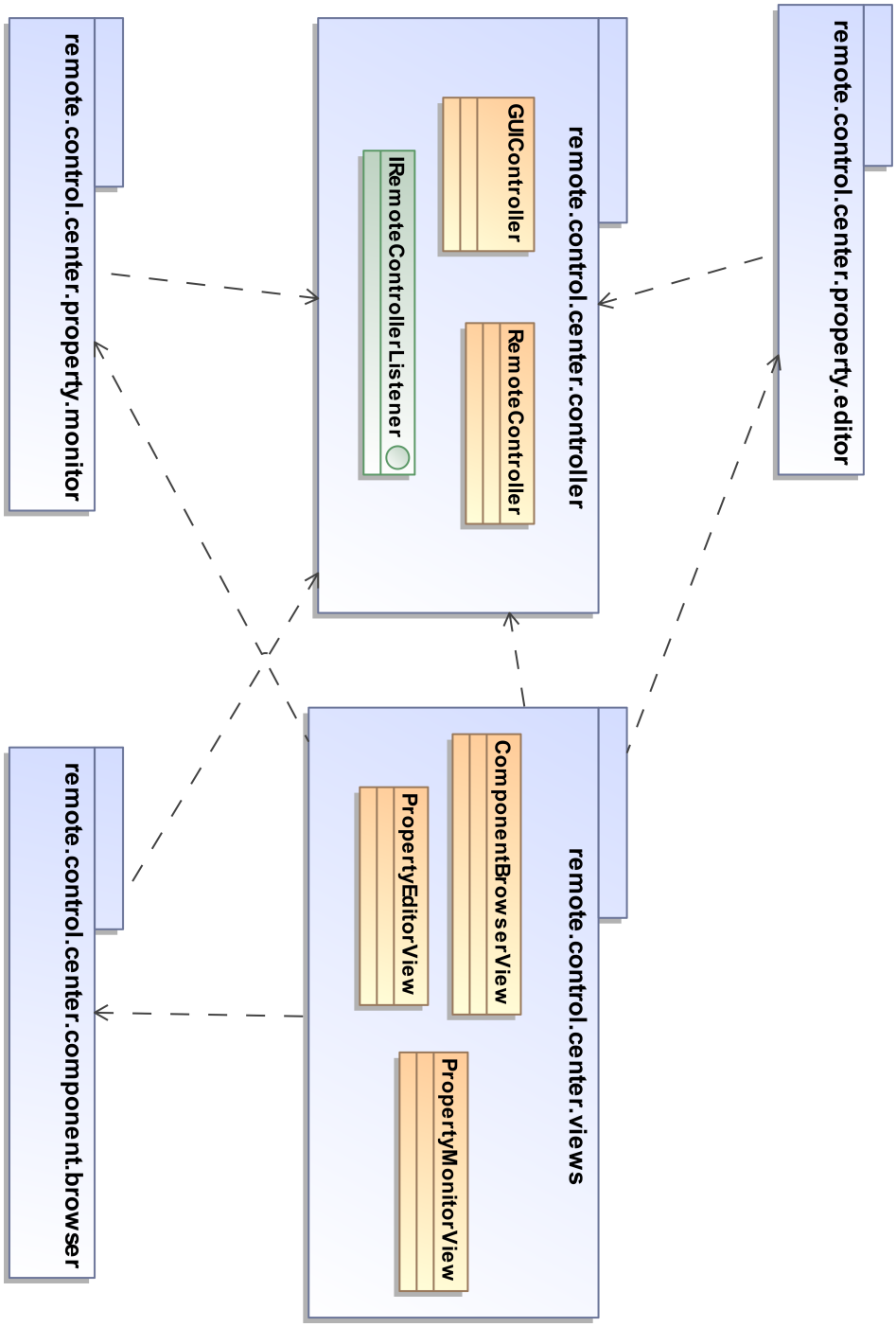


FIGURE 3.1. Overview of important packages

1.2. Overview of the Software modules. 3.2 gives an overview the components of which the remote control center exists: The connection to the running REFLECT application, which should be accessed remotely, is handled by the Remote Controller. Component Browser, Property Editor and Property Monitor show

