Laboratoire d'Informatique de Grenoble
Unité de recherche UGA, Grenoble INP, CNRS
UMR 5217
700 AVENUE CENTRALE
CS 40700,
F-38058 GRENOBLE CEDEX 9 FRANCE

RoZ Notes v2.0

Yves Ledru

Univ. Grenoble Alpes, CNRS, Grenoble INP
LIG, UMR 5217,
F-38000, Grenoble, France
Yves.Ledru@imag.fr

 February 2018

Chapter 1

RoZ in a few words

RoZ is a model transformation program which takes as input a class diagram, enhanced with Z annotations, and produces a Z specification. The Z specification includes the types and data structures of the model, and adds basic operations (constructors, destructors, setters, getters) to the Z specification. It also allows to provide user-defined operations.

JazaGUI is a graphical user interface which integrates RoZ with the Jaza animator. It also allows to represent the current state of the animation as an object diagram.

RoZ combined with Jaza has been presented in the following papers:

- Y. Ledru,
  "Using Jaza to animate RoZ specifications of UML class diagrams"
  ZUM'06, April 2006, postproceedings IEEE CS Press
- Sophie Dupuy, Yves Ledru, Monique Chabre-Peccoud,
  "An Overview of RoZ: A Tool for Integrating UML and Z Specifications."
  CAiSE 2000: 417-430, LNCS 1789, Springer

1.1 What this version of RoZ does not

Until 2017, the original version of RoZ was distributed as a plugin for Rational Rose. With the advent of eclipse and Papyrus, it has been ported to this platform. Most features of the original RoZ are supported except the generation of proof obligations as presented in:

- Yves Ledru
  "Identifying Pre-Conditions with the Z/EVES Theorem Prover."
  ASE 1998: , IEEE CS Press

If you feel that it should be a mandatory feature, please contact Yves Ledru.

1.2 RoZ Licensing

1. RoZ is distributed freely for non commercial research. The software is distributed without any guarantee.
2. If you are planning to use RoZ for commercial activities please contact Yves.Ledru@imag.fr.
3. The contents of the lib directory are copied from the standard eclipse (luna) distribution and are subject to the EPL license.
4. The Jaza animator is Copyright (C) Mark Utting, 2000-2005. It is available from `https://github.com/uho/jaza`.

Chapter 2

RoZ Installation

2.1 Prerequisites

2.1.1 Eclipse and Papyrus

This new version of RoZ takes as input UML models created with the eclipse environment, in particular with the Papyrus tool. RoZ has been tested with

- eclipse Luna under windows, with Papyrus 1.0
- eclipse Mars under windows, with Papyrus 1.1
- eclipse Neon under windows, with Papyrus 2.0
- eclipse Oxygen under windows, with Papyrus 3.2

In order to use RoZ, you will need a version of eclipse with Papyrus. You can get such a version of Papyrus by downloading *Eclipse Modeling Tools* from `http://www.eclipse.org/` , then install Papyrus via *Install more Modelling Tools* or via *Help>Install Modeling Components*.

2.1.2 Graphviz

JazaGUI uses Graphviz to visualize the object diagram.

You can download Graphviz from `https://www.graphviz.org/`

2.1.3 Java

JazaGUI is a java application and requires to have a java JRE installed on your machine. The command "java" must be on your path.

2.1.4 Optional items

- This distribution of RoZ includes the jaza animator. But you might want to read its documentation which is available at `https://github.com/uho/jaza/tree/master/doc`.
- Community Z tools (`http://czt.sourceforge.net/`) is a set of tools for Z. They can be loaded as an eclipse plugin to check that Z specifications are syntactically correct.

2.2 Installation of RoZ and JazaGUI

RoZ is embedded in JazaGUI. In order to install RoZ, you need to get a zip of RoZandJazaGUI. Unzip it on your machine, and don't modify the structure of directories!

You must modify file *jazaGUI.bat* and add the address of *dotty.exe*, located in the Graphviz distribution.

1. Unzip *RoZandJazaGUI.zip* into a directory (typically, `C:\Program Files\JazaGui`)
2. Open the *jazaGUI.bat* file with a text editor. It includes a line of the form
   `java -jar %~dp0\JazaGUI_Project.jar %~dp0\jaza.exe "C:\...\bin\dotty.exe"`
3. Replace `"C:\...\bin\dotty.exe"` with the path to the *dotty.exe* executable on your machine.
4. Add the directory which includes *jazaGUI.bat* to your path.[1]

[1] To set the environment variable, right click on the computer icon in the windows explorer and select *properties*, then *advanced system settings* and click the *Environment Variables* button.

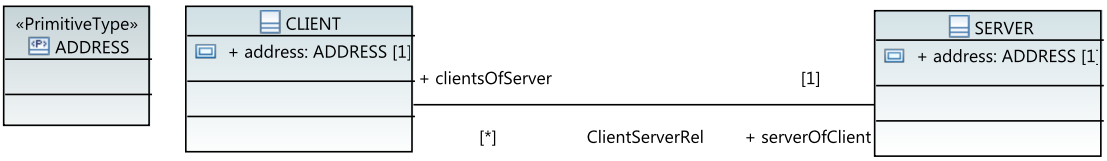
Chapter 3

Try it on a simple example

3.1 A simple model

In this chapter, we will work on a simple model which does not include Z types or invariants. We will see how this class diagram can be translated into Z and how it can be animated with Jaza.

The proposed model features classes CLIENT and SERVER. Both have a single attribute, which is the IP address of the machine. A server is linked to several clients, and a client is linked to one and only one server. ADDRESS is declared as a primitive type.



Please note that the current version of RoZ does not support inheritance. So we cannot model this problem by specializing an abstract class, which stores the address, into clients and servers.

3.2 Import the model into eclipse

In the *Examples* directory of the RoZandJazaGUI distribution, you will find two versions of a client-server example. Let us import the simplest version as an eclipse project:

1. Open Eclipse.
2. Import the file ClientServer.zip in your Eclipse workspace. (`File>import>General` , then `Existing project into workspace`, then `select archive file`.
3. Select file `PapyrusClientServer.zip` and click on `Finish`.

