

# 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.

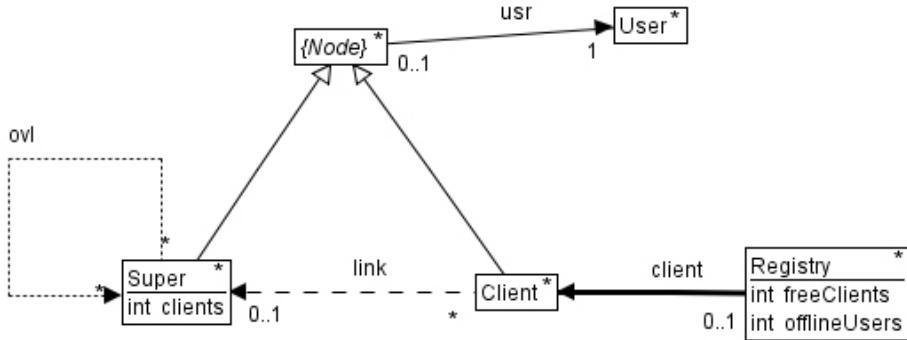
Rule name abbreviations that have been used in accompanying research are given after each rule name.

## 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)

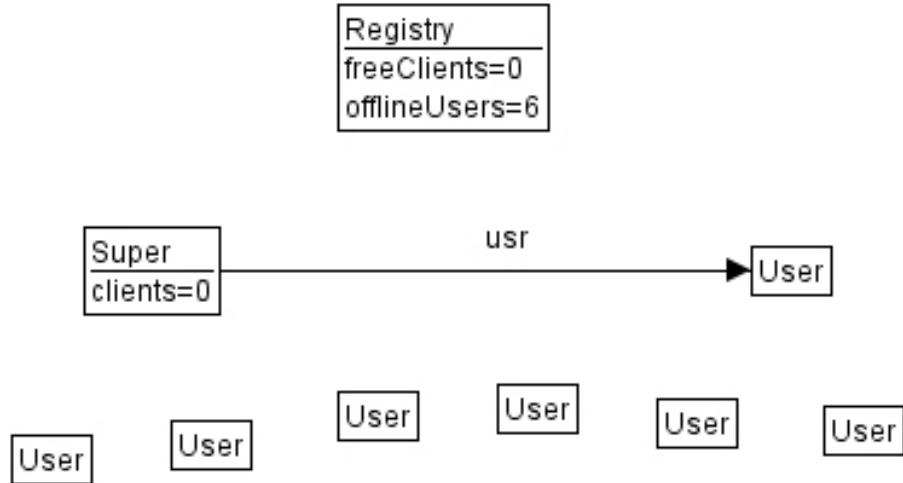
```

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.

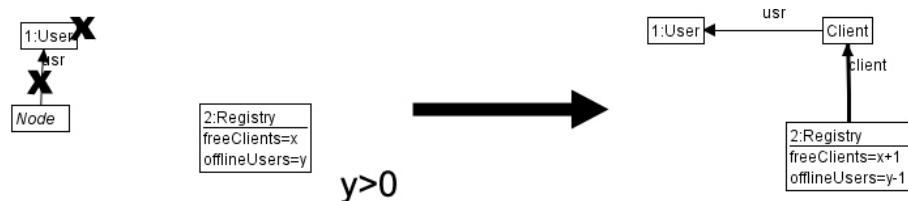
Figure 4 illustrates a user wishing to communicate with another via the overlay network set up between Super nodes. The client must not have an existing link to a super node, and the super it connects to must have fewer than five clients. This is considered the maximum shareable bandwidth before a super 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.

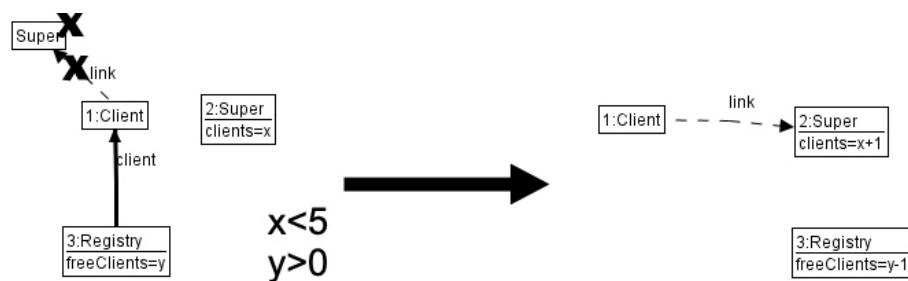
Figure 6 depicts the rule that is analogous to the creation of a new client, except that the user has sufficient bandwidth to support a super node. The super node is immediately connected to the overlay network via an existing super node.



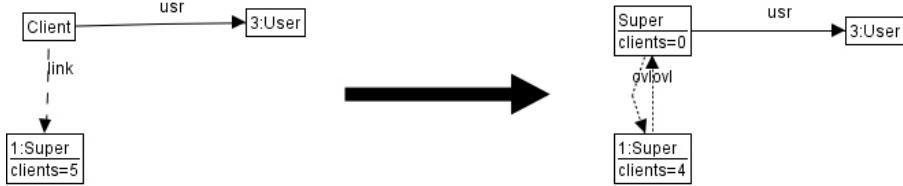
**Fig. 2.** Concrete start graph



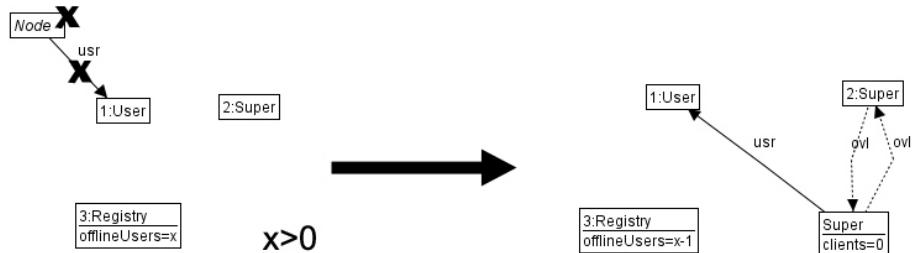
**Fig. 3.** New client, concrete GT rule (*NC*)



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

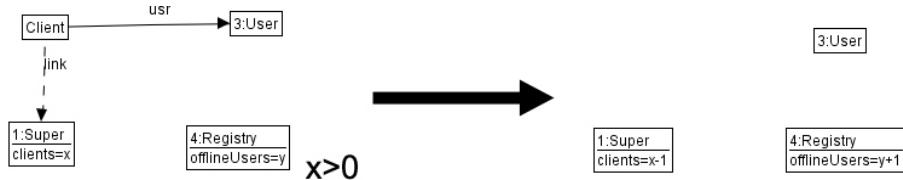


**Fig. 5.** Promote client, concrete GT rule (*PC*)



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

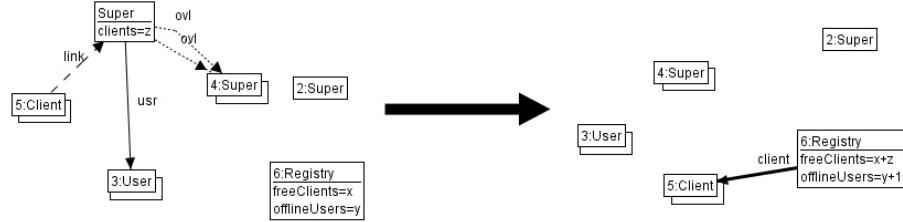
Figure 7 shows the termination of a client while it is connected to the overlay network (i.e., during a call). The rule represents the shutting down of the VoIP program by a user.



