

# Case Study - VoIP Model Graph Transformation Rules

Mayur Bapodra, Reiko Heckel

Department of Computer Science, University of Leicester, UK  
mb294@mcs.le.ac.uk, reiko@mcs.le.ac.uk

## 1 Introduction

This report details a case study based on a very simple overview of peer-to-peer (P2P) connections over a voice over IP (VoIP) network such as that used for Skype. We present the concrete version of the model (with aggregating attributes already added) followed by the resulting abstract model.

Rule name abbreviations that have been used in accompanying research are given after each rule name.

## 2 Concrete Model

The VoIP network is constructed from Client and Super nodes, each servicing a single user, administrated by a central registry. Figure 1 shows the concrete type graph. *User* represents a physical user. *Client* represents a node that users attach to in order to make a call. Users can also attach to a *Super* node, but since Supers form the overall communication network via *ovl* edges, linking Clients to each other through this network, the user in question must meet a bandwidth threshold. In our model, we simplify this threshold by not explicitly measuring bandwidth, but instead controlling the likelihood that a user has the minimum bandwidth through stochastic parameters.

Our smallest start graph is shown in Figure 2. It consists of the registry node, one super node (with associated user) and six disconnected users. In the abstract model we aim to simplify the type graph by hiding User and Client types. There are therefore three aggregating attributes in this model to retain information. The Registry stores the number of clients not yet connected to the overlay network via a link edge to a Super (as *freeClients*). It also stores the number of users that have not yet turned on their VoIP program and therefore do not have an associated Client or Super node (as *offlineUsers*). Additionally, Supers store the number of clients linked to them. The OCL constraints giving their values in the concrete model are as follows:

```
context: Registry
inv: self.offlineUsers = User.allInstances -> count (u:User |
    Node.allInstances -> forall(n:Node | n.usr ≠ u))
```

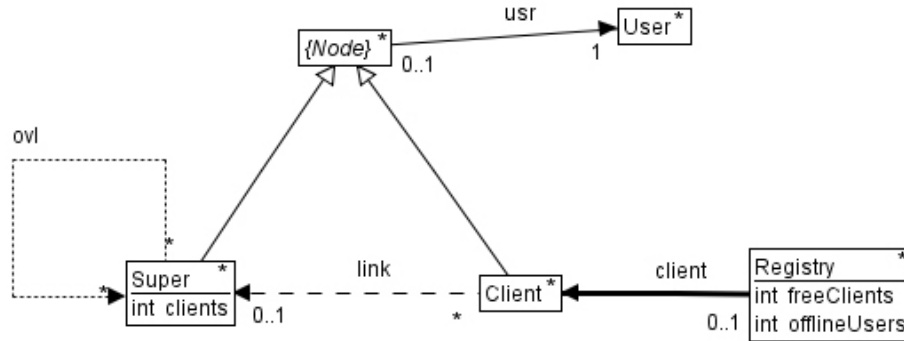


Fig. 1. Concrete type graph

```

context: Registry
inv: self.freeClients = Client.allInstances -> count (c:Client |
    c.link -> isEmpty())
  
```

```

context: Super
inv: self.clients = Client.allInstances ->
    count (c : Client | c.link = self)
  
```

The behavioural semantics of the model are explained through the GT rules. Figure 3 shows the creation of a new client when a user connects to the VoIP program. The NAC shows that the user must not already be connected. The new client is connected at the registry. The condition on the *offlineUsers* aggregating attribute is naturally redundant since we have the user node and the NAC at the concrete level.

Figure 4 illustrates a user wishing to communicate with another via the overlay network set up between Super nodes. The client must not have an existing link to a super node, and the super it connects to must have fewer than five clients. This is considered the maximum shareable bandwidth before a super node is overloaded. Note that ideally, although redundant, a NAC should also be present to represent this condition, but we omit it and leave the condition on the aggregating attribute only for simplicity.

Figure 5 shows the promotion of a client to a super when it is connected to an overloaded super node. This reduces some of the pressure on the overused super node.

Figure 6 depicts the rule that is analogous to the creation of a new client, except that the user has sufficient bandwidth to support a super node. The super node is immediately connected to the overlay network via an existing super node.

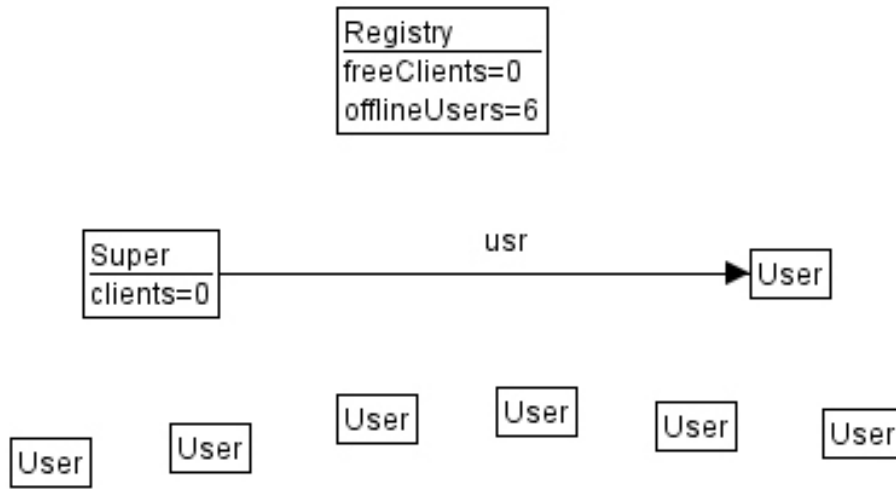


Fig. 2. Concrete start graph

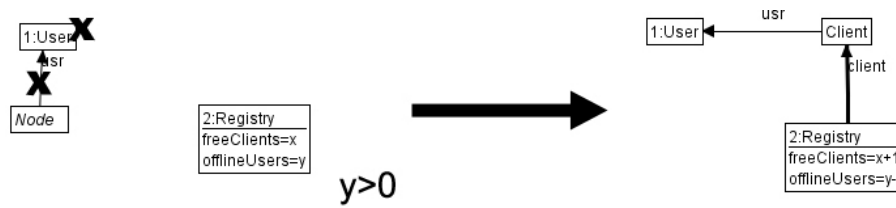


Fig. 3. New client, concrete GT rule (NC)