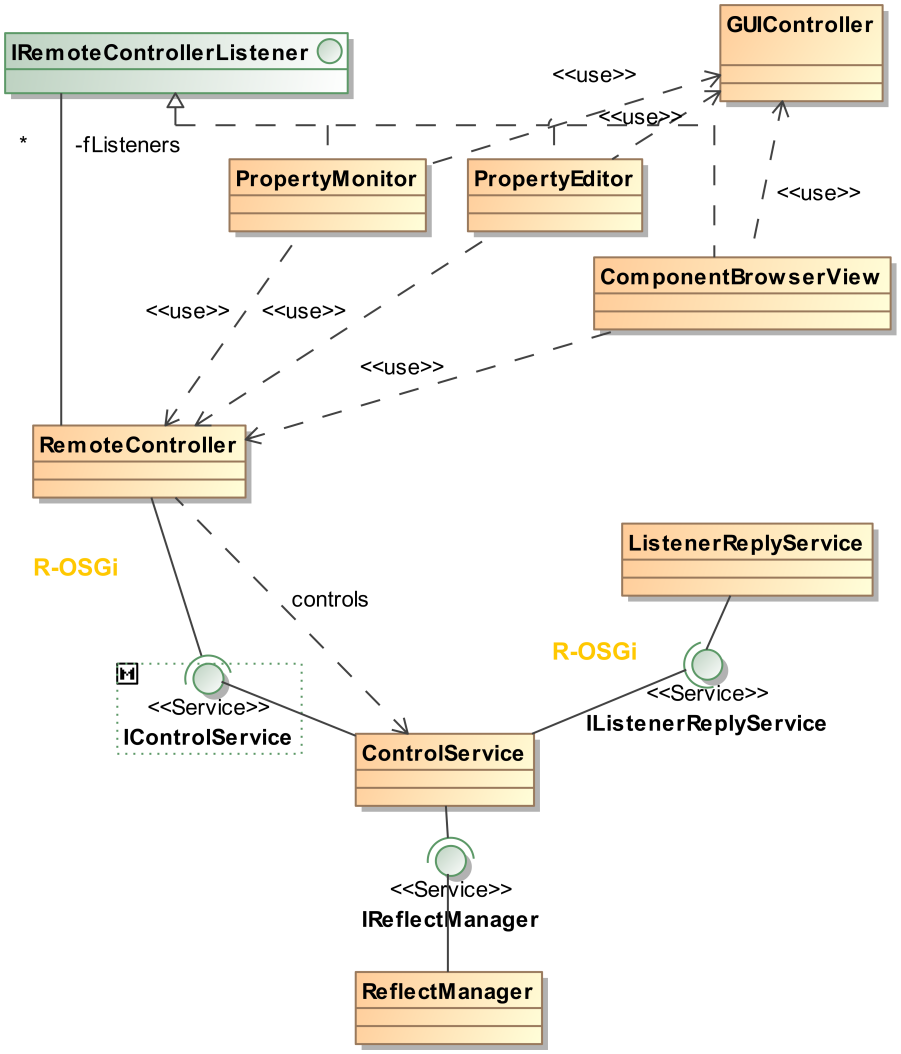


FIGURE 3.2. Overview of software modules

a different behavior and appearance, depending on if a connection has been established or not. The Component Browser e.g. presents available components and properties, if the user has established a connection and a simple "Not connected" message otherwise. Therefore the mentioned components get notified about changes of the connection status. This is achieved by using the Event Listener/Observer Pattern. Parts like the Component Browser or Monitor are registering themselves as listener at the Remote Controller and from then on receive messages whenever the connection status changes. For this reason they implement the "IRemoteControllerListener" interface. The GUIController can globally be accessed, manages interactions between the different parts of the graphical user interface and offers methods for changing the state of different parts, which are often used by actions invoked by the user.
The Component Browser can be used to navigate through the components and properties of a running REFLECT application and select properties to be monitored or edited.

The Property Editor offers the possibility to edit Modifiable Properties of running components.
The Property Monitor can be used to plot data of selected Properties and save the collected data into an XML file afterwards.

2. Detailed description of the software components

2.1. Controllers. Both the Remote Controller and the GUI Controller have to be accessed by many classes and components of the software. Both controllers are assumed to behave exactly the same way for every class which uses it. Because of the previously mentioned reasons the Singleton Pattern was chosen for the implementation for both the Remote and GUI Controller.
This assures that only one instance of both controllers can be accessed by other classes. The initialization is not lazy but takes place within the Activator class of the application, as both Controllers have to be initialized anyhow.

2.1.1. *Remote Controller.* The Remote Controller is a essential part of the Remote Control Center. At first it offers functions for safely connecting to and disconnecting from a REFLECT application using R-OSGi. On the other hand it offers several functions for communicating with and retrieving information from the REFLECT application, to which the Control Center is currently connected to. When a connection is established, the Remote Controller notifies all interested views(e.g. Component Browser) about the established connection. When a connection error has occurred or the user has disconnected from the REFLECT application, a method is invoked in an analogous way.

2.1.2. *GUIController.* The GUI Controller also plays an important role within the Remote Control Center. It offers methods for getting input from the user, e.g. prompting him which file to load, and for interaction between different views.
For this reason the different views (Property Editor, ComponentBrowser ...) register themselves when starting up at the GUI Controller. This has several benefits: On the one hand , coupling is reduced, as no view has to be connected to every other view but only has to be registered once at the GUI Controller. On the other hand, convenience methods for adding properties to the Property Editor or Monitor without the need to use the RCP view registry. Finally the GUI Controller offers methods which are useful for every view, e.g. for refreshing and repainting it.