3.3 Launching RoZ and JazaGUI

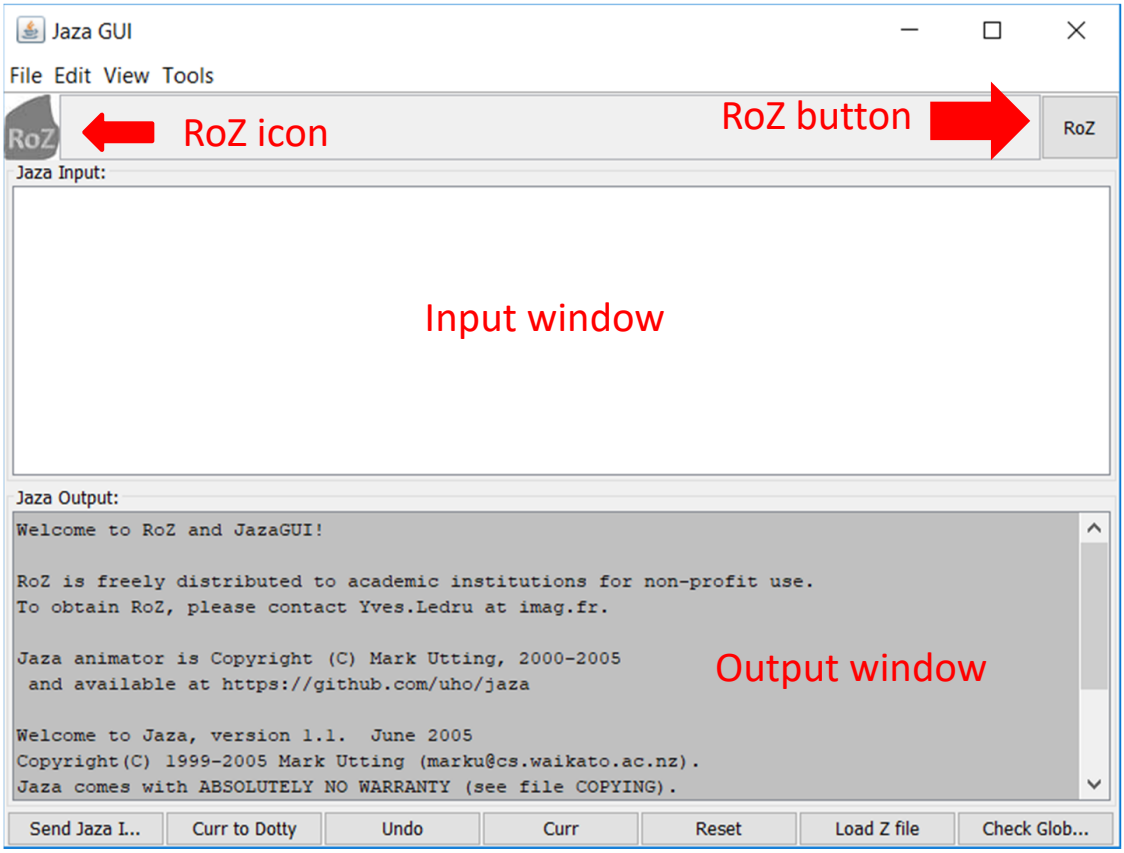
1. Open a command window and go to the directory which stores the client server model:

   ```
   > cd PapyrusClientServer
   ```

2. Lauch JazaGUI:

   ```
   > jazagui
   ```

This will open a window like this one:



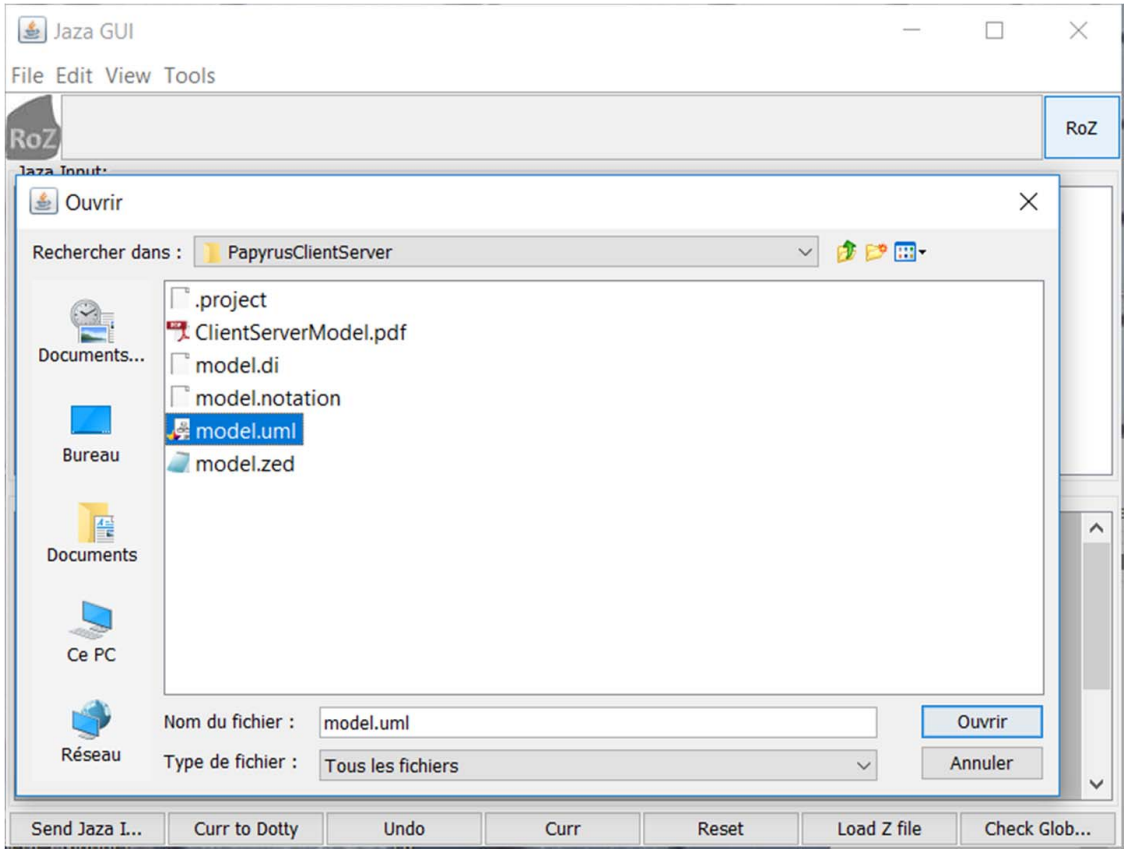
This window includes :

- A **RoZ button**, which launches the RoZ transformation.
- A RoZ icon, which is stored on a machine of our laboratory. This will eventually provide us with anonymous information about the usage of RoZ. If your machine is not connected to the Internet, JazaGUI waits for a small time, and then starts normally. This will simply slow down the launch of JazaGUI.
- An input window for inputs to send to the jaza animator.
- An output window which displays the answers of the animator.
- Several menus and buttons to interact with Jaza or RoZ.

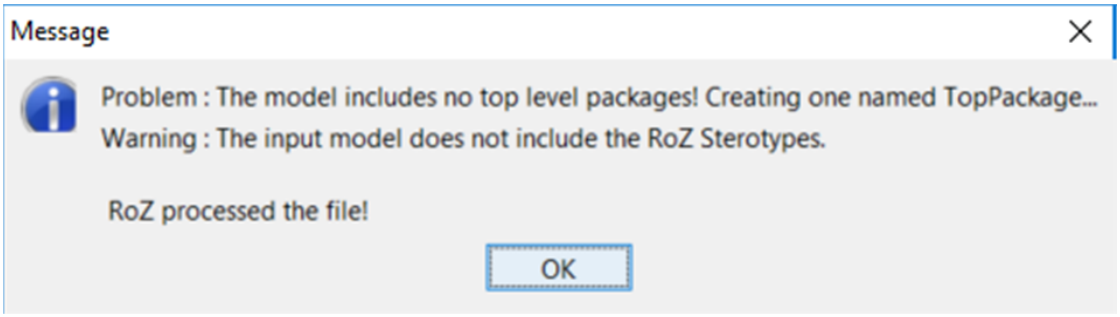
3.4 Translation into Z

1. Click on the RoZ button, at the top right of the JazaGUI window. This will prompt a choose window displaying the contents of the directory where JazaGUI was launched. [1]

[1] Actually, this choose window is only opened the first time you click on the button. Subsequent clicks will automatically select the same file. If you need to process another file, choose *'Tool> RoZ another UML File'*.