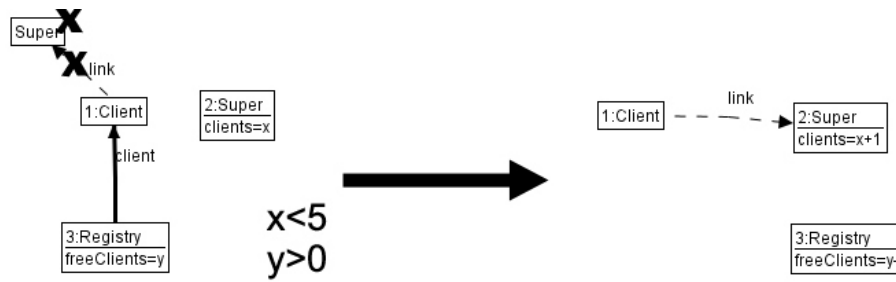
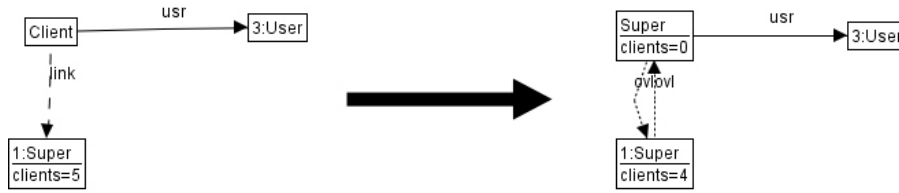
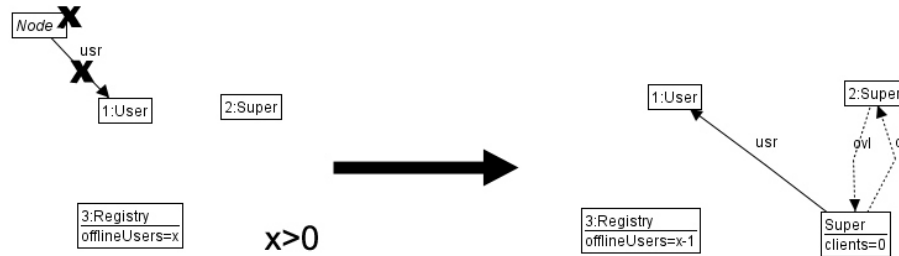


Fig. 4. Link client, concrete GT rule (LC)

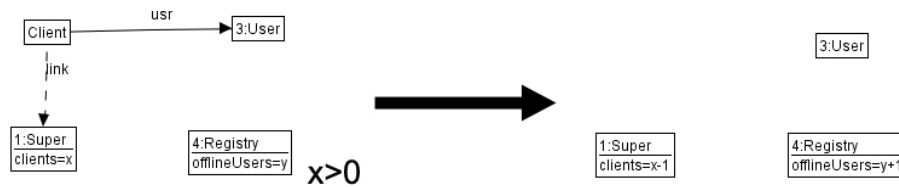


**Fig. 5.** Promote client, concrete GT rule (*PC*)



**Fig. 6.** New super, concrete GT rule (*NS*)

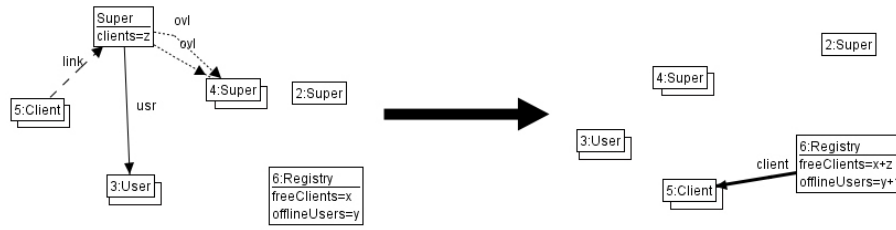
Figure 7 shows the termination of a client while it is connected to the overlay network (i.e., during a call). The rule represents the shutting down of the VoIP program by a user.



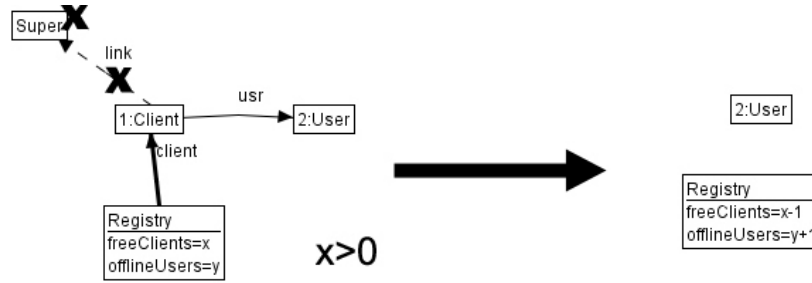
**Fig. 7.** Terminate linked client, concrete GT rule (*TL*)

The rule in Figure 8 represents the shutting down of the VoIP program by a user that is connected directly through a super node. This is only allowed if there is at least one other super node present. To prevent violation of the dangling condition, we must explicitly represent all other graph vertices the deleted node is connected to. Furthermore, all clients that were connected to the deleted super node are returned to the registry. Through the deletion of a super node by this rule, the overlay network may become disconnected so that some clients may become unreachable from others.