2.2. Component Browser. The Component Browser can be used to navigate through available components and properties and edit properties to be monitored or edited ,if they are modifiable. It listens on the Remote Controller in order to change it's outline corresponding to the actual connection status.

2.3. Property Editor. The Property Editor can be used to edit modifiable Properties. It's initialized when the Control Center starts and resides within the Property Editor View. The Property Editor can be saved to and restored from an previously saved XML file, so that composition of modifiable properties can be kept. The created XML looks like the following:

```
<PropertyEditor>
    <PropertyEntry>
        <ComponentName>
            ComponentOne
        </ComponentName>
       <PropertyName>
           ExampleProp
       </PropertyName>
    </PropertyEntry>
```

```
    <PropertyEntry>
        <ComponentName>
            ComponentTwo
        </ComponentName>
       <PropertyName>
           ExamplePropTwo
       </PropertyName>
   </PropertyEntry>
</PropertyEditor>
```

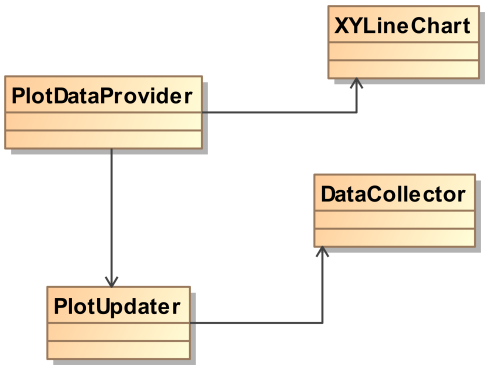


FIGURE 3.3. Property Monitor class diagram

2.4. Property Monitor. The Property Monitor is instantiated within the Property Monitor View. Whenever the user wants to monitor a certain property, a new instance of the view is created. The plotting of the view is done using the JFreeChart library. [**Lim09**]. 3.3 shows the basic architecture of the Property Monitor. The PlotUpdater checks for new data in regular intervals and updates the monitoring graph (represented by the class XYLineChart). The Property Monitor also makes use of a class called DataCollector. Its purpose is to fetch actual data from the remote application , store it and also to provide the PlotUpdater Thread with the latest data. Finally when the user closes a Property Monitor or the connection is lost, the user is prompted to save the data collected by the Property Monitor to an XML file. The created XML looks like the following:

```
<CollectedData>
    <PropertyName>
        MonitoredProperty
    </PropertyName>
    <TimeStamp>
        1234567
    </Timestamp>
    <Value>
        12.4
    </Value>
    <TimeStamp>
        123456789
    </Timestamp>
    <Value>
        5.4
```

```
    </Value>
</CollectedData>
```

2.5. The Control Service. The Control Service is an OSGi bundle which has to be started with every REFLECT application that should be accessed remotely by the REFLECT Remote Control Center. It makes use of R-OSGi in order to be accessible from a remote host. The Control Service offers several functionalities which are mandatory for the Remote Control Center, e.g. methods for getting the names of the actually running components and their properties or for setting modifiable properties.
To achieve this functionality the Control Service accesses the REFLECT Manager via the OSGi Service Registry. The Control Service serves as a facade to the REFLECT application.

2.6. The Reply Service. The Reply Service is necessary for establishing property listeners across different systems. This is achieved by dynamic proxy creation for the corresponding listener.

3. Connecting to a REFLECT application

3.1. Accessing REFLECT Applications using R-OSGi. Being able to access REFLECT applications remotely in order to retrieve information about its current state or set parameters is one of the most essential features for the Remote Control Center. As already mentioned in the introduction, R-OSGi was chosen to perform this task. Within this chapter i want to outline some topics that were important when implementing the Remote Control Center. The establishment of a connection can be divided into two parts: The actions that take place on client side referring the GUI and what happens to access the REFLECT application using R-OSGi where two different system are involved.

3.2. Basic overview of the connection process. 3.4 shows the simplified sequence of actions when connecting to a REFLECT application.

(1) The Connection to the Control Service is established using R-OSGi. Therefore the Control Service has to be running as an OSGi Bundle on the same system and within the same Virtual Machine, where the REFLECT Application resides. R-OSGi offers an abstraction from the underlying TCP/IP protocol, so once the connection has been established, the Control Service can be accessed by dynamic proxy creation.
(2) From this point, the Control Service tries to access the REFLECT Manager. The REFLECT Manager is also available as an OSGi-Bundle and can be fetched using the OSGi Registry. The Manager serves as an interface to the REFLECT application.
(3) The REFLECT Manager is able to access all running components of the running REFLECT Applications. It performs the designated actions (e.g. setting or retrieving values) and returns the result ,if necessary.