2. choose the model.uml file and open it.
   This will generate a *model.zed* file and prompt the following warnings:



The contents of the *model.zed* file are explained in appendix.

3.5 Model animation with Jaza

3.5.1 Loading the Z model in Jaza

Click on "*Reset*" to empty the jaza workspace and "*Load Z File*" to load file `model.zed`. The Jaza output window will display the following message:

```
Loading 'C:\Users\ledru\workspaces\LunaRoZ\PapyrusClientServer\model.zed' ...
Added 41 definitions.
JAZA>
```

If the zed model is not syntactically correct, an error message will be displayed.

3.5.2 Initializing the animation

The following commands are stored in file SCENARIO.TXT . You can copy and paste them in the input window, using CTRL-V to paste them.

You should now call an initialisation operation of the zed specification. This will create the first state of your animation. To do this, you should call the "`do`" command and provide the initialisation operation as argument. Type the following text in the Jaza input window, then hit the return key or click on "*Send Jaza Input*"

```
do Init\_model
```

The output window displays the following text:

```
\lblot Client'==\{\}, Server'==\{\}, clientsOfServer'==\{\},
       serverOfClient'==\{\} \rblot
JAZA>
```

It corresponds to the current state of the animation. In this initial state, the sets which include all current instances of the classes are empty, and the functions corresponding to the association roles are also empty.

Let us create an object of class Server:

```
; Server\_new
```

The semi-colon expresses that this operation is applied on the current state. The output window displays the following message, asking for an object of class Server:

```
Input server? =
```

The next input is a tuple including all attribute values for the object:

```
\lblot address == "1.2.3.4" \rblot
```

The resulting state is now displayed:

```
\lblot Client'==\{\}, Server'==\{\lblot address=="1.2.3.4" \rblot\},
       clientsOfServer'==\{\}, serverOfClient'==\{\} \rblot
JAZA>
```

Now, the set storing the instances of class Server has a unique element.

We can create a second server. Let us input:

```
; Server\_new
\lblot address == "5.6.7.8" \rblot
```

Which produces the following state:

```
\lblot Client'==\{\},
       Server'==\{\lblot address=="1.2.3.4" \rblot,
              \lblot address=="5.6.7.8" \rblot\},
       clientsOfServer'==\{\}, serverOfClient'==\{\} \rblot
```

Let us now create an instance of class Client. A closer look at the class diagram tells us that each client is linked to a server. So we should invoke an operation which creates the client and its link simultaneously.

```
; Client\_newWithLinks
```

This operation first requires a client:

```
Input client? =
```

Let us input a client:

```
\lblot address == "11.12.13.14" \rblot
```

then it requires a server

```
Input serverOfClient? =
```

Let us answer with an existing server:

```
\lblot address == "1.2.3.4" \rblot
```

This produces the following state:

```
\lblot Client'==\{\lblot address=="11.12.13.14" \rblot\},
       Server'==\{\lblot address=="1.2.3.4" \rblot,
  \lblot address=="5.6.7.8" \rblot\},
       clientsOfServer'==\{(\lblot address=="1.2.3.4" \rblot,
    \{\lblot address=="11.12.13.14" \rblot\})\},
       serverOfClient'==\{(\lblot address=="11.12.13.14" \rblot,
   \lblot address=="1.2.3.4" \rblot)\} \rblot
```

We can create a second client:

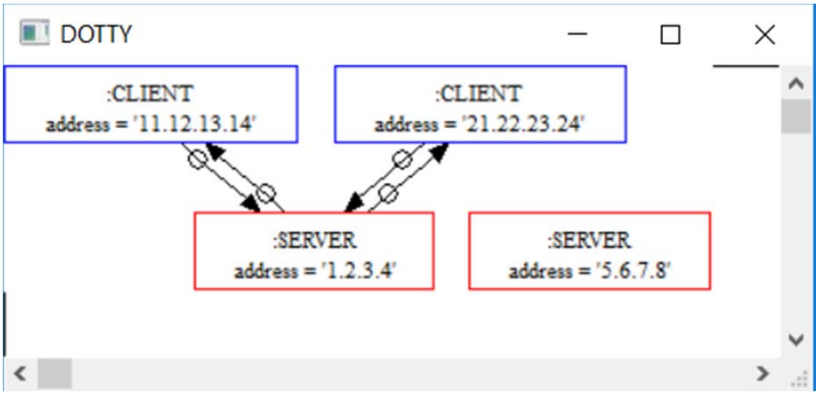
```
; Client\_newWithLinks
\lblot address == "21.22.23.24" \rblot
\lblot address == "1.2.3.4" \rblot
```

which produces the following state:

```
\lblot Client'==\{\lblot address=="11.12.13.14" \rblot,
  \lblot address=="21.22.23.24" \rblot\},
       Server'==\{\lblot address=="1.2.3.4" \rblot,
  \lblot address=="5.6.7.8" \rblot\},
       clientsOfServer'==\{(\lblot address=="1.2.3.4" \rblot,
    \{\lblot address=="11.12.13.14" \rblot,
      \lblot address=="21.22.23.24" \rblot\})\},
       serverOfClient'==\{(\lblot address=="11.12.13.14" \rblot,
   \lblot address=="1.2.3.4" \rblot),
  (\lblot address=="21.22.23.24" \rblot,
   \lblot address=="1.2.3.4" \rblot)\} \rblot
```

3.5.3 Visualizing the current state

The current state becomes difficult to read. Therefore, you can call "Curr to Dotty" which will use *dotty.exe* from the GraphViz toolset to display an object diagram of the current state. This opens the following window.



The window diplays our two servers and our two clients.

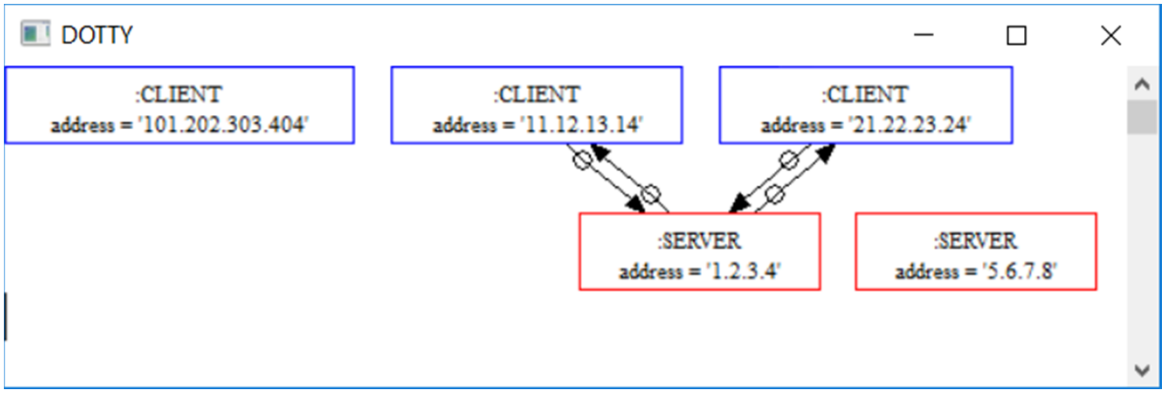
3.6 Checking incorrect states and getting explanations

Each Z operation checks that its scope verifies the associated invariant properties in both initial and final states. Unfortunately, it is possible to reach states which are incorrect with respect to properties out of the scope of the operations which led to this state.