Figure 9 depicts the loss of an unlinked client, i.e., a user switching off the VoIP program without being in a call.

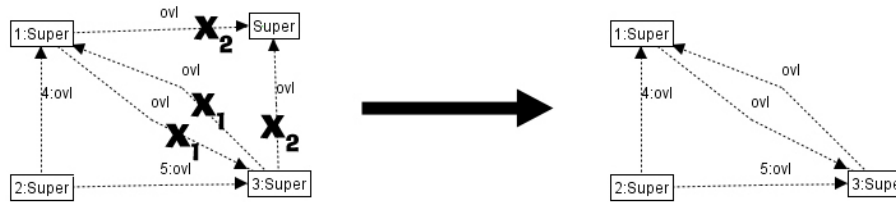


**Fig. 8.** Terminate super, concrete GT rule (*TS*)



**Fig. 9.** Terminate unlinked client, concrete GT rule (*TU*)

Finally, Figure 10 describes the creation of redundant links between supers in order to make the overlay network more robust. Client pairs are less susceptible to becoming disconnected when a super is terminated if there is more than one path between the supers to which they are connected. Note that the rule has two NACs: one stating that there should be no existing overlay connection between the supers that are to be newly linked, and secondly, that there is no alternate pathway between these supers via a fourth super node.



**Fig. 10.** Create shortcut, concrete GT rule (*CS*)

The global behaviour of interest in this model is the proportion of total clients at any time that are connected to the overlay network, and the proportion of total super node pairs that are unreachable from each other (i.e., there is no transitive closure of the *ovl* edge between them).

For the former, we create a probe rule that searches matches for Client nodes, and another that searches matches for Client nodes linked to Super nodes, dividing the second by the first to find the measure we require. This measurement is taken after every step in the simulation and we use the average value over each of these steps as our result.

Transitive closure measurements require modifications to the standard VIATRA2 installation [1]. A description of the necessary steps are outlined by the authors in an online addendum to [1] at <http://viatra.inf.mit.bme.hu/grats>. We specify the pattern between just two super nodes with a pair of oppositely directed *ovl* edges between them. The transitive closure version of the probe then looks for all pairs of super nodes between which there is a chain of *ovl* edges. A negation of this pattern then finds all such pairs between which there is no such transitive connection. We divide this number of pairs by the total number of super node pairs to find our measure. Just as with the connected client measure, we take the average of this value over all simulation steps.

Code Fragment 1.1 shows the VTCL specification implementing the entire behaviour of the concrete model, complete with probes and auxiliary patterns.

**Code Fragment 1.1.** VTCL Specification of Concrete Model

```

1 namespace p2p;
2
3
4 import DSM.metamodels.p2p_TG;
5 import datatypes;
6
7 @incremental
8 machine rules_and_constraints {
9
10 //////////////////////////////////////
11 // PATTERNS
12 //////////////////////////////////////
13
14     pattern UserConnected(U) = {
15         User(U);
16         P2PNode(N);
17         P2PNode.usr(U1, N, U);
18     }
19
20     pattern ClientLinked(C) = {
21         Client(C);
22         Super(S);
23         Client.link(L, C, S);
24     }
25
26     pattern Clients(C) = {
27         Client(C);
28     }
29
30     pattern SuperPairs(S1, S2) = {
31         Super(S1);
32         Super(S2);
33     }
34
35     shareable pattern SuperLinked(S1, S2) = {
36         Super(S1);
37         Super(S2);
38         Super.ovl(Ov1, S1, S2);
39         Super.ovl(Ov2, S2, S1);
40     }

```

```

41
42
43 @Incremental(reinterpret=transitiveClosure, ofPattern=SuperLinked)
44 pattern transitiveClosureOfSuperLinked(S1, S2) = {}
45
46
47
48 //////////////////////////////////////
49 // CONCRETE RULES
50 //////////////////////////////////////
51
52 gtrule NewClient() = {
53
54     precondition pattern lhs(U, OU, FC, R) = {
55         Registry(R);
56         Registry.OfflineUsers(OU) in R;
57         Registry.offlineUsers(Ou, R, OU);
58         Registry.FreeClients(FC) in R;
59         Registry.freeClients(Fc, R, FC);
60         User(U);
61         neg find UserConnected(U);
62     }
63
64     action {
65         let C=undef,
66             Cl=undef,
67             U1=undef,
68             Model=DSM.models.model
69         in seq {
70             new (Client(C) in Model);
71             rename(C, "cl_"+str.replaceAll(str.replaceAll(str.
72                 toLowerCase(name(C)), "_", ""), "un", ""));
73             new (P2PNode.usr(U1, C, U));
74             rename(U1, "usr");
75             new (Registry.client(Cl, R, C));
76             rename(Cl, "client_"+str.replaceAll(str.replaceAll(str.
77                 toLowerCase(name(Cl)), "_", ""), "un", ""));
78             setValue(OU, toString(toInteger(value(OU)) - 1));
79             setValue(FC, toString(toInteger(value(FC)) + 1));
80             println("GT RULE... NewClient applied to create Client "
81                 + name(C));
82         }
83     }
84 }
85
86 gtrule LinkClient() = {
87
88     precondition pattern lhs(FC, Cl1, C, S, S-C) = {
89         Super(S);
90         Super.Clients(S-C) in S;
91         Super.clients(S-Cx, S, S-C);
92         check (toInteger(value(S-C)) < 5);
93         Registry(R);
94         Registry.FreeClients(FC) in R;
95         Registry.freeClients(Fc, R, FC);
96         Client(C);
97         Registry.client(Cl1, R, C);
98         neg find ClientLinked(C);
99     }
100
101     action {
102         let L=undef
103         in seq {
104             delete(Cl1);
105             new(Client.link(L, C, S));
106             rename(L, "link");
107             setValue(S-C, toString(toInteger(value(S-C)) + 1));