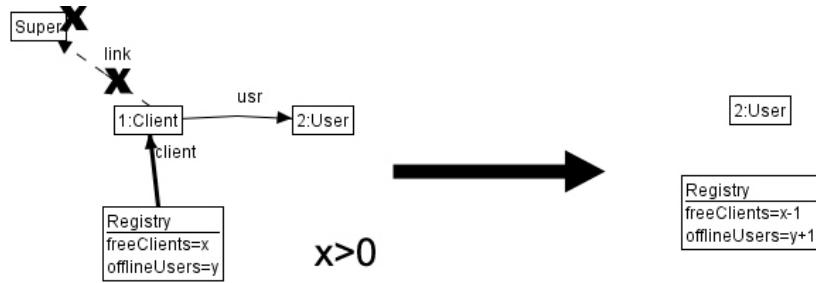
**Fig. 7.** Terminate linked client, concrete GT rule (*TL*)

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.

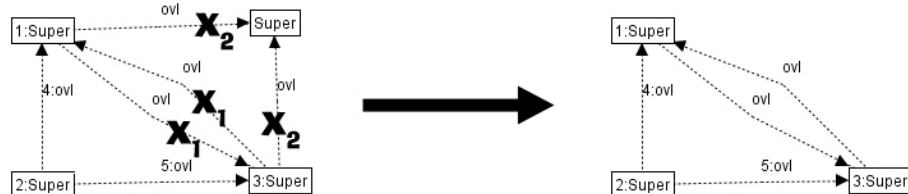


**Fig. 8.** Terminate super, concrete GT rule (*TS*)



**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 VIA-TRA2 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  import DSM.metamodels.p2p-TG;
4  import datatypes;
5
6  @incremental
7  machine rules_and_constraints {
8
9  //////////////////////////////////////////////////////////////////
10 // PATTERNNS
11 //////////////////////////////////////////////////////////////////
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 //////////////// CONCRETE RULES ///////////////////
49
50
51 gtrule NewClient() = {
52
53     precondition pattern lhs(U, OU, FC, R) = {
54         Registry(R);
55         Registry.OfflineUsers(OU) in R;
56         Registry.offlineUsers(Ou, R, OU);
57         Registry.FreeClients(FC) in R;
58         Registry.freeClients(Fc, R, FC);
59         User(U);
60         neg find UserConnected(U);
61     }
62
63
64     action {
65         let C=undef,
66             C1=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(C1, R, C));
76             rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
77                toLowerCase(name(C1)), "-", ""), "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     gtrule LinkClient() = {
86
87         precondition pattern lhs(FC, Cl1, C, S, S_C) = {
88             Super(S);
89             Super.Clients(S_C) in S;
90             Super.clients(S_Cx, S, S_C);
91             check (toInteger(value(S_C)) < 5);
92             Registry(R);
93             Registry.FreeClients(FC) in R;
94             Registry.freeClients(Fc, R, FC);
95             Client(C);
96             Registry.client(Cl1, R, C);
97             neg find ClientLinked(C);
98         }
99
100        action {
101            let L=undef
102            in seq {
103                delete(Cl1);
104                new(Client.link(L, C, S));
105                rename(L, "link");
106                setValue(S_C, toString(toInteger(value(S_C)) + 1));
107            }
108        }
109    }
110
111    gtrule ClientLinked() = {
112
113        precondition pattern lhs(L, C, S) = {
114            Client(L);
115            Client.clients(C) in S;
116            Client.clients(Cx, S, C);
117            check (toInteger(value(C)) < 5);
118            Registry(R);
119            Registry.FreeClients(L) in R;
120            Registry.freeClients(Fc, R, FC);
121            Client(C);
122            Registry.client(L, R, C);
123            neg find ClientLinked(C);
124        }
125
126        action {
127            let S=undef
128            in seq {
129                delete(C);
130                new(Client.link(L, C, S));
131                rename(L, "link");
132                setValue(C, toString(toInteger(value(C)) + 1));
133            }
134        }
135    }
136
137    gtrule ClientUnlinked() = {
138
139        precondition pattern lhs(L, C, S) = {
140            Client(L);
141            Client.clients(C) in S;
142            Client.clients(Cx, S, C);
143            check (toInteger(value(C)) < 5);
144            Registry(R);
145            Registry.FreeClients(L) in R;
146            Registry.freeClients(Fc, R, FC);
147            Client(C);
148            Registry.client(L, R, C);
149            neg find ClientLinked(C);
150        }
151
152        action {
153            let S=undef
154            in seq {
155                delete(C);
156                new(Client.link(L, C, S));
157                rename(L, "link");
158                setValue(C, toString(toInteger(value(C)) + 1));
159            }
160        }
161    }
162
163    gtrule ClientConnected() = {
164
165        precondition pattern lhs(U, OU, FC, R) = {
166            Registry(R);
167            Registry.OfflineUsers(OU) in R;
168            Registry.offlineUsers(Ou, R, OU);
169            Registry.FreeClients(FC) in R;
170            Registry.freeClients(Fc, R, FC);
171            User(U);
172            neg find UserConnected(U);
173        }
174
175        action {
176            let C=undef,
177                C1=undef,
178                U1=undef,
179                Model=DSM.models.model
180            in seq {
181                new (Client(C) in Model);
182                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
183                   toLowerCase(name(C)), "-", ""), "un", ""));
184                new(P2PNode.usr(U1, C, U));
185                rename(U1, "usr");
186                new (Registry.client(C1, R, C));
187                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
188                   toLowerCase(name(C1)), "-", ""), "un", ""));
189                setValue(OU, toString(toInteger(value(OU)) - 1));
190                setValue(FC, toString(toInteger(value(FC)) + 1));
191                println("GT RULE... ClientConnected applied to create Client "
192                    + name(C));
193            }
194        }
195    }
196
197    gtrule ClientDisconnected() = {
198
199        precondition pattern lhs(U, OU, FC, R) = {
200            Registry(R);
201            Registry.OfflineUsers(OU) in R;
202            Registry.offlineUsers(Ou, R, OU);
203            Registry.FreeClients(FC) in R;
204            Registry.freeClients(Fc, R, FC);
205            User(U);
206            neg find UserConnected(U);
207        }
208
209        action {
210            let C=undef,
211                C1=undef,
212                U1=undef,
213                Model=DSM.models.model
214            in seq {
215                new (Client(C) in Model);
216                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
217                   toLowerCase(name(C)), "-", ""), "un", ""));
218                new(P2PNode.usr(U1, C, U));
219                rename(U1, "usr");
220                new (Registry.client(C1, R, C));
221                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
222                   toLowerCase(name(C1)), "-", ""), "un", ""));
223                setValue(OU, toString(toInteger(value(OU)) + 1));
224                setValue(FC, toString(toInteger(value(FC)) - 1));
225                println("GT RULE... ClientDisconnected applied to create Client "
226                    + name(C));
227            }
228        }
229    }
230
231    gtrule ClientUnconnected() = {
232
233        precondition pattern lhs(U, OU, FC, R) = {
234            Registry(R);
235            Registry.OfflineUsers(OU) in R;
236            Registry.offlineUsers(Ou, R, OU);
237            Registry.FreeClients(FC) in R;
238            Registry.freeClients(Fc, R, FC);
239            User(U);
240            neg find UserConnected(U);
241        }
242
243        action {
244            let C=undef,
245                C1=undef,
246                U1=undef,
247                Model=DSM.models.model
248            in seq {
249                new (Client(C) in Model);
250                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
251                   toLowerCase(name(C)), "-", ""), "un", ""));
252                new(P2PNode.usr(U1, C, U));
253                rename(U1, "usr");
254                new (Registry.client(C1, R, C));
255                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
256                   toLowerCase(name(C1)), "-", ""), "un", ""));
257                setValue(OU, toString(toInteger(value(OU)) + 1));
258                setValue(FC, toString(toInteger(value(FC)) - 1));