For example, let us create a client without server:

```
; Client\_new
\lblot address == "101.202.303.404" \rblot
```

This produces the following (erroneous) state.



This state does not conform to the class diagram, because the third client should be linked to a server. Unfortunately, this was not noticed by the jaza animator because the scope of operation `Client\_new` is limited to class CLIENT.

We can force evaluation of all invariant properties on the current state by calling operation `CheckGlobalInvariant`, or hitting the "*Check Global Invariant*" button. This produces the following result:

```
No solutions
JAZA>
```

Actually, `CheckGlobalInvariant` is an operation which does nothing, but applies to the largest scope. Therefore, it checks all invariants of this scope in its initial state, and finds constraints that are not satisfied. You can then ask Jaza why it does not find solutions by typing :

```
why
```

This produces the following result to explain why it does not find solutions:

```
\begin{schema}{CheckGlobalInvariant}
1  false  \because{\bigcup
\ran
\{(\lblot address=="1.2.3.4" \rblot,
   \{\lblot address=="11.12.13.14" \rblot,
     \lblot address=="21.22.23.24" \rblot\})\} =
\{\lblot address=="101.202.303.404" \rblot,
  \lblot address=="11.12.13.14" \rblot,
  \lblot address=="21.22.23.24" \rblot\}; }
2  Client'==\{\lblot address=="101.202.303.404" \rblot,
   \lblot address=="11.12.13.14" \rblot,
   \lblot address=="21.22.23.24" \rblot\}:\finset
  \{address:ADDRESS @ \lblot address==address \rblot\}
3  ...
...
The maximum number of true constraints was 0.
JAZA>
```

Here, it reports that it was not able to satisfy any constraint. So the first constraint is what caused a problem. This first constraint tells that :

```
1  false  \because{\bigcup
\ran
\{(\lblot address=="1.2.3.4" \rblot,
   \{\lblot address=="11.12.13.14" \rblot,
     \lblot address=="21.22.23.24" \rblot\})\} =
\{\lblot address=="101.202.303.404" \rblot,
  \lblot address=="11.12.13.14" \rblot,
  \lblot address=="21.22.23.24" \rblot\}; }
```

It actually explains that the range of the *clientsOfServer* function should contain all three clients.

3.7 Incorrect operation calls

Let us consider a smaller scenario, where one tries to create a client associated to a non-existing server. Here, the association between servers and clients is in the scope of the operation, so its associated invariant properties should be checked before and after the execution of the operation.

```
do Init\_model
; Client\_newWithLinks
\lblot address == "11.12.13.14" \rblot
\lblot address == "1.2.3.404" \rblot
```

This scenario computes the initial state correctly, but then reports "`No solutions`" for the operation. Asking "`why`" produces the following explanation:

```
...
5  false  \because{\lblot address=="1.2.3.404" \rblot \in \{\}; }
...
The maximum number of true constraints was 4.
```

This means thatt the first four constraints were satisfied, but not constraint 5 which requires that the server belongs to the set of existing servers. Unfortunately, this set is currently empty.

3.8 Additional commands of JazaGUI

Undo button Click on this button to undo the last operation call. You may only undo the last operation.

File > Open Input File :
Command "Open Input File" loads an input file, like `Scenario.txt` in the input window of JazaGUI.

View > History The Input history is the list of all commands that were input to Jaza. This is particularly useful to record an animation into a scenario file.

The Output history is the list of all text that appeared in the output window. The Input and Output history mixes the two histories to display commands and responses.

Tools > RoZ another File The command *RoZ another File* allows to change the UML input file for RoZ.

3.9 Conclusion

In this chapter, we have seen how a class diagram can be translated in Z by RoZ and animated using Jaza. The connection to dotty allows to visualize the current state as an object diagram. We have also seen how invariant properties are checked on the scope of operations, or can be forced to check in the global scope. Finally, the use of the "*why*" command provides explanations on the failure of operations.

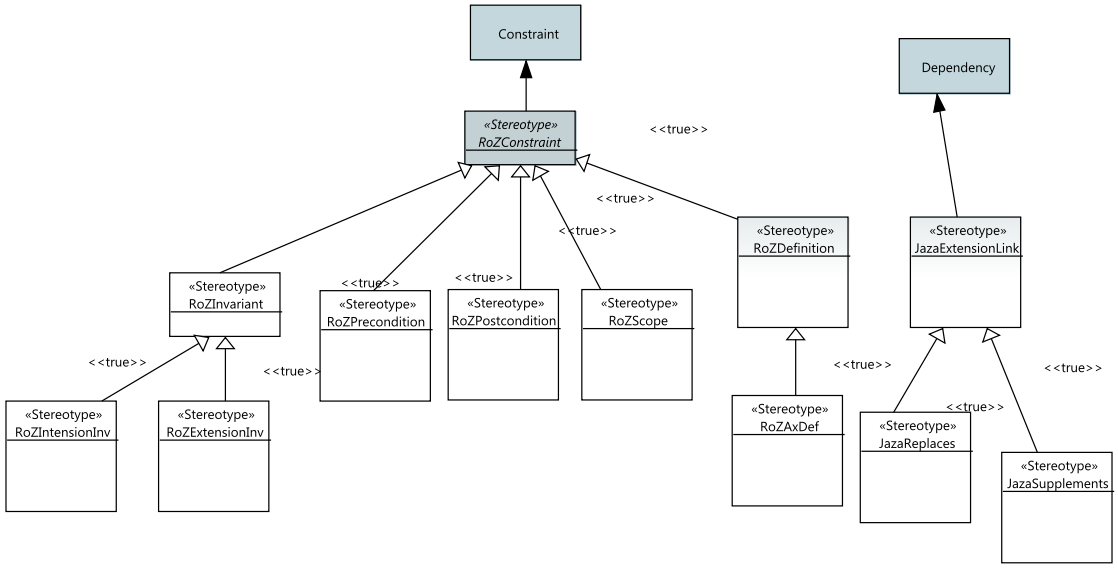
At this stage, we did not see Z specifications. This shows that the tool can be used without mastering formal specifications. In the next chapter, we will see how Z annotations can be added to the diagram and inserted in the formal specification.

Chapter 4

A more elaborated example

4.1 RoZ profile

We will now consider an advanced version of the client server example which uses the RoZ profile and annotates the model with Z properties. But this requires to use a UML profile, named `RoZConstraintProfile` and defined in `RoZConstraint.profile.uml`. The profile diagram is the following one:



This diagram defines stereotypes for constraints and UML dependencies. In this more elaborated example, we use the following stereotypes:

- `RoZDefinition` applies to constraints which refine a type definition.
- `RoZInvariant` applies to constraints which express an invariant at the level of an association or a package.
- `RoZExtensionInv` applies to constraints expressing an invariant on the extension of a class[1].
- `RoZIntensionInv` applies to constraints expressing an invariant on the class intension[2].

The sterotype associated to a constraint is used by RoZ to locate the adequate place for the constraint in the Z specification.