```

```

106         setValue(FC, toString(toInteger(value(FC)) - 1));
107         println("GT RULE... LinkClient applied to link Client "
                + name(C));
108     }
109 }
110 }
111 }
112 }
113 gtrule PromoteClient() = {
114
115     precondition pattern lhs(S, C, U, L, U1, S_C) = {
116         Super(S);
117         Super.Clients(S_C) in S;
118         check (toInteger(value(S_C)) >= 5);
119         Super.clients(S_Cx, S, S_C);
120
121         Client(C);
122         Client.link(L, C, S);
123
124         User(U);
125         P2PNode.usr(U1, C, U);
126     }
127
128     action {
129         let S2=undef,
130             S_CNew=undef,
131             S_CNew_X=undef,
132             Ovl1=undef,
133             Ovl2=undef,
134             U2=undef,
135             Model=DSM.models.model
136         in seq {
137             print("GT RULE... PromoteClient applied to promote
                    Client " + name(C));
138             delete(L);
139             delete(U1);
140             delete(C);
141
142             new(Super(S2) in Model);
143             rename(S2, "super_"+str.replaceAll(str.replaceAll(str.
                    toLowerCase(name(S2)), "_", ""), "un", ""));
144
145             new(entity(S_CNew) in S2);
146             rename(S_CNew, "Clients");
147             setValue(S_CNew, 0);
148             new (instanceOf(S_CNew, ref("DSM.metamodels.p2p-TG.Super
                    .Clients")));
149             new(Super.clients(S_CNew_X, S2, S_CNew));
150             rename(S_CNew_X, "clients");
151
152             new(Super.ovl(Ovl1, S2, S));
153             rename(Ovl1, "ovl_"+str.replaceAll(str.replaceAll(str.
                    toLowerCase(name(Ovl1)), "_", ""), "un", ""));
154             new(Super.ovl(Ovl2, S, S2));
155             rename(Ovl2, "ovl_"+str.replaceAll(str.replaceAll(str.
                    toLowerCase(name(Ovl2)), "_", ""), "un", ""));
156             setValue(S_C, toString(toInteger(value(S_C)) - 1));
157
158             new(P2PNode.usr(U2, S2, U));
159             rename(U2, "usr");
160
161             println(" to Super " + name(S2));
162         }
163     }
164 }
165 }
166 }
167 gtrule NewSuper() = {

```



```

168
169 precondition pattern lhs(U, S, OU) = {
170     Registry(R);
171     Registry.OfflineUsers(OU) in R;
172     Registry.offlineUsers(Ou, R, OU);
173     User(U);
174     Super(S);
175     neg find UserConnected(U);
176 }
177
178 action {
179     let S2=undef,
180     S_CNew=undef,
181     S_CNew_X=undef,
182     Ovl1=undef,
183     Ovl2=undef,
184     Us1=undef,
185     Model=DSM.models.model
186     in seq {
187
188         new(Super(S2) in Model);
189         rename(S2,"super_"+str.replaceAll(str.replaceAll(str.
190             toLowerCase(name(S2)), "_", ""), "un", ""));
191
192         new(entity(S_CNew) in S2);
193         rename(S_CNew, "Clients");
194         setValue(S_CNew, 0);
195         new(instanceOf(S_CNew, ref("DSM.metamodels.p2p-TG.Super
196             .Clients")));
197         new(Super.clients(S_CNew_X, S2, S_CNew));
198         rename(S_CNew_X, "clients");
199
200         new(P2PNode.usr(Us1, S2, U));
201         rename(Us1, "usr");
202         new(Super.ovl(Ovl1, S2, S));
203         rename(Ovl1,"ovl_"+str.replaceAll(str.replaceAll(str.
204             toLowerCase(name(Ovl1)), "_", ""), "un", ""));
205         new(Super.ovl(Ovl2, S, S2));
206         rename(Ovl2,"ovl_"+str.replaceAll(str.replaceAll(str.
207             toLowerCase(name(Ovl2)), "_", ""), "un", ""));
208         setValue(OU, toString(toInteger(value(OU)) - 1));
209         println("GT RULE... NewSuper applied to create Super " +
210             name(S2) + " with Clients attribute value " + value(
211             S_CNew));
212     }
213 }
214
215 gtrule TerminateLinkedClient() = {
216     precondition pattern lhs(C, S, S_C, OU, U1, L) = {
217         Registry(R);
218         Registry.OfflineUsers(OU) in R;
219         Registry.offlineUsers(Ou, R, OU);
220         Super(S);
221         Super.Clients(S_C) in S;
222         Super.clients(S_Cx, S, S_C);
223         User(U);
224         Client(C);
225         P2PNode.usr(U1, C, U);
226         Client.link(L, C, S);
227     }
228
229     action {
230         delete(L);
231         delete(U1);
232         setValue(S_C, toString(toInteger(value(S_C)) - 1));