259                println("GT RULE... ClientUnconnected applied to create Client "
260                    + name(C));
261            }
262        }
263    }
264
265    gtrule ClientReconnected() = {
266
267        precondition pattern lhs(U, OU, FC, R) = {
268            Registry(R);
269            Registry.OfflineUsers(OU) in R;
270            Registry.offlineUsers(Ou, R, OU);
271            Registry.FreeClients(FC) in R;
272            Registry.freeClients(Fc, R, FC);
273            User(U);
274            neg find UserConnected(U);
275        }
276
277        action {
278            let C=undef,
279                C1=undef,
280                U1=undef,
281                Model=DSM.models.model
282            in seq {
283                new (Client(C) in Model);
284                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
285                   toLowerCase(name(C)), "-", ""), "un", ""));
286                new(P2PNode.usr(U1, C, U));
287                rename(U1, "usr");
288                new (Registry.client(C1, R, C));
289                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
290                   toLowerCase(name(C1)), "-", ""), "un", ""));
291                setValue(OU, toString(toInteger(value(OU)) - 1));
292                setValue(FC, toString(toInteger(value(FC)) + 1));
293                println("GT RULE... ClientReconnected applied to create Client "
294                    + name(C));
295            }
296        }
297    }
298
299    gtrule ClientReunconnected() = {
300
301        precondition pattern lhs(U, OU, FC, R) = {
302            Registry(R);
303            Registry.OfflineUsers(OU) in R;
304            Registry.offlineUsers(Ou, R, OU);
305            Registry.FreeClients(FC) in R;
306            Registry.freeClients(Fc, R, FC);
307            User(U);
308            neg find UserConnected(U);
309        }
310
311        action {
312            let C=undef,
313                C1=undef,
314                U1=undef,
315                Model=DSM.models.model
316            in seq {
317                new (Client(C) in Model);
318                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
319                   toLowerCase(name(C)), "-", ""), "un", ""));
320                new(P2PNode.usr(U1, C, U));
321                rename(U1, "usr");
322                new (Registry.client(C1, R, C));
323                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
324                   toLowerCase(name(C1)), "-", ""), "un", ""));
325                setValue(OU, toString(toInteger(value(OU)) + 1));
326                setValue(FC, toString(toInteger(value(FC)) - 1));
327                println("GT RULE... ClientReunconnected applied to create Client "
328                    + name(C));
329            }
330        }
331    }
332
333    gtrule ClientRenamed() = {
334
335        precondition pattern lhs(U, OU, FC, R) = {
336            Registry(R);
337            Registry.OfflineUsers(OU) in R;
338            Registry.offlineUsers(Ou, R, OU);
339            Registry.FreeClients(FC) in R;
340            Registry.freeClients(Fc, R, FC);
341            User(U);
342            neg find UserConnected(U);
343        }
344
345        action {
346            let C=undef,
347                C1=undef,
348                U1=undef,
349                Model=DSM.models.model
350            in seq {
351                new (Client(C) in Model);
352                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
353                   toLowerCase(name(C)), "-", ""), "un", ""));
354                new(P2PNode.usr(U1, C, U));
355                rename(U1, "usr");
356                new (Registry.client(C1, R, C));
357                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
358                   toLowerCase(name(C1)), "-", ""), "un", ""));
359                setValue(OU, toString(toInteger(value(OU)) + 1));
360                setValue(FC, toString(toInteger(value(FC)) + 1));
361                println("GT RULE... ClientRenamed applied to create Client "
362                    + name(C));
363            }
364        }
365    }
366
367    gtrule ClientRenamed2() = {
368
369        precondition pattern lhs(U, OU, FC, R) = {
370            Registry(R);
371            Registry.OfflineUsers(OU) in R;
372            Registry.offlineUsers(Ou, R, OU);
373            Registry.FreeClients(FC) in R;
374            Registry.freeClients(Fc, R, FC);
375            User(U);
376            neg find UserConnected(U);
377        }
378
379        action {
380            let C=undef,
381                C1=undef,
382                U1=undef,
383                Model=DSM.models.model
384            in seq {
385                new (Client(C) in Model);
386                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
387                   toLowerCase(name(C)), "-", ""), "un", ""));
388                new(P2PNode.usr(U1, C, U));
389                rename(U1, "usr");
390                new (Registry.client(C1, R, C));
391                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
392                   toLowerCase(name(C1)), "-", ""), "un", ""));
393                setValue(OU, toString(toInteger(value(OU)) + 1));
394                setValue(FC, toString(toInteger(value(FC)) + 1));
395                println("GT RULE... ClientRenamed2 applied to create Client "
396                    + name(C));
397            }
398        }
399    }
400
401    gtrule ClientRenamed3() = {
402
403        precondition pattern lhs(U, OU, FC, R) = {
404            Registry(R);
405            Registry.OfflineUsers(OU) in R;
406            Registry.offlineUsers(Ou, R, OU);
407            Registry.FreeClients(FC) in R;
408            Registry.freeClients(Fc, R, FC);
409            User(U);
410            neg find UserConnected(U);
411        }
412
413        action {
414            let C=undef,
415                C1=undef,
416                U1=undef,
417                Model=DSM.models.model
418            in seq {
419                new (Client(C) in Model);
420                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
421                   toLowerCase(name(C)), "-", ""), "un", ""));
422                new(P2PNode.usr(U1, C, U));
423                rename(U1, "usr");
424                new (Registry.client(C1, R, C));
425                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
426                   toLowerCase(name(C1)), "-", ""), "un", ""));
427                setValue(OU, toString(toInteger(value(OU)) + 1));
428                setValue(FC, toString(toInteger(value(FC)) + 1));
429                println("GT RULE... ClientRenamed3 applied to create Client "
430                    + name(C));
431            }
432        }
433    }
434
435    gtrule ClientRenamed4() = {
436
437        precondition pattern lhs(U, OU, FC, R) = {
438            Registry(R);
439            Registry.OfflineUsers(OU) in R;
440            Registry.offlineUsers(Ou, R, OU);
441            Registry.FreeClients(FC) in R;
442            Registry.freeClients(Fc, R, FC);
443            User(U);
444            neg find UserConnected(U);
445        }
446
447        action {
448            let C=undef,
449                C1=undef,
450                U1=undef,
451                Model=DSM.models.model
452            in seq {
453                new (Client(C) in Model);
454                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
455                   toLowerCase(name(C)), "-", ""), "un", ""));
456                new(P2PNode.usr(U1, C, U));