3.3. Making a connection request from client side view. The establishment of a connection can be invoked by the user by either using the “Connect” option from the “Host” menu or by right-clicking on the “Not connected” node within the Component Browser and choosing the “Connect” option.
3.5 shows the simplified sequence of actions on client side invoked when a connection request is invoked.
At first it is checked, if it’s possible to connect to the specified host and port, which have previously been set by the user using a dialog window. The following

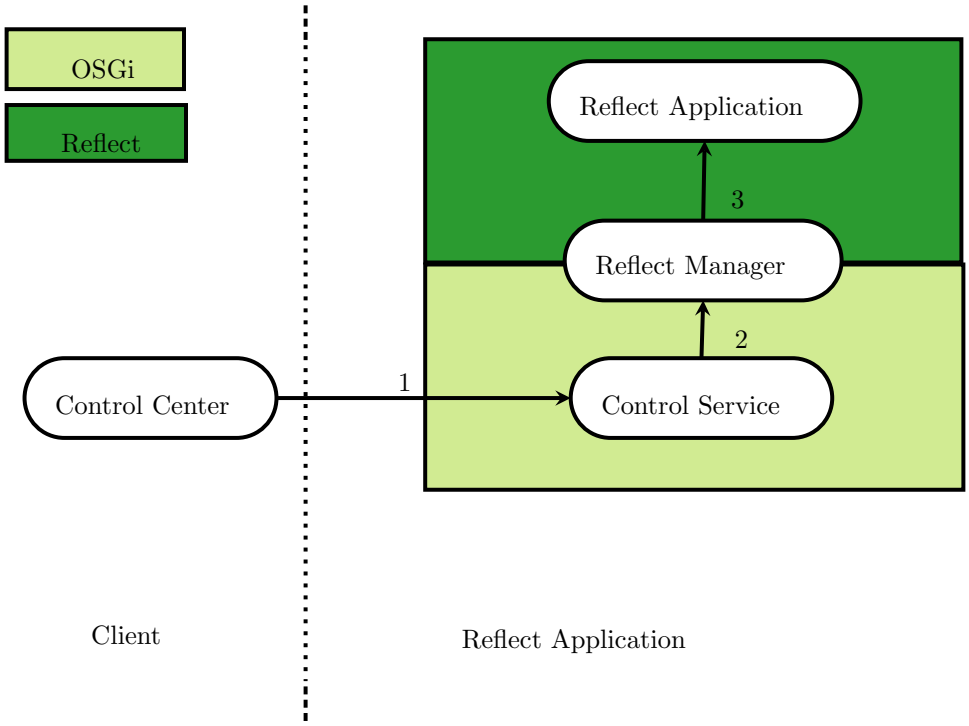


FIGURE 3.4. The basic connection process

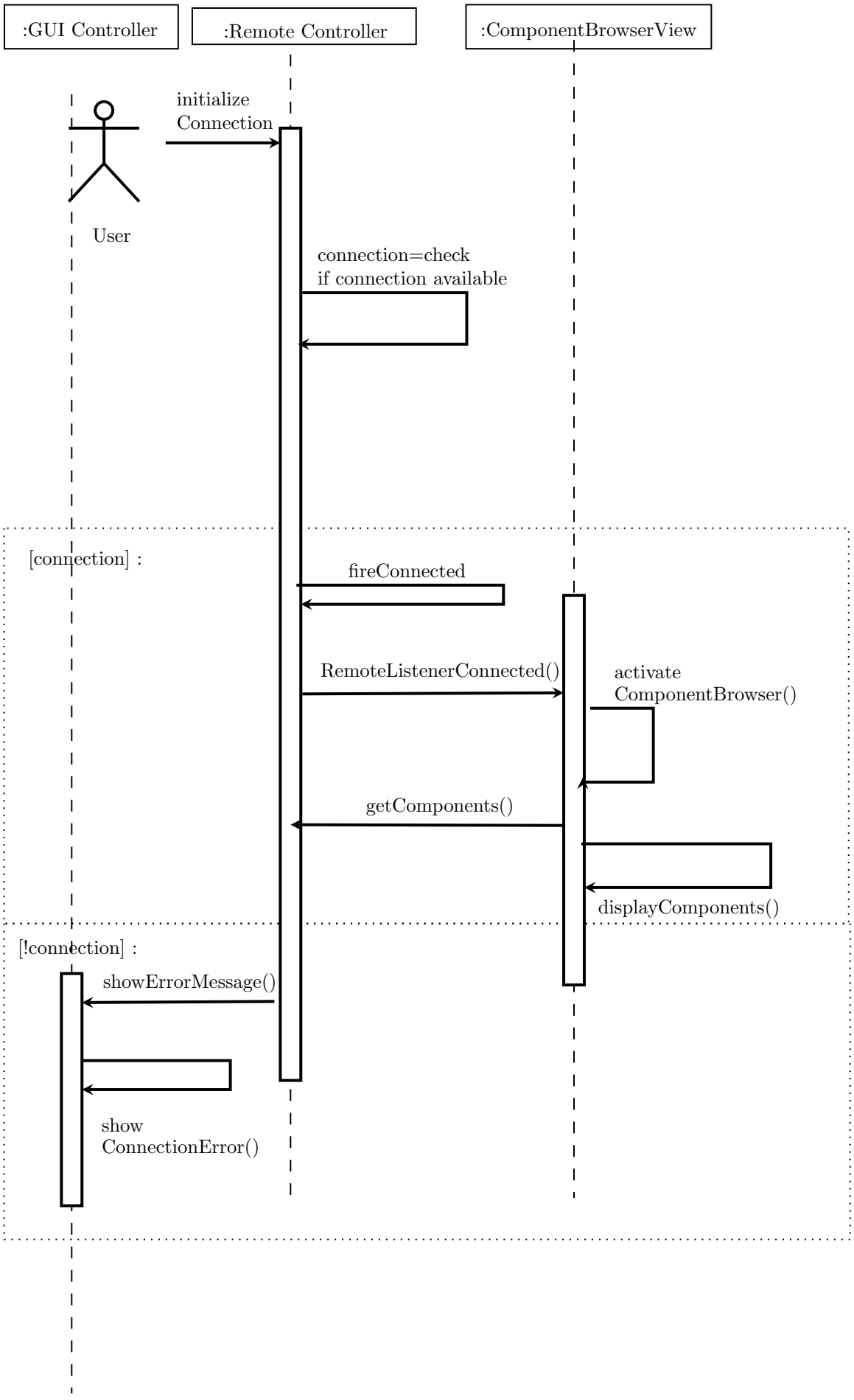
requirements have to be fulfilled in order to connect to a REFLECT application successfully:

- The IP which was given as host has to be reachable.
- A Control Service bundle was loaded with the REFLECT application to which the connection request is
- R-OSGi has successfully been started at the remote host
- The REFLECT manager as part of the REFLECT framework can be accessed by the OSGi Service Registry on application side

If the connection can't be established, an error message is displayed using the GUIController. Otherwise the *fireConnected*() method is invoked and all registered Remote Listeners are notified. An example for an Remote Listener is the Component Browser View. When it is notified, it initializes the Component Browser and uses the Remote Controller to fetch information about the components to be displayed.
What is not shown within this sequence diagram? The menu items for connecting and disconnecting are also registered as listeners and are toggled according to the given connection status.

3.4. An example: Setting a component property. 3.6 shows the program flow of setting the value of specific component property. On client side the sequence is initiated by the Remote Controller. At first it tries to fetch the ControlService described by a ServiceReference, which has been initialized when the connection to this REFLECT application was established. The following actions all take place on server side (indicated by the green box). The Control Service tries to fetch the REFLECT Manager using OSGi, as the manager itself is running as an OSGi bundle. The property is then set using the REFLECT Manager.