[1]The extension of a class is the set of objects of the class currently available in the information system.
[2]The intension of a class is the set of all possible objects of the class. The extension of a class is included in its intension.

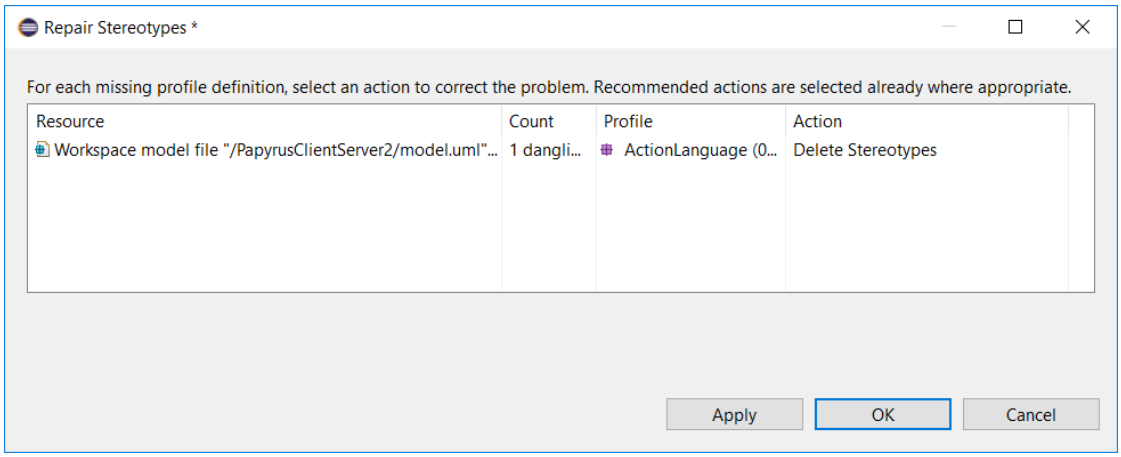
4.1.1 Installing the RoZ profile

The RoZ profile is like a project, stored in the `RoZProfile.zip` file of the Examples directory. So you should install it in your workspace with the *import* command, and select "*Existing projects into workspace*".

4.1.2 The new Client Server project

4.1.3 Installing the new Client Server Project

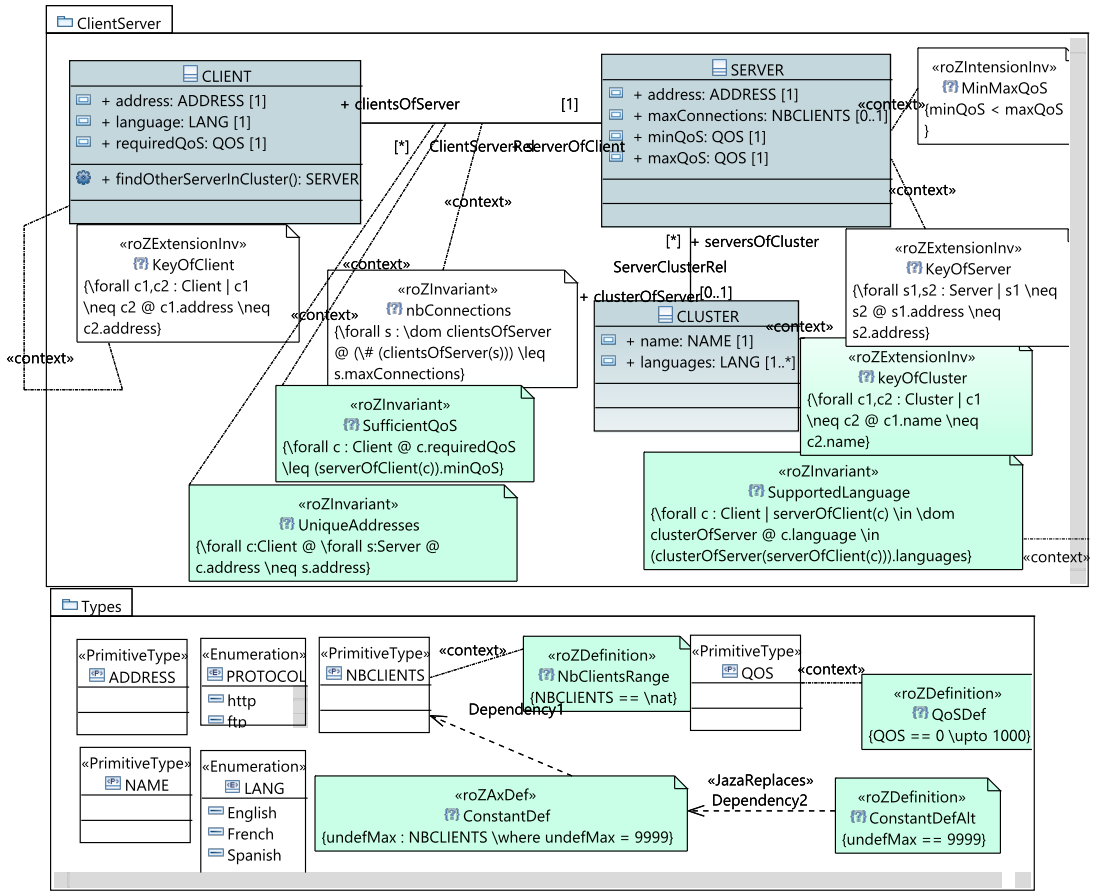
The new client server project is stored in file `PapyrusClientServer2.zip`. Install it in your workspace as a project. When you will first open the model, it might prompt the following window:



You should simply delete these stereotypes, by applying the recommended action.

4.1.4 Class diagram

The class diagram introduces an additional class (CLUSTER) and more class attributes. It also introduces the types of these attributes, several constants, and numerous constraints. RoZ also allows to add operations and their specifications.



Type definitions Most types are introduced as `PrimitiveType`. ADDRESS and NAME are introduced this way. These types will be translated as Z given types. Some types are enumerated types (PROTOCOL and LANG) and are introduced as a UML Enumeration. These types will be translated as Z enumerated types.

Type QOS is introduced as a Primitive type, associated to a constraint stereotyped as `roZDefinition`. This constraint specifies that QOS is an integer interval ranging from 0 to 1000.

NBCLIENTS is also associated to a `roZDefinition`, whose constraint says that NBCLIENTS is equal to the set of natural numbers. So this type is simply a renaming for naturals. A constant is associated to this type: `undefMax` is defined as natural number 9999. The constant definition takes two forms: an axiomatic definition, which is compatible with classical Z type-checkers. Unfortunately, jaza does not support axiomatic definitions. Therefore, a second definition is given where `undefMax` is defined as a syntactic equivalence to 9999[3].

Class SERVER Class SERVER features four attributes: the IP address, the maximum number of connections it can support, its minimum and maximum Quality of Service. An invariant expresses that the minimum QoS is always smaller than the maximum QoS. This is an intension invariant, i.e. it is ver-

[3]The current version of RoZ only generates the specification for Jaza. In future versions, an option might be provided to choose between axiomatic definitions and their Jaza counterpart.

ified by every object of the class. It is expressed by constraint MinMaxQoS, which is stereotyped as `RoZIntensionInv`.