457                rename(U1, "usr");
458                new (Registry.client(C1, R, C));
459                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
460                   toLowerCase(name(C1)), "-", ""), "un", ""));
461                setValue(OU, toString(toInteger(value(OU)) + 1));
462                setValue(FC, toString(toInteger(value(FC)) + 1));
463                println("GT RULE... ClientRenamed4 applied to create Client "
464                    + name(C));
465            }
466        }
467    }
468
469    gtrule ClientRenamed5() = {
470
471        precondition pattern lhs(U, OU, FC, R) = {
472            Registry(R);
473            Registry.OfflineUsers(OU) in R;
474            Registry.offlineUsers(Ou, R, OU);
475            Registry.FreeClients(FC) in R;
476            Registry.freeClients(Fc, R, FC);
477            User(U);
478            neg find UserConnected(U);
479        }
480
481        action {
482            let C=undef,
483                C1=undef,
484                U1=undef,
485                Model=DSM.models.model
486            in seq {
487                new (Client(C) in Model);
488                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
489                   toLowerCase(name(C)), "-", ""), "un", ""));
490                new(P2PNode.usr(U1, C, U));
491                rename(U1, "usr");
492                new (Registry.client(C1, R, C));
493                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
494                   toLowerCase(name(C1)), "-", ""), "un", ""));
495                setValue(OU, toString(toInteger(value(OU)) + 1));
496                setValue(FC, toString(toInteger(value(FC)) + 1));
497                println("GT RULE... ClientRenamed5 applied to create Client "
498                    + name(C));
499            }
500        }
501    }
502
503    gtrule ClientRenamed6() = {
504
505        precondition pattern lhs(U, OU, FC, R) = {
506            Registry(R);
507            Registry.OfflineUsers(OU) in R;
508            Registry.offlineUsers(Ou, R, OU);
509            Registry.FreeClients(FC) in R;
510            Registry.freeClients(Fc, R, FC);
511            User(U);
512            neg find UserConnected(U);
513        }
514
515        action {
516            let C=undef,
517                C1=undef,
518                U1=undef,
519                Model=DSM.models.model
520            in seq {
521                new (Client(C) in Model);
522                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
523                   toLowerCase(name(C)), "-", ""), "un", ""));
524                new(P2PNode.usr(U1, C, U));
525                rename(U1, "usr");
526                new (Registry.client(C1, R, C));
527                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
528                   toLowerCase(name(C1)), "-", ""), "un", ""));
529                setValue(OU, toString(toInteger(value(OU)) + 1));
530                setValue(FC, toString(toInteger(value(FC)) + 1));
531                println("GT RULE... ClientRenamed6 applied to create Client "
532                    + name(C));
533            }
534        }
535    }
536
537    gtrule ClientRenamed7() = {
538
539        precondition pattern lhs(U, OU, FC, R) = {
540            Registry(R);
541            Registry.OfflineUsers(OU) in R;
542            Registry.offlineUsers(Ou, R, OU);
543            Registry.FreeClients(FC) in R;
544            Registry.freeClients(Fc, R, FC);
545            User(U);
546            neg find UserConnected(U);
547        }
548
549        action {
550            let C=undef,
551                C1=undef,
552                U1=undef,
553                Model=DSM.models.model
554            in seq {
555                new (Client(C) in Model);
556                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
557                   toLowerCase(name(C)), "-", ""), "un", ""));
558                new(P2PNode.usr(U1, C, U));
559                rename(U1, "usr");
560                new (Registry.client(C1, R, C));
561                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
562                   toLowerCase(name(C1)), "-", ""), "un", ""));
563                setValue(OU, toString(toInteger(value(OU)) + 1));
564                setValue(FC, toString(toInteger(value(FC)) + 1));
565                println("GT RULE... ClientRenamed7 applied to create Client "
566                    + name(C));
567            }
568        }
569    }
570
571    gtrule ClientRenamed8() = {
572
573        precondition pattern lhs(U, OU, FC, R) = {
574            Registry(R);
575            Registry.OfflineUsers(OU) in R;
576            Registry.offlineUsers(Ou, R, OU);
577            Registry.FreeClients(FC) in R;
578            Registry.freeClients(Fc, R, FC);
579            User(U);
580            neg find UserConnected(U);
581        }
582
583        action {
584            let C=undef,
585                C1=undef,
586                U1=undef,
587                Model=DSM.models.model
588            in seq {
589                new (Client(C) in Model);
590                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
591                   toLowerCase(name(C)), "-", ""), "un", ""));
592                new(P2PNode.usr(U1, C, U));
593                rename(U1, "usr");
594                new (Registry.client(C1, R, C));
595                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
596                   toLowerCase(name(C1)), "-", ""), "un", ""));
597                setValue(OU, toString(toInteger(value(OU)) + 1));
598                setValue(FC, toString(toInteger(value(FC)) + 1));
599                println("GT RULE... ClientRenamed8 applied to create Client "
600                    + name(C));
601            }
602        }
603    }
604
605    gtrule ClientRenamed9() = {
606
607        precondition pattern lhs(U, OU, FC, R) = {
608            Registry(R);
609            Registry.OfflineUsers(OU) in R;
610            Registry.offlineUsers(Ou, R, OU);
611            Registry.FreeClients(FC) in R;
612            Registry.freeClients(Fc, R, FC);
613            User(U);
614            neg find UserConnected(U);
615        }
616
617        action {
618            let C=undef,
619                C1=undef,
620                U1=undef,
621                Model=DSM.models.model
622            in seq {
623                new (Client(C) in Model);
624                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
625                   toLowerCase(name(C)), "-", ""), "un", ""));
626                new(P2PNode.usr(U1, C, U));
627                rename(U1, "usr");
628                new (Registry.client(C1, R, C));
629                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
630                   toLowerCase(name(C1)), "-", ""), "un", ""));
631                setValue(OU, toString(toInteger(value(OU)) + 1));
632                setValue(FC, toString(toInteger(value(FC)) + 1));
633                println("GT RULE... ClientRenamed9 applied to create Client "
634                    + name(C));
635            }
636        }
637    }
638
639    gtrule ClientRenamed10() = {
640
641        precondition pattern lhs(U, OU, FC, R) = {
642            Registry(R);
643            Registry.OfflineUsers(OU) in R;
644            Registry.offlineUsers(Ou, R, OU);
645            Registry.FreeClients(FC) in R;
646            Registry.freeClients(Fc, R, FC);
647            User(U);
648            neg find UserConnected(U);
649        }
650
651        action {