3.5. Remote Property Listeners. Remote Property Listeners are important for the Control Center as they are used by both the Property Monitor for getting notified about changed data and the Property Editor for getting notified about changed Property Domains. The current version of the REFLECT framework already offers the possibility to register local Property Listeners , which is used to coordinate dependencies between different components. The Remote Control Center offers the possibility to register Remote Property Listeners, which makes use of the ServerRemoteListenerManager, which handles the local Property Listeners on REFLECT application side and the ClientRemoteListenerManager, which handles the remote listeners on the side of the Remote Control Center. When a Remote Property Listener is registered by the Remote Control Center, the ServerRemoteListenerManager adds a new local Property Listener on REFLECT application side and saves the connection to the accordant Remote Control Center, while the ClientRemoteListenerManager handles the connection to the according component of the Control Center (e.g. a PropetyMonitor). 3.7 gives an overview of what happens when a property's domain or values changes. At first, the ServerRemoteListenerManager, which is part of the Control Service, is notified and checks, if ServerRemoteListeners, which represent clients from different hosts, have been registered and are listening at this specific property. The previous registration has taken place using the Control Service. If no listeners for this property are present, the ServerRemoteListenerManager removes itself (as a listener) from the property that has send the update message. If there are listeners present, they get notified using R-OSGi and the ReplyService available on client side. The ReplyService delegates the message to the ClientRemoteListenerManager, which serves as a proxy and checks if listeners for specified component and property are present and if so , notifies them. If there isn't any listeners, it tells the ServerRemoteListenerManager to remove this client from its listener list.
Using this approach, unnecessary listener connections are removed the first time they are detected , as both Server- and ClientRemoteListenerManager check if there are any listeners at all for which the received update message is of relevance.