```

```

230         setValue(OU, toString(toInteger(value(OU)) + 1));
231         println("GT RULE... TerminateLinkedClient applied to
                terminate Client " + name(C));
232         delete(C);
233     }
234 }
235
236
237 gtrule TerminateUnlinkedClient() = {
238
239     precondition pattern lhs(C, OU, FC, U1, C1) = {
240         Registry(R);
241         Registry.OfflineUsers(OU) in R;
242         Registry.offlineUsers(Ou, R, OU);
243         Registry.FreeClients(FC) in R;
244         Registry.freeClients(Fc, R, FC);
245
246         User(U);
247         Client(C);
248         P2PNode.usr(U1, C, U);
249         Registry.client(C1, R, C);
250     }
251
252     action {
253         delete(C1);
254         delete(U1);
255         setValue(FC, toString(toInteger(value(FC)) - 1));
256         setValue(OU, toString(toInteger(value(OU)) + 1));
257         println("GT RULE... TerminateUnlinkedClient applied to
                terminate Client " + name(C));
258         delete(C);
259     }
260 }
261
262
263 gtrule TerminateSuper() = {
264
265     precondition pattern lhs(OU, FC, S2, R, S_C) = {
266         Registry(R);
267         Registry.OfflineUsers(OU) in R;
268         Registry.offlineUsers(Ou, R, OU);
269         Registry.FreeClients(FC) in R;
270         Registry.freeClients(Fc, R, FC);
271
272         Super(S1);
273         Super(S2);
274         Super.Clients(S_C);
275         Super.clients(S_CX, S2, S_C);
276     }
277
278     action {
279         setValue(FC, toString(toInteger(value(FC)) + toInteger(value
                (S_C))));
280         iterate choose with apply relinkClients(S2, R);
281         setValue(OU, toString(toInteger(value(OU)) + 1));
282         println("GT RULE... TerminateSuper applied to terminate
                Super " + name(S2));
283         delete(S2);
284     }
285 }
286
287
288 gtrule CreateShortcut() = {
289
290     precondition pattern lhs(S1, S2) = {
291         Super(S1);
292         Super(S2);
293         Super(S3);

```



```

358
359   gtrule Probe_ConnectedClients(inout C) = {
360     precondition find ClientLinked(C)
361   }
362
363   gtrule Probe_AllClients(inout C) = {
364     precondition find Clients(C)
365   }
366
367   gtrule Probe_Disconnected(inout S1, inout S2) = {
368     precondition pattern lhs(S1, S2) = {
369       Super(S1);
370       Super(S2);
371       neg find transitiveClosureOfSuperLinked(S1, S2);
372     }
373   }
374
375   gtrule Probe_AllSuperPairs(inout S1, inout S2) = {
376     precondition find SuperPairs(S1, S2)
377   }
378
379   gtrule Probe_Supers(inout S1) = {
380     precondition pattern lhs(S1) = {
381       Super(S1);
382     }
383   }
384 }
385 }

```

### 3 Abstract Model

In the abstract model, we retain only the *Super* and *Registry* types, with the inheritance of *Super* from *Node* as an inconsequential artefact. The abstract type graph is shown in Figure 11. The abstraction in this case does not reduce the number of rules, but with fewer graph elements in each rule and in each instance graph, there are fewer and smaller matches for each rule. Therefore, an increase in performance during stochastic simulation is still expected. The start graph is reduced to just two elements: the registry and the first super node, as shown in Figure 12.

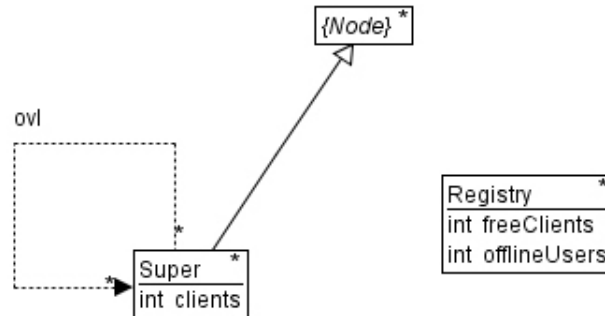
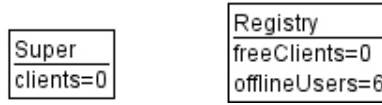


Fig. 11. Abstract type graph

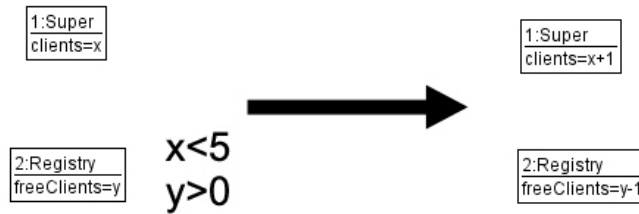


**Fig. 12.** Abstract start graph

Figures 13 to 19 show the projection of the concrete rules to the abstract type graph. Note that except for those of create shortcut, all NACs are lost but conditions on aggregating attributes still remain.



**Fig. 13.** New client, abstract GT rule ( $A_{NC}$ )



**Fig. 14.** Link client, abstract GT rule ( $A_{LC}$ )

The probes responsible for finding disconnected pairs of super nodes can still be used in the abstract model since the *Super* type is still present. However, to calculate the proportion of clients that are connected to the overlay network, the *clients* attribute of *Super* must be accessible along with the *freeClients* attribute of *Registry*. The sum of the *clients* attribute for all super nodes divided by this value plus the *freeClients* value gives us the measure we require.