652            let C=undef,
653                C1=undef,
654                U1=undef,
655                Model=DSM.models.model
656            in seq {
657                new (Client(C) in Model);
658                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
659                   toLowerCase(name(C)), "-", ""), "un", ""));
660                new(P2PNode.usr(U1, C, U));
661                rename(U1, "usr");
662                new (Registry.client(C1, R, C));
663                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
664                   toLowerCase(name(C1)), "-", ""), "un", ""));
665                setValue(OU, toString(toInteger(value(OU)) + 1));
666                setValue(FC, toString(toInteger(value(FC)) + 1));
667                println("GT RULE... ClientRenamed10 applied to create Client "
668                    + name(C));
669            }
670        }
671    }
672
673    gtrule ClientRenamed11() = {
674
675        precondition pattern lhs(U, OU, FC, R) = {
676            Registry(R);
677            Registry.OfflineUsers(OU) in R;
678            Registry.offlineUsers(Ou, R, OU);
679            Registry.FreeClients(FC) in R;
680            Registry.freeClients(Fc, R, FC);
681            User(U);
682            neg find UserConnected(U);
683        }
684
685        action {
686            let C=undef,
687                C1=undef,
688                U1=undef,
689                Model=DSM.models.model
690            in seq {
691                new (Client(C) in Model);
692                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
693                   toLowerCase(name(C)), "-", ""), "un", ""));
694                new(P2PNode.usr(U1, C, U));
695                rename(U1, "usr");
696                new (Registry.client(C1, R, C));
697                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
698                   toLowerCase(name(C1)), "-", ""), "un", ""));
699                setValue(OU, toString(toInteger(value(OU)) + 1));
700                setValue(FC, toString(toInteger(value(FC)) + 1));
701                println("GT RULE... ClientRenamed11 applied to create Client "
702                    + name(C));
703            }
704        }
705    }
706
707    gtrule ClientRenamed12() = {
708
709        precondition pattern lhs(U, OU, FC, R) = {
710            Registry(R);
711            Registry.OfflineUsers(OU) in R;
712            Registry.offlineUsers(Ou, R, OU);
713            Registry.FreeClients(FC) in R;
714            Registry.freeClients(Fc, R, FC);
715            User(U);
716            neg find UserConnected(U);
717        }
718
719        action {
720            let C=undef,
721                C1=undef,
722                U1=undef,
723                Model=DSM.models.model
724            in seq {
725                new (Client(C) in Model);
726                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
727                   toLowerCase(name(C)), "-", ""), "un", ""));
728                new(P2PNode.usr(U1, C, U));
729                rename(U1, "usr");
730                new (Registry.client(C1, R, C));
731                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
732                   toLowerCase(name(C1)), "-", ""), "un", ""));
733                setValue(OU, toString(toInteger(value(OU)) + 1));
734                setValue(FC, toString(toInteger(value(FC)) + 1));
735                println("GT RULE... ClientRenamed12 applied to create Client "
736                    + name(C));
737            }
738        }
739    }
740
741    gtrule ClientRenamed13() = {
742
743        precondition pattern lhs(U, OU, FC, R) = {
744            Registry(R);
745            Registry.OfflineUsers(OU) in R;
746            Registry.offlineUsers(Ou, R, OU);
747            Registry.FreeClients(FC) in R;
748            Registry.freeClients(Fc, R, FC);
749            User(U);
750            neg find UserConnected(U);
751        }
752
753        action {
754            let C=undef,
755                C1=undef,
756                U1=undef,
757                Model=DSM.models.model
758            in seq {
759                new (Client(C) in Model);
760                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
761                   toLowerCase(name(C)), "-", ""), "un", ""));
762                new(P2PNode.usr(U1, C, U));
763                rename(U1, "usr");
764                new (Registry.client(C1, R, C));
765                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
766                   toLowerCase(name(C1)), "-", ""), "un", ""));
767                setValue(OU, toString(toInteger(value(OU)) + 1));
768                setValue(FC, toString(toInteger(value(FC)) + 1));
769                println("GT RULE... ClientRenamed13 applied to create Client "
770                    + name(C));
771            }
772        }
773    }
774
775    gtrule ClientRenamed14() = {
776
777        precondition pattern lhs(U, OU, FC, R) = {
778            Registry(R);
779            Registry.OfflineUsers(OU) in R;
780            Registry.offlineUsers(Ou, R, OU);
781            Registry.FreeClients(FC) in R;
782            Registry.freeClients(Fc, R, FC);
783            User(U);
784            neg find UserConnected(U);
785        }
786
787        action {
788            let C=undef,
789                C1=undef,
790                U1=undef,
791                Model=DSM.models.model
792            in seq {
793                new (Client(C) in Model);
794                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
795                   toLowerCase(name(C)), "-", ""), "un", ""));
796                new(P2PNode.usr(U1, C, U));
797                rename(U1, "usr");
798                new (Registry.client(C1, R, C));
799                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
800                   toLowerCase(name(C1)), "-", ""), "un", ""));
801                setValue(OU, toString(toInteger(value(OU)) + 1));
802                setValue(FC, toString(toInteger(value(FC)) + 1));
803                println("GT RULE... ClientRenamed14 applied to create Client "
804                    + name(C));
805            }
806        }
807    }
808
809    gtrule ClientRenamed15() = {
810
811        precondition pattern lhs(U, OU, FC, R) = {
812            Registry(R);
813            Registry.OfflineUsers(OU) in R;
814            Registry.offlineUsers(Ou, R, OU);
815            Registry.FreeClients(FC) in R;
816            Registry.freeClients(Fc, R, FC);
817            User(U);
818            neg find UserConnected(U);
819        }
820
821        action {
822            let C=undef,
823                C1=undef,
824                U1=undef,
825                Model=DSM.models.model
826            in seq {
827                new (Client(C) in Model);
828                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
829                   toLowerCase(name(C)), "-", ""), "un", ""));
830                new(P2PNode.usr(U1, C, U));
831                rename(U1, "usr");
832                new (Registry.client(C1, R, C));
833                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
834                   toLowerCase(name(C1)), "-", ""), "un", ""));
835                setValue(OU, toString(toInteger(value(OU)) + 1));
836                setValue(FC, toString(toInteger(value(FC)) + 1));
837                println("GT RULE... ClientRenamed15 applied to create Client "
838                    + name(C));
839            }
840        }
841    }
842
843    gtrule ClientRenamed16() = {
844