:GUI Controller
:Remote Controller
:ComponentBrowserView
initialize
Connection
User
connection=check
if connection available
[connection] :
fireConnected
RemoteListenerConnected()
activate
ComponentBrowser()
getComponents()
displayComponents()
[!connection] :
showErrorMessage()
show
ConnectionError()

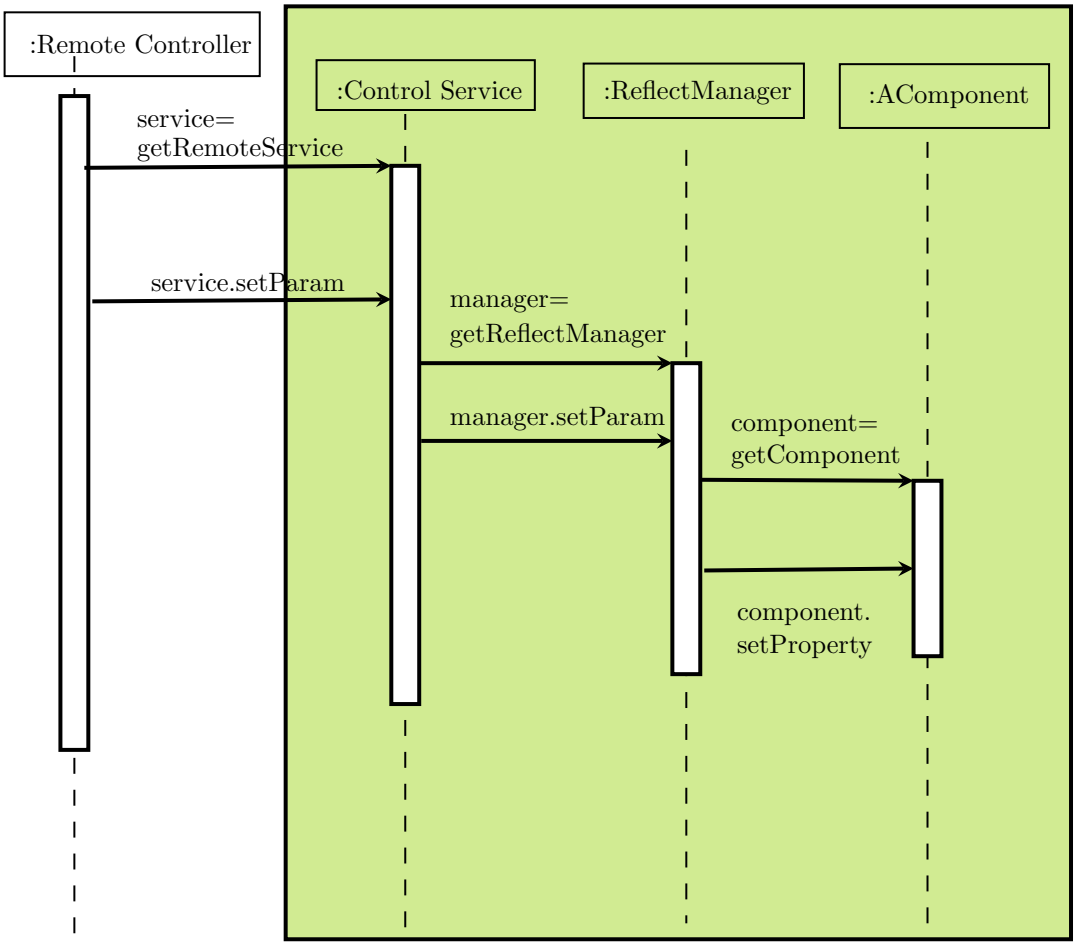


FIGURE 3.6. Set a property remotely using R-OSGi

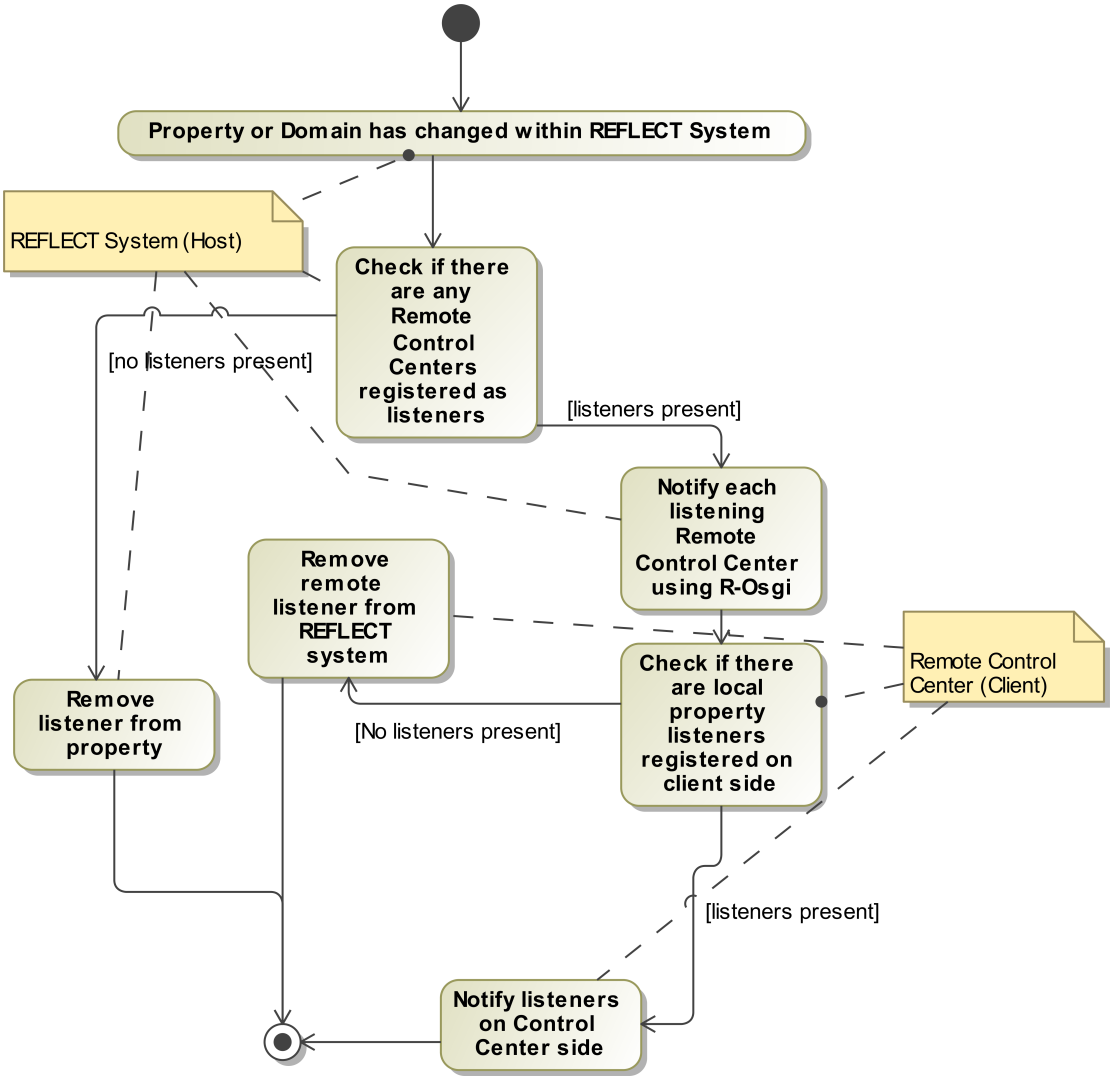


FIGURE 3.7. Remote listener functionality

CHAPTER 4

Adding Property domain functionality to the REFLECT Framework

1. Property Domains

While developing of the REFLECT Remote Control Center, several new possibilities for creating Property domains were introduced.
Domains can now be associated with specific properties within the component class by using Java Annotations.
Property Domains are important for the Remote Control Center, as they are transferred to the Control Center with every property which should be edited. When the users edits a property and chooses to commit his changes, the designated value is at first validated against the property domain on client side. If the values isn't valid, an error message is shown and the value won't be set within the REFLECT application. If the values from the Control Center would be passed unchecked, an exception on server side would be thrown and the client would be notified that an error has occurred while trying to set the property. By validating the value already on client side, this exception and the resulting network traffic can be prevented.
Property Domains are also needed for choosing the appropiate input method for a specific property within the Property Editor. This would be a slider for a RangeDomain or a Combo Box for a CollectionDomain. One example application would be a sensor component with two properties. One could determine which kind of camera should be used for this sensor, if there is more than one available. This would be represented best by a CollectionDomain. Another property of the sensor component could be the sensitivity of the sensor, for which only values within a certain range make sense, so that this property would be restricted by a RangeDomain.

2. RangeDomains

```
@Property domain="[12.6,20.8["
```

would associate a RangeDomain with lower bound 12.6, upper bound 20.80 ,lower included and upper bound excluded to the annotated domain. The type of the resulting domain depends on the type of the property, currently Integer and Double are supported for Range Domains.
You can also use inf to set the minimum (maximum) of the corresponding data type, e.g.

```
@Property domain="]inf,inf]"
public void setProp(int new_val){ ...
```

would result in a Range Domain of type Integer, which lower bound is the minimum Integer value and the maximum Integer value as its upper bound, where the lower bound is excluded while the upper bound is included.