Since the reading of attribute values is not currently supported by VIATRA2, a hard coded, model specific solution was implemented as a temporary measure. Probe rules were created to pass required attribute entities in the modelspace to the simulation engine. The hard coded solution recognized the probe by name and returns/sums the required attribute value for all matches of the probe rather than simply counting matches themselves. The solution will be incorporated into the stochastic simulation package of VIATRA2 once a naming convention/methodology for attribute value probes is decided upon.



Fig. 15. Promote client, abstract GT rule ( $A_{PC}$ )

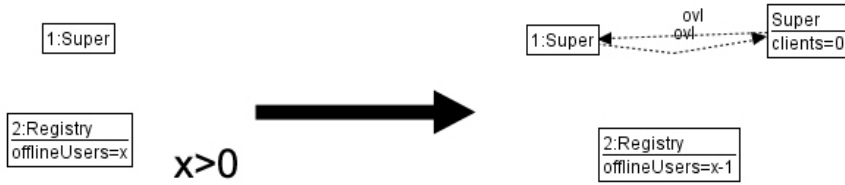


Fig. 16. New super, abstract GT rule ( $A_{NS}$ )



Fig. 17. Terminate linked client, abstract GT rule ( $A_{TL}$ )

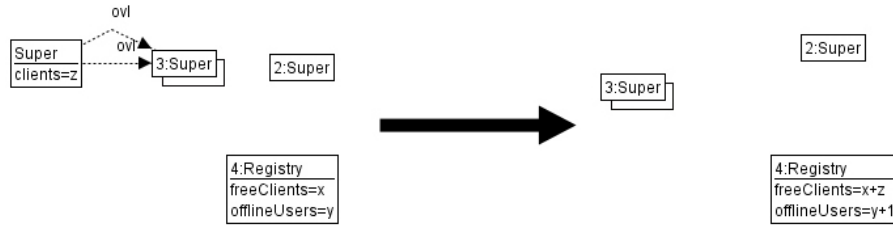


Fig. 18. Terminate super, abstract GT rule ( $A_{TS}$ )



Fig. 19. Terminate unlinked client, abstract GT rule ( $A_{TU}$ )

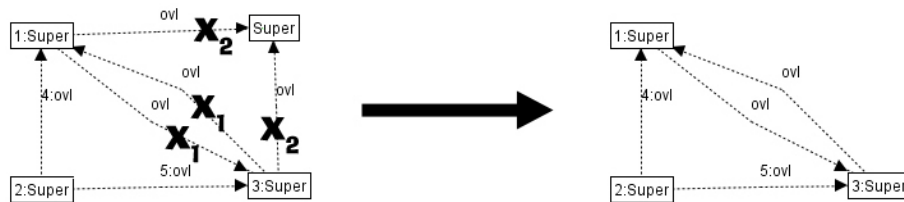


Fig. 20. Create shortcut, abstract GT rule ( $A_{CS}$ )

The portion of the *rules.vtcl* file that defines these abstract rules is given in Code Fragment 1.2.

**Code Fragment 1.2.** VTCL Specification of Abstract Model

```

1
2 ///////////////////////////////////////////////////////////////////
3 // ABSTRACT RULES
4 ///////////////////////////////////////////////////////////////////
5
6 gtrule Abstract_NewClient() = {
7
8     precondition pattern lhs(OU, FC) = {
9         Registry(R);
10        Registry.OfflineUsers(OU) in R;
11        Registry.offlineUsers(Ou, R, OU);
12        Registry.FreeClients(FC) in R;
13        Registry.freeClients(Fc, R, FC);
14        check (toInteger(value(OU)) > 0);
15    }
16
17    action {
18        setValue(OU, toString(toInteger(value(OU)) - 1));
19        setValue(FC, toString(toInteger(value(FC)) + 1));
20        println("GT RULE... NewClient applied to create Client");
21    }
22 }
23
24
25 gtrule Abstract_LinkClient() = {
26
27     precondition pattern lhs(FC, S, S_C) = {
28         Registry(R);
29         Registry.FreeClients(FC) in R;
30         Registry.freeClients(Fc, R, FC);
31         check (toInteger(value(FC)) > 0);
32         Super(S);
33         Super.Clients(S_C) in S;
34         Super.clients(S_Cx, S, S_C);
35         check (toInteger(value(S_C)) < 5);
36    }
37
38    action {
39        setValue(S_C, toString(toInteger(value(S_C)) + 1));
40        setValue(FC, toString(toInteger(value(FC)) - 1));
41        println("GT RULE... LinkClient applied to link Super " +
42              name(S));
43    }
44 }
45
46 gtrule Abstract_PromoteClient() = {
47
48     precondition pattern lhs(S, S_C) = {
49         Super(S);
50         Super.Clients(S_C) in S;
51         Super.clients(S_Cx, S, S_C);
52         check (toInteger(value(S_C)) >= 5);
53    }
54
55    action {
56        let S2=undef,
57        S_CNew=undef,
58        S_CNew_X=undef,
59        Ov1=undef,
60        Ov2=undef,
61        Model=DSM.models.model
62        in seq {