Another constraint applies to class SERVER: the IP address is a key for the class. This is expressed by constraint KeyOfServer which applies to the extension of the class. It is stereotyped as `RoZExtensionInv`.

Class CLIENT Class CLIENT has three attributes: the IP address, the language of the client, and the required quality of service. Only one constraint applies to this class: it is the key constraint, expressed by KeyOfCLient and stereotyped as `RoZExtensionInv`.

Class CLUSTER A server may belong to a cluster. A cluster is characterized by its name and the set of languages it supports. The name is the key of class CLUSTER. This is expressed by constraint keyOfCluster stereotyped as `RoZExtensionInv`.

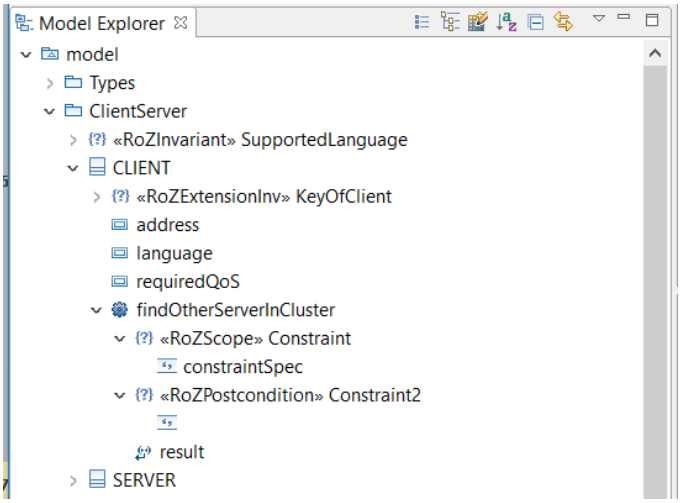
ClientServerRel Association Every client is linked to one and only one server. This association is also the place to express three constraints, stereotyped as `RoZInvariant`:

- nbConnections expresses that the number of clients attached to a server is always less than or equal to the maximum number of connections supported by the server.
- SufficientQoS expresses that the QoS required by the client is less than or equal to the minimum QoS offered by the server
- UniqueAddresses expresses that the addresses of servers are disjoint from the adresses of clients.

ServerClusterRel Association This association expresses that a cluster may include several servers, and that each server belongs at most to one cluster. No constraint is associated to this association.

Package ClientServer The three classes and the two associations are grouped in a package. This is the place to express global constraints which link attributes of CLIENT with attributes of CLUSTER. Here constraint SupportedLanguage, stereotyped as `RoZInvariant`, expresses that the language of the client is one of the languages of the cluster.

Operation findOtherServerInCluster RoZ automatically generates basic operations (constructors, destructors, setters and getters), but application specific operations may also be necessary. Besides the type definitions and the constraints expressed at every level of the class diagram, RoZ also supports the user definition of operations. These user-defined operations are declared as methods of the class, and use constraints to express post-conditions, and to specify their scope. These stereotyped constraints for scope and post-condition do not appear in the graphical UML view, but in the model explorer of eclipse, as shown on the following screen shot:



findOtherServerInCluster is an example of such application specific operations. This operation takes as input the current object which can be referred to as `this?` in the post-conditions. The result of the method is referred to as `result!`. It is possible to provide additional input parameters in the method declaration. The scope explicits which schemas will be read and/or modified by the operation. Here the constraint is

```
\Xi model
```

which means that the operation has read access (`\Xi`) to the whole model.

The post-conditions are stereotyped constraints. Here the post-condition includes most of the specification of the operation, written as follows:

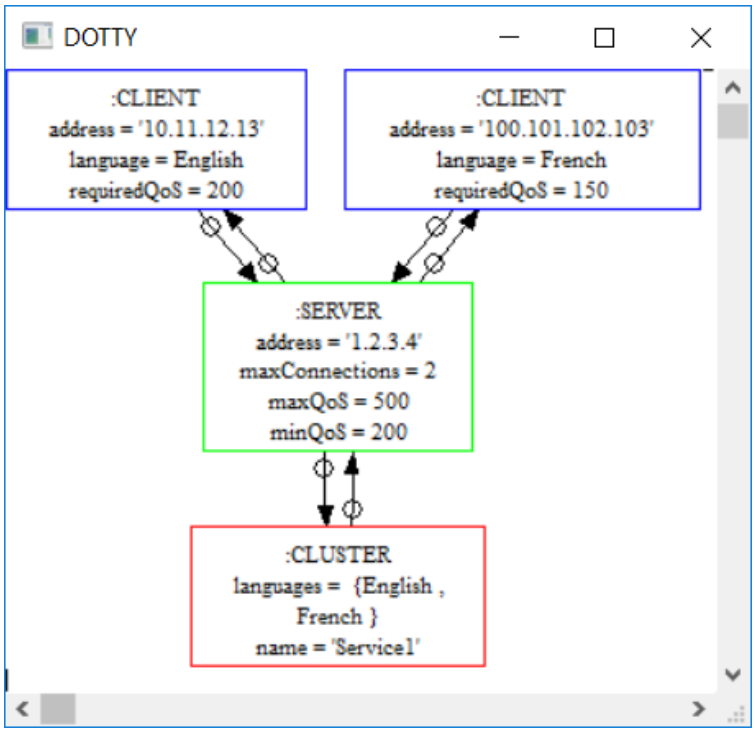
```
serverOfClient(this?) \in \dom clusterOfServer \\
 \# (serversOfCluster(clusterOfServer(serverOfClient(this?)))) > 1 \\
result! \in (serversOfCluster(clusterOfServer(serverOfClient(this?))))
              \setminus \{ serverOfClient(this?) \}\\
result!.minQoS \geq this?.requiredQoS
```

The operation returns another server of the cluster than the current one, with a sufficient QoS. The first line of the post-condition states that the server of the current object is member of a cluster. The second line expresses that this cluster includes more than one server. The third line expresses that the result is a server of the cluster different than the current server. Finally the fourth line expresses that the result has sufficient QoS.

4.2 Specification generation and animation

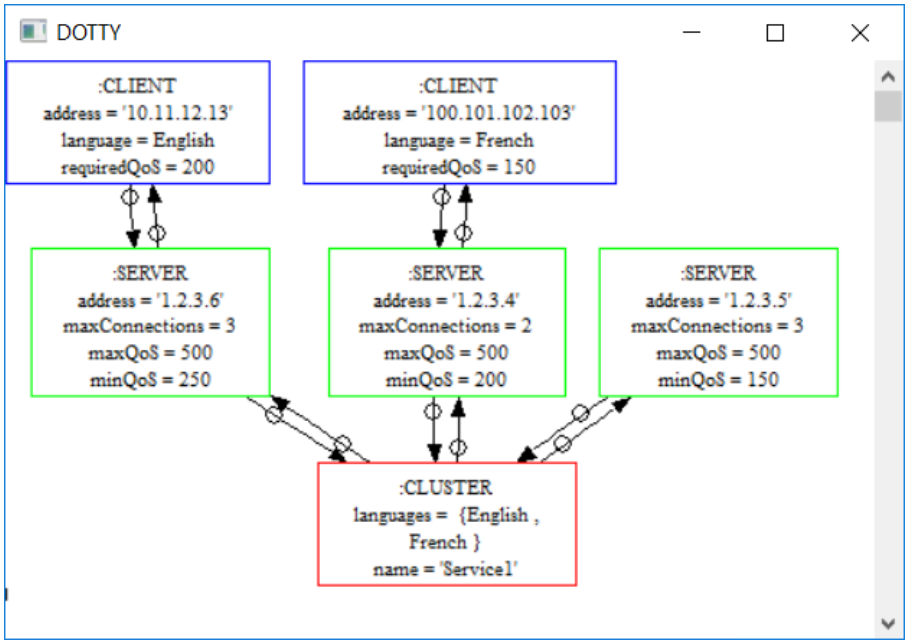
You can now use JazaGUI to call RoZ on the class diagram, and then to perform animations. Two scenarios are provided:

- The first scenario creates one cluster, one server and two clients, as shown in the following object diagram.