845        precondition pattern lhs(U, OU, FC, R) = {
846            Registry(R);
847            Registry.OfflineUsers(OU) in R;
848            Registry.offlineUsers(Ou, R, OU);
849            Registry.FreeClients(FC) in R;
850            Registry.freeClients(Fc, R, FC);
851            User(U);
852            neg find UserConnected(U);
853        }
854
855        action {
856            let C=undef,
857                C1=undef,
858                U1=undef,
859                Model=DSM.models.model
860            in seq {
861                new (Client(C) in Model);
862                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
863                   toLowerCase(name(C)), "-", ""), "un", ""));
864                new(P2PNode.usr(U1, C, U));
865                rename(U1, "usr");
866                new (Registry.client(C1, R, C));
867                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
868                   toLowerCase(name(C1)), "-", ""), "un", ""));
869                setValue(OU, toString(toInteger(value(OU)) + 1));
870                setValue(FC, toString(toInteger(value(FC)) + 1));
871                println("GT RULE... ClientRenamed16 applied to create Client "
872                    + name(C));
873            }
874        }
875    }
876
877    gtrule ClientRenamed17() = {
878
879        precondition pattern lhs(U, OU, FC, R) = {
880            Registry(R);
881            Registry.OfflineUsers(OU) in R;
882            Registry.offlineUsers(Ou, R, OU);
883            Registry.FreeClients(FC) in R;
884            Registry.freeClients(Fc, R, FC);
885            User(U);
886            neg find UserConnected(U);
887        }
888
889        action {
890            let C=undef,
891                C1=undef,
892                U1=undef,
893                Model=DSM.models.model
894            in seq {
895                new (Client(C) in Model);
896                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
897                   toLowerCase(name(C)), "-", ""), "un", ""));
898                new(P2PNode.usr(U1, C, U));
899                rename(U1, "usr");
900                new (Registry.client(C1, R, C));
901                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
902                   toLowerCase(name(C1)), "-", ""), "un", ""));
903                setValue(OU, toString(toInteger(value(OU)) + 1));
904                setValue(FC, toString(toInteger(value(FC)) + 1));
905                println("GT RULE... ClientRenamed17 applied to create Client "
906                    + name(C));
907            }
908        }
909    }
910
911    gtrule ClientRenamed18() = {
912
913        precondition pattern lhs(U, OU, FC, R) = {
914            Registry(R);
915            Registry.OfflineUsers(OU) in R;
916            Registry.offlineUsers(Ou, R, OU);
917            Registry.FreeClients(FC) in R;
918            Registry.freeClients(Fc, R, FC);
919            User(U);
920            neg find UserConnected(U);
921        }
922
923        action {
924            let C=undef,
925                C1=undef,
926                U1=undef,
927                Model=DSM.models.model
928            in seq {
929                new (Client(C) in Model);
930                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
931                   toLowerCase(name(C)), "-", ""), "un", ""));
932                new(P2PNode.usr(U1, C, U));
933                rename(U1, "usr");
934                new (Registry.client(C1, R, C));
935                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
936                   toLowerCase(name(C1)), "-", ""), "un", ""));
937                setValue(OU, toString(toInteger(value(OU)) + 1));
938                setValue(FC, toString(toInteger(value(FC)) + 1));
939                println("GT RULE... ClientRenamed18 applied to create Client "
940                    + name(C));
941            }
942        }
943    }
944
945    gtrule ClientRenamed19() = {
946
947        precondition pattern lhs(U, OU, FC, R) = {
948            Registry(R);
949            Registry.OfflineUsers(OU) in R;
950            Registry.offlineUsers(Ou, R, OU);
951            Registry.FreeClients(FC) in R;
952            Registry.freeClients(Fc, R, FC);
953            User(U);
954            neg find UserConnected(U);
955        }
956
957        action {
958            let C=undef,
959                C1=undef,
960                U1=undef,
961                Model=DSM.models.model
962            in seq {
963                new (Client(C) in Model);
964                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
965                   toLowerCase(name(C)), "-", ""), "un", ""));
966                new(P2PNode.usr(U1, C, U));
967                rename(U1, "usr");
968                new (Registry.client(C1, R, C));
969                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
970                   toLowerCase(name(C1)), "-", ""), "un", ""));
971                setValue(OU, toString(toInteger(value(OU)) + 1));
972                setValue(FC, toString(toInteger(value(FC)) + 1));
973                println("GT RULE... ClientRenamed19 applied to create Client "
974                    + name(C));
975            }
976        }
977    }
978
979    gtrule ClientRenamed20() = {
980
981        precondition pattern lhs(U, OU, FC, R) = {
982            Registry(R);
983            Registry.OfflineUsers(OU) in R;
984            Registry.offlineUsers(Ou, R, OU);
985            Registry.FreeClients(FC) in R;
986            Registry.freeClients(Fc, R, FC);
987            User(U);
988            neg find UserConnected(U);
989        }
990
991        action {
992            let C=undef,
993                C1=undef,
994                U1=undef,
995                Model=DSM.models.model
996            in seq {
997                new (Client(C) in Model);
998                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
999                   toLowerCase(name(C)), "-", ""), "un", ""));
1000               new(P2PNode.usr(U1, C, U));
1001               rename(U1, "usr");
1002               new (Registry.client(C1, R, C));
1003               rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
1004                 toLowerCase(name(C1)), "-", ""), "un", ""));
1005               setValue(OU, toString(toInteger(value(OU)) + 1));
1006               setValue(FC, toString(toInteger(value(FC)) + 1));
1007               println("GT RULE... ClientRenamed20 applied to create Client "
1008                   + name(C));
1009            }
1010        }
1011    }
1012
1013    gtrule ClientRenamed21() = {
1014
1015        precondition pattern lhs(U, OU, FC, R) = {
1016            Registry(R);
1017            Registry.OfflineUsers(OU) in R;
1018            Registry.offlineUsers(Ou, R, OU);
1019            Registry.FreeClients(FC) in R;
1020            Registry.freeClients(Fc, R, FC);
1021            User(U);
1022            neg find UserConnected(U);
1023        }
1024
1025        action {
1026            let C=undef,
1027                C1=undef,
1028                U1=undef,
1029                Model=DSM.models.model
1030            in seq {
1031                new (Client(C) in Model);
1032                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
1033                   toLowerCase(name(C)), "-", ""), "un", ""));
1034                new(P2PNode.usr(U1, C, U));
1035                rename(U1, "usr");
1036                new (Registry.client(C1, R, C));
1037                rename(C1, "client_" + str.replaceAll(str.replaceAll(str.