```

```

63         new(Super(S2) in Model);
64         rename(S2,"super_"+str.replaceAll(str.replaceAll(str.
           toLowerCase(name(S2)), "_", ""), "un", ""));
65
66         new(entity(S_CNew) in S2);
67         rename(S_CNew, "Clients");
68         setValue(S_CNew, 0);
69         new (instanceOf(S_CNew, ref("DSM.metamodels.p2p-TG.Super
           .Clients")));
70         new(Super.clients(S_CNew_X, S2, S_CNew));
71         rename(S_CNew_X, "clients");
72
73         new(Super.ovl(Ov1, S2, S));
74         rename(Ov1,"ovl_"+str.replaceAll(str.replaceAll(str.
           toLowerCase(name(Ov1)), "_", ""), "un", ""));
75         new(Super.ovl(Ov2, S, S2));
76         rename(Ov2,"ovl_"+str.replaceAll(str.replaceAll(str.
           toLowerCase(name(Ov2)), "_", ""), "un", ""));
77
78         setValue(S_C, toString(toInteger(value(S_C)) - 1));
79
80         println("GT RULE... PromoteClient applied to new Super "
           + name(S2));
81     }
82 }
83 }
84
85
86 gtrule Abstract_NewSuper() = {
87
88     precondition pattern lhs(S, OU) = {
89         Registry(R);
90         Registry.OfflineUsers(OU) in R;
91         Registry.offlineUsers(Ou, R, OU);
92         check (toInteger(value(OU)) >0);
93         Super(S);
94     }
95
96     action {
97         let S2=undef,
98             S_CNew=undef,
99             S_CNew_X=undef,
100            Ov1=undef,
101            Ov2=undef,
102            Model=DSM.models.model
103         in seq {
104             new(Super(S2) in Model);
105             rename(S2,"super_"+str.replaceAll(str.replaceAll(str.
               toLowerCase(name(S2)), "_", ""), "un", ""));
106
107             new(entity(S_CNew) in S2);
108             rename(S_CNew, "Clients");
109             setValue(S_CNew, 0);
110             new (instanceOf(S_CNew, ref("DSM.metamodels.p2p-TG.Super
               .Clients")));
111             new(Super.clients(S_CNew_X, S2, S_CNew));
112             rename(S_CNew_X, "clients");
113
114             new(Super.ovl(Ov1, S2, S));
115             rename(Ov1,"ovl_"+str.replaceAll(str.replaceAll(str.
               toLowerCase(name(Ov1)), "_", ""), "un", ""));
116             new(Super.ovl(Ov2, S, S2));
117             rename(Ov2,"ovl_"+str.replaceAll(str.replaceAll(str.
               toLowerCase(name(Ov2)), "_", ""), "un", ""));
118             setValue(OU, toString(toInteger(value(OU)) - 1));
119             println("GT RULE... NewSuper applied to create Super "
               + name(S2));
120

```



```

121     }
122   }
123 }
124
125
126 gtrule Abstract_TerminateLinkedClient() = {
127
128   precondition pattern lhs(S, S_C, OU) = {
129     Registry(R);
130     Registry.OfflineUsers(OU) in R;
131     Registry.offlineUsers(Ou, R, OU);
132     Super(S);
133     Super.Clients(S_C) in S;
134     Super.clients(S_Cx, S, S_C);
135     check (toInteger(value(S_C)) > 0);
136   }
137
138   action {
139     setValue(S_C, toString(toInteger(value(S_C)) - 1));
140     setValue(OU, toString(toInteger(value(OU)) + 1));
141     println("GT RULE... TerminateLinkedClient applied to
142           terminate Client on Super" + name(S));
143   }
144 }
145
146 gtrule Abstract_TerminateUnlinkedClient() = {
147
148   precondition pattern lhs(OU, FC) = {
149     Registry(R);
150     Registry.OfflineUsers(OU) in R;
151     Registry.offlineUsers(Ou, R, OU);
152     Registry.FreeClients(FC) in R;
153     Registry.freeClients(Fc, R, FC);
154     check (toInteger(value(FC)) > 0);
155   }
156
157   action {
158     setValue(FC, toString(toInteger(value(FC)) - 1));
159     setValue(OU, toString(toInteger(value(OU)) + 1));
160     println("GT RULE... TerminateUnlinkedClient applied");
161   }
162 }
163
164
165 gtrule Abstract_TerminateSuper() = {
166
167   precondition pattern lhs(OU, FC, S2, R, S_C) = {
168     Registry(R);
169     Registry.OfflineUsers(OU) in R;
170     Registry.offlineUsers(Ou, R, OU);
171     Registry.FreeClients(FC) in R;
172     Registry.freeClients(Fc, R, FC);
173
174     Super(S1);
175     Super(S2); // to delete
176     Super.Clients(S_C);
177     Super.clients(S_CX, S2, S_C);
178   }
179
180   action {
181     setValue(FC, toString(toInteger(value(FC)) + toInteger(value
182           (S_C))));
183     setValue(OU, toString(toInteger(value(OU)) + 1));
184     println("GT RULE... TerminateSuper applied to terminate
185           Super " + name(S2));
186     delete(S2);