3. CollectionDomain

A Collection Domain is primarily intended to give options between several fixed possible settings.

```
@Property domain="{Option1,Option2,Option3}"
```

would result in a Collection Domain with the Elements Option1,Option2 and Option3. Please Note that CollectionDomains, that contain characters, are only valid for Properties of type String.
Other supported data types for Collection Domains are Integer and Double.

4. Dynamic Property Domains

An alternative way to associate property Domains to specific properties is not to annotate the property itself but to declare a method within this class as an Dynamic Property Domain. This way the Domain for a property can change at an any time corresponding to the class'es internal state or other parameters that may influence the range of allowed values for a property. An example:

```
private double fInternalValue;

.
.
.


@Property
public double getDoubleVal(){
    return fDoubleVal;
}

@Property
public void setDoubleVal(double val){
    fDoubleVal=val;
}

@DynamicPropertyDomain(value="doubleVal")
public IPropertyDomain getPropertyDomain(){
    return new RangeDomain<Double>(fInternalValue, 20.53, Double.class, false,
    true);
}
```

The above example would result in RangeDomain which lower bound is dynamically fetched from the attribute "fInternalValue" of this component. **Note:**
As a component programmer you have to make sure to call a components's *updatedProperty* method to make sure that all property listeners are noticed about the changed domain.

5. Full Domains

If no domain description is given for a property, a FullRange Domain will automatically be associated to this property. A FullRange Domain contains every possible element which is of the corresponding type.

CHAPTER 5

Future prospects

Within this chapter I want to outline some ideas about how the Control Center could be extended or improved.

1. Improving existent functionality

1.1. Improving the Property Monitor functionality. A reasonable way for adding functionality to the existent Remote Control Center would be to extend the Property Monitor. At the moment, the Property Monitor can only be used to monitor properties of type *Integer* or *Double*. In both cases the JFreeChart [**Lim09**] was used. The library seemed to be a good choice referring to the integration into an RCP architecture, but it has its limitation when it comes to live rendering of data. One idea for improving the Property Monitor would be to use a different charting framework which is more optimized for live plotting and maybe even offers the possibility to select certain areas of the plot in order to save only a part of the collected data.
Another idea for improving the current version is surely to introduce the possibility to monitor various data type. One example would be the type *String*, it could be simply monitored by printing its current value into a not-changeable text field. The collected data could be saved in a similar way to how it's done for *Double* or *Integer*.
Maybe the best and most generic way would be to use RCP's extension functionality to add plugins for monitoring specific types. By this way, new features could easily be added to the existing architecture.

1.2. Upgrading the connection handling for Reflect App. In order to access the Reflect application remotely, R-OSGi [**Rel09**] was used for the implementation of the Remote Control Center. In the meantime R-OSGi has been enhanced so that its services can now be accessed as Web Services and the newer version claims to be more stable. A reimplementation of the Control Center could benefit from the new possibilities. Another idea would be to implement a "Reflect Discovery" feature in order to scan for Reflect application within a given network for being able to access the systems without knowing the machine's IP addresses.

2. Ideas for extending the Control Center

2.1. Listening on ports of components. A convenient feature for future version would be the ability to monitor data which is transferred between different components using the Reflect ports. This could be done similar to the way properties are monitored but also the introduction of new elements for the GUI could make sense.

2.2. Dynamic instantiation of components. It would be useful feature if components with corresponding properties could be created using the GUI of the Remote Control Center while the Reflect application is running. This would offer the possibility to adapt the behavior of a running application in a profound

way without the need for restarting it. It would also make sense being able to connect the newly instantiated components using their associated ports. Both of the mentioned ideas would offer a possibility for fast dynamic adaption of a running software system or experimenting with different components.

CHAPTER 6

Conclusion

Using the Remote Control Center it's possible to connect to REFLECT systems and monitor and edit properties of components on the fly. To establish the connection to the REFLECT system, r-OSGi is used. The Remote Control Center GUI has been built on the Eclipse Rich Client Platform. It can be extended by using Eclipse's plugin functionality.

Bibliography

[All09] The OSGI Alliance. About / osgi technology. http://www.osgi.org/About/Technology, 2009.

[eli09] elipse.org. Swt: The standard widget toolkit. http://www.eclipse.org/swt/, 2009.

[eW09] eclipse Wiki. Rich client platform. http://wiki.eclipse.org/index.php/Rich_Client_Platform, 2009.

[Lim09] Object Refinery Limited. Jfreechart. http://www.jfree.org/jfreechart/, 2009.

[Rel09] Jan S. Rellermeyer. Maven - r-osgi - transparent osgi remote extension for distributed services. http://r-osgi.sourceforge.net/, 2009.