1038                   toLowerCase(name(C1)), "-", ""), "un", ""));
1039                setValue(OU, toString(toInteger(value(OU)) + 1));
1040                setValue(FC, toString(toInteger(value(FC)) + 1));
1041                println("GT RULE... ClientRenamed21 applied to create Client "
1042                    + name(C));
1043            }
1044        }
1045    }
1046
1047    gtrule ClientRenamed22() = {
1048
1049        precondition pattern lhs(U, OU, FC, R) = {
1050            Registry(R);
1051            Registry.OfflineUsers(OU) in R;
1052            Registry.offlineUsers(Ou, R, OU);
1053            Registry.FreeClients(FC) in R;
1054            Registry.freeClients(Fc, R, FC);
1055            User(U);
1056            neg find UserConnected(U);
1057        }
1058
1059        action {
1060            let C=undef,
1061                C1=undef,
1062                U1=undef,
1063                Model=DSM.models.model
1064            in seq {
1065                new (Client(C) in Model);
1066                rename(C, "cl_" + str.replaceAll(str.replaceAll(str.
1067                   toLowerCase(name(C)), "-", ""), "un", ""));
1068                new(P2PNode.usr(U1, C, U));
1069                rename(U1, "
```

```

106         setValue(FC, toString(toInteger(value(FC)) - 1));
107         println("GT RULE... LinkClient applied to link Client "
108             + name(C));
109     }
110 }
111
112 gtrule PromoteClient() = {
113
114     precondition pattern lhs(S, C, U, L, U1, S_C) = {
115         Super(S);
116         Super.Clients(S_C) in S;
117         check (toInteger(value(S_C)) >= 5);
118         Super.clients(S_Cx, S, S_C);
119
120         Client(C);
121         Client.link(L, C, S);
122
123         User(U);
124         P2PNode.usr(U1, C, U);
125     }
126
127     action {
128         let S2=undef,
129         S_CNew=undef,
130         S_CNew_X=undef,
131         Ovl1=undef,
132         Ovl2=undef,
133         U2=undef,
134         Model=DSM.models.model
135         in seq {
136             print("GT RULE... PromoteClient applied to promote
137                 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.
144                 toLowerCase(name(S2)), "-", ""), "un", ""));
145
146             new(entity(S_CNew) in S2);
147             rename(S_CNew, "Clients");
148             setValue(S_CNew, 0);
149             new(instanceOf(S_CNew, ref("DSM.metamodels.p2p-TG.Super
150                 .Clients")));
151             new(Super.clients(S_CNew_X, S2, S_CNew));
152             rename(S_CNew_X, "clients");
153
154             new(Super.ovl(Ovl1, S2, S));
155             rename(Ovl1,"ovl_" + str.replaceAll(str.replaceAll(str.
156                 toLowerCase(name(Ovl1)), "-", ""), "un", ""));
157             new(Super.ovl(Ovl2, S, S2));
158             rename(Ovl2,"ovl_" + str.replaceAll(str.replaceAll(str.
159                 toLowerCase(name(Ovl2)), "-", ""), "un", ""));
160             setValue(S_C, toString(toInteger(value(S_C)) - 1));
161
162             new(P2PNode.usr(U2, S2, U));
163             rename(U2, "usr");
164
165             println(" to Super " + name(S2));
166         }
167     }
168
169     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 "+name(S2)+" with Clients attribute value "+value(S_CNew));
210     }
211 }
212
213 gtrule TerminateLinkedClient() = {
214     precondition pattern lhs(C, S, S_C, OU, U1, L) = {
215         Registry(R);
216         Registry.OfflineUsers(OU) in R;
217         Registry.offlineUsers(Ou, R, OU);
218         Super(S);
219         Super.Clients(S_C) in S;
220         Super.clients(S_Cx, S, S_C);
221         User(U);
222         Client(C);
223         P2PNode.usr(U1, C, U);
224         Client.link(L, C, S);
225     }
226     action {
227         delete(L);
228         delete(U1);
229         setValue(S_C, toString(toInteger(value(S_C))-1));

```

```

230     setValue(OU, toString(toInteger(value(OU)) + 1));
231     println("GT RULE... TerminateLinkedClient applied to
232         terminate Client " + name(C));
233     delete(C);
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
258             terminate Client " + name(C));
259         delete(C);
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
280             (S_C))));
281         iterate choose with apply relinkClients(S2, R);
282         setValue(OU, toString(toInteger(value(OU)) + 1));
283         println("GT RULE... TerminateSuper applied to terminate
284             Super " + name(S2));
285         delete(S2);
286     }
287
288 gtrule CreateShortcut() = {
289
290     precondition pattern lhs(S1, S2) = {
291         Super(S1);
292         Super(S2);
293         Super(S3);

```

```

294     Super.ovl(O1, S3, S1);
295     Super.ovl(O2, S3, S2);
296     neg find SupersConnected(S1, S2);
297     neg find SupersBypass(S1, S2, S3);
298 }
299
300 action {
301     let Ovl1=undef,
302     Ovl2=undef
303     in seq {
304         new(Super.ovl(Ovl1, S1, S2));
305         rename(Ovl1,"ovl_"+str.replaceAll(str.replaceAll(str.
306            toLowerCase(name(Ovl1)), "-", ""), "un", ""));
307         new(Super.ovl(Ovl2, S2, S1));
308         rename(Ovl2,"ovl_"+str.replaceAll(str.replaceAll(str.
309            toLowerCase(name(Ovl2)), "-", ""), "un", ""));
310     }
311 }
312
313 gtrule countClients(in S, in R, in Count) = {
314
315     precondition pattern lhs(S, L, C) = {
316
317         Super(S);
318         Client(C);
319         Client.link(L, C, S);
320         Registry(R);
321     }
322     action {
323         let C1=undef
324         in seq{
325             setValue(Count, toString(toInteger(value(Count)) + 1));
326             delete(L);
327             new(Registry.client(C1, R, C));
328             rename(C1, "client_"+str.replaceAll(str.replaceAll(str.
329                toLowerCase(name(C1)), "-", ""), "un", ""));
330         }
331     }
332 }
333
334 gtrule relinkClients(in S, in R) = {
335
336     precondition pattern lhs(S, L, C, R) = {
337
338         Super(S);
339         Client(C);
340         Client.link(L, C, S);
341         Registry(R);
342     }
343     action {
344         let C1=undef
345         in seq{
346             delete(L);
347             new(Registry.client(C1, R, C));
348             rename(C1, "client_"+str.replaceAll(str.replaceAll(str.