The last commands of the scenario lead to invariant violations.

- The second scenario creates three servers, two clients and one cluster.



4.3 Conclusion

This section has shown how a UML class diagram can be enhanced with Z annotations. These annotations express invariant properties, and operation specifications.

Chapter 5

Misc

5.1 Contacts

Please send your questions/reactions to Yves.Ledru@imag.fr.

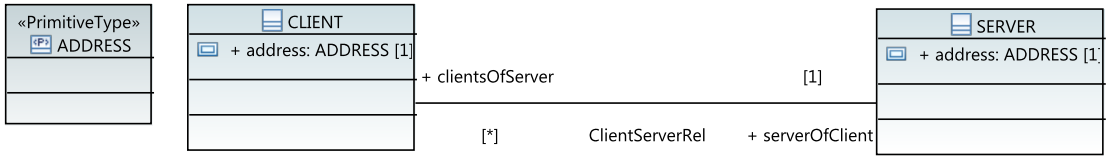
5.2 Acknowledgements

- The first version of RoZ has been mainly developed by Sophie Dupuy during her PhD thesis.
- JazaGui has been developed by Sébastien Laborie.
- The adaptation of Tobias to Jaza benefited from the active help of Mark Utting.
- Mohamed Amine Labiadh provided useful hints when redeveloping RoZ under eclipse.
- German Vega is now working on the next version of RoZ for eclipse and helped maintaining the current version.

Appendix A

Operations generated by RoZ

This document aims to list the basic operations automatically generated by RoZ. To illustrate this document, we rely on the following example:



At this stage, no constraint or Z annotation is attached to the diagram.

A.1 Translation of the data structures

The types and data structures are translated in Z as follows.

$[ADDRESS]$

CLIENT

$$address : ADDRESS$$

ClientExt

$$Client : \mathbb{F}\, CLIENT$$

SERVER

$$address : ADDRESS$$

ServerExt

$$Server : \mathbb{F}\, SERVER$$

ClientServerRel

$$ClientExt;\ ServerExt$$
$$clientsOfServer : SERVER \pfun \mathbb{F}\, CLIENT$$
$$serverOfClient : CLIENT \pfun SERVER$$

$$\mathrm{dom}\, clientsOfServer \subseteq Server$$
$$\bigcup(\mathrm{ran}\, clientsOfServer) = Client$$
$$clientsOfServer = \{x : \mathrm{ran}\, serverOfClient \bullet x \mapsto \{y : \mathrm{dom}\, serverOfClient \mid x = serverOfClient(y)\}\}$$
$$serverOfClient = \bigcup\{x : \mathrm{dom}\, clientsOfServer \bullet \{y : clientsOfServer(x) \bullet y \mapsto x\}\}$$

model

ClientExt
ServerExt
ClientServerRel

A.2 List of generated operations

The operations are prefixed with the name of their class. There are a few operations on the whole model (initialisation, invariant checking and coverage measurement).

A.2.1 Operations on the whole model

Model initialization

The initialization operation does not take input parameters and does not return any value, but initializes the state. Its name depends on the name of the model.

- *Init_model*

Checking all invariants

This operation checks all invariants of the model but does not modify the state. It is an operation which doe not modify the state and only succeeds if the state verifies all invariants.

- *Check_model*
- *CheckGlobalInvariant*

Coverage measurement

Coverage measurement checks that the current state of the model hits all sets and functions. It returns 3 values reporting how much is covered, what is the maximum value of the coverage in this model, and how much is not covered yet. The best measure is when *znotCovered*! is equal to 0.

- $Coverage\_model == [\ldots;\ zcoverage!, zcoverageMax!, znotCovered! : \mathbb{Z} \mid \ldots]$

A.2.2 Constructors and destructors

Every class comes with two constructors. The first one *Class_new* creates a single object and adds it to the extension of the class, taking as argument a binding of the type of the INTENSION. The second one *Class_newWithLinks* creates an object and links it to other objects. This second constructor is the one that should be used to construct correct models. Links are limited to the only mandatory ones. For example, the constructor of *CLIENT* takes as argument a client and the server linked to it, but the constructor of *SERVER* does not take a client in its arguments.

- $Client\_new == [\ldots;\ client? : CLIENT \mid \ldots]$
- $Client\_newWithLinks == [\ldots;\ client? : CLIENT;\ serverOfClient? : SERVER \mid \ldots]$
- $Server\_new == [\ldots;\ server? : SERVER \mid \ldots]$
- $Server\_newWithLinks == [\ldots;\ server? : SERVER \mid \ldots]$

Each class also features two destructors. One that destroys only the object, and the other one which also deletes the associated links. It is this second destructor that should be used to keep the consistency of the model. Both take the same arguments.

- $Client\_delete == [\ldots;\ this? : CLIENT \mid \ldots]$
- $Client\_delWithLinks == [\ldots;\ this? : CLIENT \mid \ldots]$
- $Server\_delete == [\ldots;\ this? : SERVER \mid \ldots]$
- $Server\_delWithLinks == [\ldots;\ this? : SERVER \mid \ldots]$

A.2.3 Getters and setters

There are several operations generated for getters and setters but the one that should be used is the one hereunder. The setter takes an object and a new value for its attribute as argument. The setter only takes an object, and returns a result of the type of the attribute.

- $Client\_getAddress == [\ldots;\ this? : CLIENT;\ result! : ADDRESS \mid \ldots]$
- $Client\_setAddress == [\ldots;\ this? : CLIENT;\ address? : ADDRESS \mid \ldots]$
- $Server\_getAddress == [\ldots;\ this? : SERVER;\ result! : ADDRESS \mid \ldots]$
- $Server\_setAddress == [\ldots;\ this? : SERVER;\ address? : ADDRESS \mid \ldots]$

A.2.4 Operations on associations

Each association has operations to create and delete links, and the operations are defined at both ends. It also includes getters for each of the roles.

- $Client\_linkServerOfClient == [\ldots;\ this? : CLIENT;\ s? : SERVER \mid \ldots]$
- $Client\_unlinkServerOfClient == [\ldots;\ this? : CLIENT;\ s? : SERVER \mid \ldots]$
- $Client\_getServerOfClient == [\ldots;\ this? : CLIENT;\ result! : SERVER \mid \ldots]$
- $Server\_linkClientsOfServer == [\ldots;\ this? : SERVER;\ c? : CLIENT \mid \ldots]$
- $Server\_unlinkClientsOfServer == [\ldots;\ this? : SERVER;\ c? : CLIENT \mid \ldots]$
- $Server\_getClientsOfServer == [\ldots;\ this? : SERVER;\ result! : \mathbb{F}\ CLIENT \mid \ldots]$

Appendix B

Code of the generated operations

In this appendix, we give the detailed code of the generated operations for the simple version of the client-server example.