```

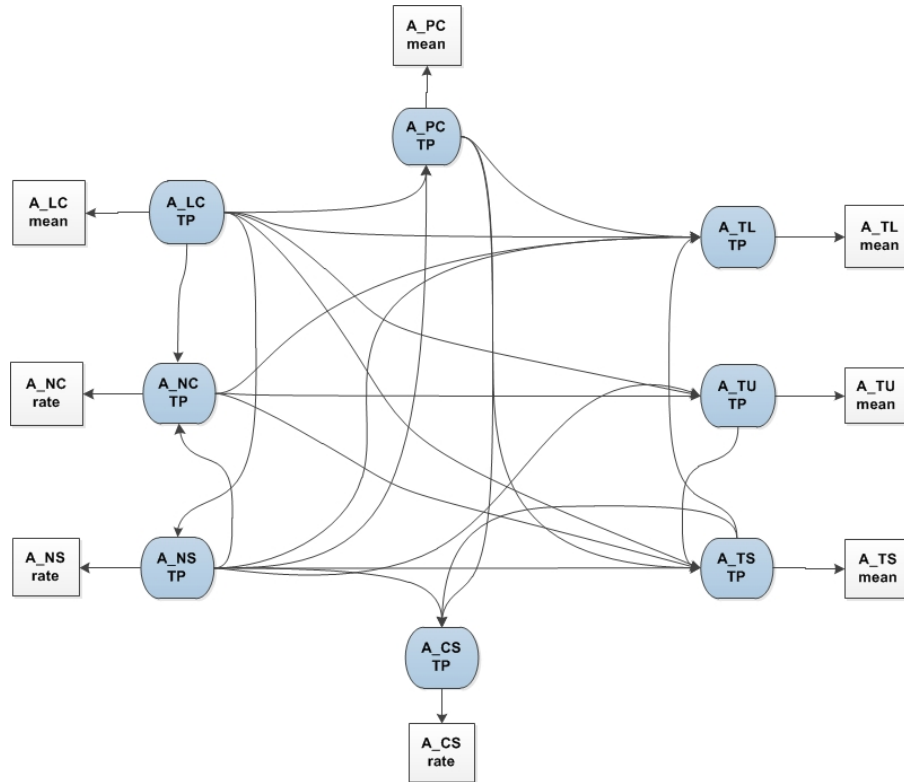
```

186     }
187   }
188
189
190   gtrule Abstract_CreateShortcut() = {
191
192     precondition pattern lhs(S1, S2) = {
193       Super(S1);
194       Super(S2);
195       Super(S3);
196       Super.ovl(O1, S3, S1);
197       Super.ovl(O2, S3, S2);
198       neg find SupersConnected(S1, S2);
199       neg find SupersBypass(S1, S2, S3);
200     }
201
202     action {
203       let Ovl1=undef,
204           Ovl2=undef
205       in seq {
206         new(Super.ovl(Ovl1, S1, S2));
207         rename(Ovl1,"ovl_"+str.replaceAll(str.replaceAll(str.
208           toLowerCase(name(Ovl1)), "_", ""), "un", ""));
209         new(Super.ovl(Ovl2, S2, S1));
210         rename(Ovl2,"ovl_"+str.replaceAll(str.replaceAll(str.
211           toLowerCase(name(Ovl2)), "_", ""), "un", ""));
212       }
213     }
214
215   }
216
217   //////////////////////////////////////
218   // PROBE RULES
219   //////////////////////////////////////
220
221   gtrule Probe_Disconnected(inout S1, inout S2) = {
222     precondition pattern lhs(S1, S2) = {
223       Super(S1);
224       Super(S2);
225       neg find transitiveClosureOfSuperLinked(S1, S2);
226     }
227   }
228
229   gtrule Probe_AllSuperPairs(inout S1, inout S2) = {
230     precondition find SuperPairs(S1, S2)
231   }
232
233   gtrule Probe_Supers(inout S1) = {
234     precondition pattern lhs(S1) = {
235       Super(S1);
236     }
237   }
238
239   gtrule Probe_AttributeFreeClients(inout FC) = {
240     precondition pattern lhs(FC) = {
241       Registry(R);
242       Registry.FreeClients(FC) in R;
243       Registry.freeClients(Fc, R, FC);
244     }
245   }
246
247   gtrule Probe_AttributeConnectedClients(inout S-C) = {
248     precondition pattern lhs(S-C) = {
249       Super(S);
250       Super.Clients(S-C);
251       Super.clients(S.CX, S, S-C);
252     }
253   }

```

## 4 Bayesian Network

The dynamic incidence matrix produced for abstract rules over the concrete model (i.e., diagonal analysis) is given in Table 1. The arbitrary partial order on rules names was decided as  $LC > NS > NC > TU > PC > TS > TL > CS$ . The resulting Bayesian network is shown in Figure 21.



**Fig. 21.** BN generated for VoIP case study (TP abbreviates throughput)

Table 1. Dynamic incidence matrix for abstract rules over concrete model (aggregate of diagonal conflicts and dependencies)

Rule name	Applied Rule										
	Start	LC	NC	NS	PC	TL	TS	TU	CS		
A_LC	0	-1.44 (0.106)	1.26 (0.104)	0.423 (0.0353)	0.696 (0.51)	0.00384 (0.00511)	0.275 (0.0991)	-0.884 (0.144)	0		
A_NC	1.0 (0.0)	0	-0.186 (0.025)	-0.252 (0.0311)	0	0.213 (0.0338)	0.254 (0.031)	0.142 (0.0368)	0		
A_NS	1.0 (0.0)	0	-0.564 (0.086)	-0.2 (0.155)	0.536 (0.267)	0.713 (0.124)	0.205 (0.155)	0.337 (0.103)	0		
A_PC	0	0.0192 (0.00861)	0	0	-1.0 (0.0)	-0.00839 (0.00753)	-0.00026 (0.00115)	0	0		
A_TL	0	0.503 (0.0314)	0	0	0	-0.379 (0.0401)	-0.360 (0.0342)	0	0		
A_TS	0	0	0	1.18 (0.0273)	1.88 (0.171)	0	-1.19 (0.0278)	0	0		
A_TU	0	-0.494 (0.0314)	0.464 (0.0320)	0	0	0	0.239 (0.0304)	-0.378 (0.0512)	0		
A_CS	0	0	0	2.41 (0.141)	0.203 (0.323)	0	-0.521 (0.131)	0	-2.49 (0.133)		

## References

1. Bergmann, G., Ráth, I., Szabó, T., Torrini, P., Varró, D.: Incremental pattern matching for the efficient computation of transitive closure. In: Sixth International Conference on Graph Transformation. Bremen, Germany (09/2012 2012)