ChangeClient

$$\Delta ClientExt$$
$$\Delta CLIENT$$
$$this? : CLIENT$$

$$this? \in Client$$
$$\theta CLIENT = this?$$
$$Client' = Client \setminus \{this?\} \cup \{\theta CLIENT'\}$$

SubstituteClientInRels

$$\Delta ClientServerRel$$
$$\Xi ServerExt$$
$$\Delta CLIENT$$

$$\exists newId : CLIENT \pfun CLIENT \mid newId = \{x : Client \setminus \{\theta CLIENT\} \bullet x \mapsto x\} \cup \{\theta CLIENT \mapsto \theta CLIENT'\} \bullet serverOfClient' = newId^{\sim} \fatsemi serverOfClient$$

Client_new

$$\Delta ClientExt$$
$$client? : CLIENT$$

$$Client' = Client \cup \{client?\}$$

Client_newWithLinks

$$\Delta ClientExt$$
$$\Delta ClientServerRel$$
$$\Xi ServerExt$$
$$client? : CLIENT$$
$$serverOfClient? : SERVER$$

$$Client' = Client \cup \{client?\}$$
$$serverOfClient? \in Server$$
$$serverOfClient' = serverOfClient \cup \{client? \mapsto serverOfClient?\}$$

Client_delete

$\Delta ClientExt$
$this? : CLIENT$

$Client' = Client \setminus \{this?\}$

Client_delWithLinks

$\Delta ClientExt$
$\Delta ClientServerRel$
$\Xi ServerExt$
$this? : CLIENT$

$Client' = Client \setminus \{this?\}$
$serverOfClient' = \{m : serverOfClient \mid first(m) \neq this?\}$

Client_setAddress_INT

$\Delta CLIENT$
$address? : ADDRESS$

$address' = address?$

$Client\_setAddress\_EXT == (ChangeClient \land Client\_setAddress\_INT) \setminus (address, address')$

$Client\_setAddress == (ChangeClient \land Client\_setAddress\_INT \land SubstituteClientInRels) \setminus (address, address')$

Client_getAddress_INT

$\Xi CLIENT$
$result! : ADDRESS$

$result! = address$

$Client\_getAddress\_EXT == (ChangeClient \land Client\_getAddress\_INT) \setminus (address, address')$

$Client\_getAddress == (ChangeClient \land Client\_getAddress\_INT \land SubstituteClientInRels) \setminus (address, address')$

Client_linkServerOfClient

$\Delta ClientServerRel$
$\Xi ClientExt$
$\Xi ServerExt$
$this? : CLIENT$
$s? : SERVER$

$s? \in Server$
$this? \in Client$
$serverOfClient' = serverOfClient \oplus \{this? \mapsto s?\}$

Client_unlinkServerOfClient

$\Delta ClientServerRel$
$\Xi ClientExt$
$\Xi ServerExt$
$this? : CLIENT$
$s? : SERVER$

$this? \in \operatorname{dom} serverOfClient$
$serverOfClient' = serverOfClient \setminus \{this? \mapsto s?\}$

Client_getServerOfClient

$\Xi model$
$this? : CLIENT$
$result! : SERVER$

$result! = serverOfClient(this?)$

ChangeServer

$\Delta ServerExt$
$\Delta SERVER$
$this? : SERVER$

$this? \in Server$
$\theta SERVER = this?$
$Server' = Server \setminus \{this?\} \cup \{\theta SERVER'\}$

SubstituteServerInRels

$\Delta ClientServerRel$
$\Xi ClientExt$
$\Delta SERVER$

$\exists newId : SERVER \pfun SERVER \mid newId = \{x : Server \setminus \{\theta SERVER\} \bullet$
$x \mapsto x\} \cup \{\theta SERVER \mapsto \theta SERVER'\} \bullet clientsOfServer' = newId^{\sim} \fatsemi clientsOfServer$

Server_new

$\Delta ServerExt$
$server? : SERVER$

$Server' = Server \cup \{server?\}$

Server_newWithLinks

$\Delta ServerExt$
$server? : SERVER$

$Server' = Server \cup \{server?\}$

Server_delete

$\Delta ServerExt$
$this? : SERVER$

$Server' = Server \setminus \{this?\}$

Server_delWithLinks

$\Delta ServerExt$
$\Delta ClientServerRel$
$\Xi ClientExt$
$this? : SERVER$

$Server' = Server \setminus \{this?\}$
$clientsOfServer' = \{m : clientsOfServer \mid first(m) \neq this?\}$

Server_setAddress_INT

$\Delta SERVER$
$address? : ADDRESS$

$address' = address?$

$Server\_setAddress\_EXT == (ChangeServer \land Server\_setAddress\_INT) \setminus (address, address')$

$Server\_setAddress == (ChangeServer \land Server\_setAddress\_INT \land SubstituteServerInRels) \setminus (address, address')$

Server_getAddress_INT

$\Xi SERVER$
$result! : ADDRESS$

$result! = address$

$Server\_getAddress\_EXT == (ChangeServer \land Server\_getAddress\_INT) \setminus (address, address')$

$Server\_getAddress == (ChangeServer \land Server\_getAddress\_INT \land SubstituteServerInRels) \setminus (address, address')$

Server_linkClientsOfServer

$\Delta ClientServerRel$
$\Xi ClientExt$
$\Xi ServerExt$
$this? : SERVER$
$c? : CLIENT$

$c? \in Client$
$this? \in Server$
$serverOfClient' = serverOfClient \oplus \{c? \mapsto this?\}$

Server_unlinkClientsOfServer

$\Delta ClientServerRel$
$\Xi ClientExt$
$\Xi ServerExt$
$this? : SERVER$
$c? : CLIENT$

$this? \in \mathrm{dom}\, clientsOfServer$
$serverOfClient' = serverOfClient \setminus \{c? \mapsto this?\}$

Server_getClientsOfServer

$\Xi model$
$this? : SERVER$
$result! : \mathbb{F}\, CLIENT$

$result! = clientsOfServer(this?)$

Init_model

$model'$

$Client' = \varnothing$
$Server' = \varnothing$
$clientsOfServer' = \varnothing$
$serverOfClient' = \varnothing$

Check_model

$\Xi model$

$CheckGlobalInvariant == Check\_model$

Coverage_model

$\Xi model$
$zcoverage!, zcoverageMax!, znotCovered! : \mathbb{Z}$

$zcoverage! = 0 + (\ IF\ Client \neq \varnothing\ THEN\ 1\ ELSE\ 0) +$
$(\ IF\ Server \neq \varnothing\ THEN\ 1\ ELSE\ 0) +$
$(\ IF\ clientsOfServer \neq \varnothing\ THEN\ 1\ ELSE\ 0) +$
$(\ IF\ serverOfClient \neq \varnothing\ THEN\ 1\ ELSE\ 0)$
$zcoverageMax! = 4$
$znotCovered! = zcoverageMax! - zcoverage!$