349                toLowerCase(name(C1)), "-", ""), "un", ""));
350         }
351     }
352
353
354
355 // PROBE RULES
356 /////////////////
357

```

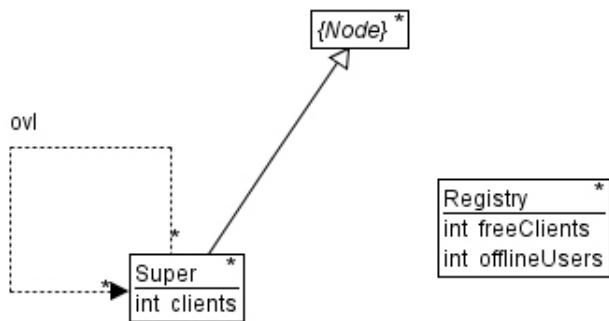
```

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

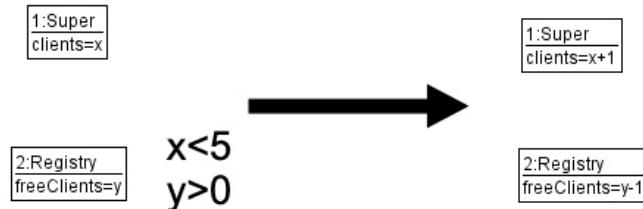


**Fig. 12.** Abstract start graph

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.



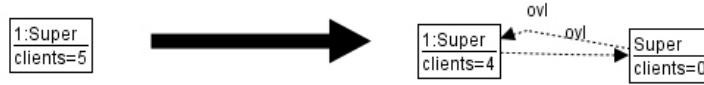
**Fig. 13.** New client, abstract GT rule ( $A\_NC$ )



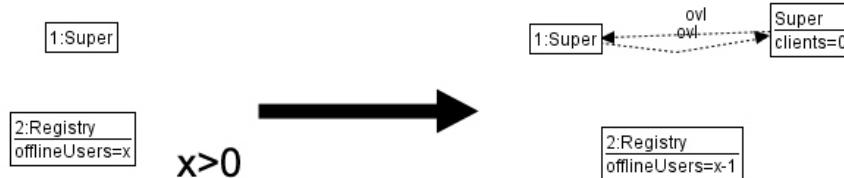
**Fig. 14.** Link client, abstract GT rule ( $A\_LC$ )

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.



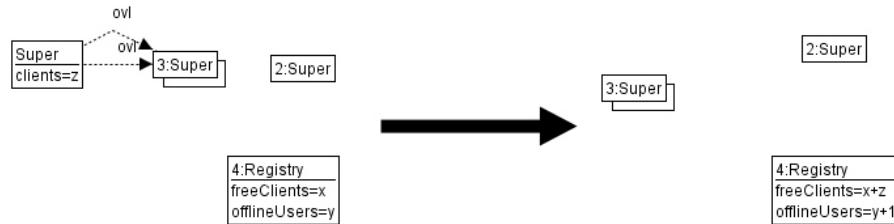
**Fig. 15.** Promote client, abstract GT rule (*A\_PPC*)



**Fig. 16.** New super, abstract GT rule (*A\_NS*)



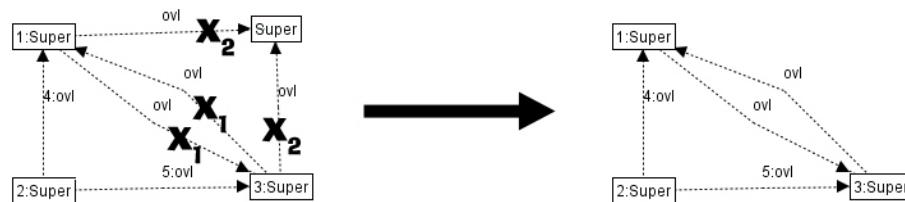
**Fig. 17.** Terminate linked client, abstract GT rule (*A\_TL*)



**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 // ABSTRACT RULES
3 //////////////////////////////////////////////////////////////////
4
5 gtrule Abstract_NewClient() = {
6
7     precondition pattern lhs(OU, FC) = {
8         Registry(R);
9         Registry.OfflineUsers(OU) in R;
10        Registry.offlineUsers(Ou, R, OU);
11        Registry.FreeClients(FC) in R;
12        Registry.freeClients(Fc, R, FC);
13        check (toInteger(value(OU)) > 0);
14    }
15
16    action {
17        setValue(OU, toString(toInteger(value(OU)) - 1));
18        setValue(FC, toString(toInteger(value(FC)) + 1));
19        println("GT RULE... NewClient applied to create Client");
20    }
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             Ovl1=undef,
60             Ovl2=undef,
61             Model=DSM.models.model
62         in seq {

```

```

63     new(Super(S2) in Model);
64     rename(S2,"super_"+str.replaceAll(str.replaceAll(str.
65        toLowerCase(name(S2)), "-", ""), "un", ""));
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
69         .Clients")));
70     new(Super.clients(S_CNew_X, S2, S_CNew));
71     rename(S_CNew_X, "clients");
72
73     new(Super.ovl(Ovl1, S2, S));
74     rename(Ovl1,"ovl_"+str.replaceAll(str.replaceAll(str.
74        toLowerCase(name(Ovl1)), "-", ""), "un", ""));
75     new(Super.ovl(Ovl2, S, S2));
76     rename(Ovl2,"ovl_"+str.replaceAll(str.replaceAll(str.
76        toLowerCase(name(Ovl2)), "-", ""), "un", ""));
77
78     setValue(S_C, toString(toInteger(value(S_C)) - 1));
79
80     println("GT RULE... PromoteClient applied to new Super "
80         + 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      Ovl1=undef,
101      Ovl2=undef,
102      Model=DSM.models.model
103      in seq {
104        new(Super(S2) in Model);
105        rename(S2,"super_"+str.replaceAll(str.replaceAll(str.
105         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
110         .Clients")));
111        new(Super.clients(S_CNew_X, S2, S_CNew));
112        rename(S_CNew_X, "clients");
113
114        new(Super.ovl(Ovl1, S2, S));
115        rename(Ovl1,"ovl_"+str.replaceAll(str.replaceAll(str.
115         toLowerCase(name(Ovl1)), "-", ""), "un", ""));
116        new(Super.ovl(Ovl2, S, S2));
117        rename(Ovl2,"ovl_"+str.replaceAll(str.replaceAll(str.
117         toLowerCase(name(Ovl2)), "-", ""), "un", ""));
118        setValue(OU, toString(toInteger(value(OU)) - 1));
119        println("GT RULE... NewSuper applied to create Super " +
120          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);

```

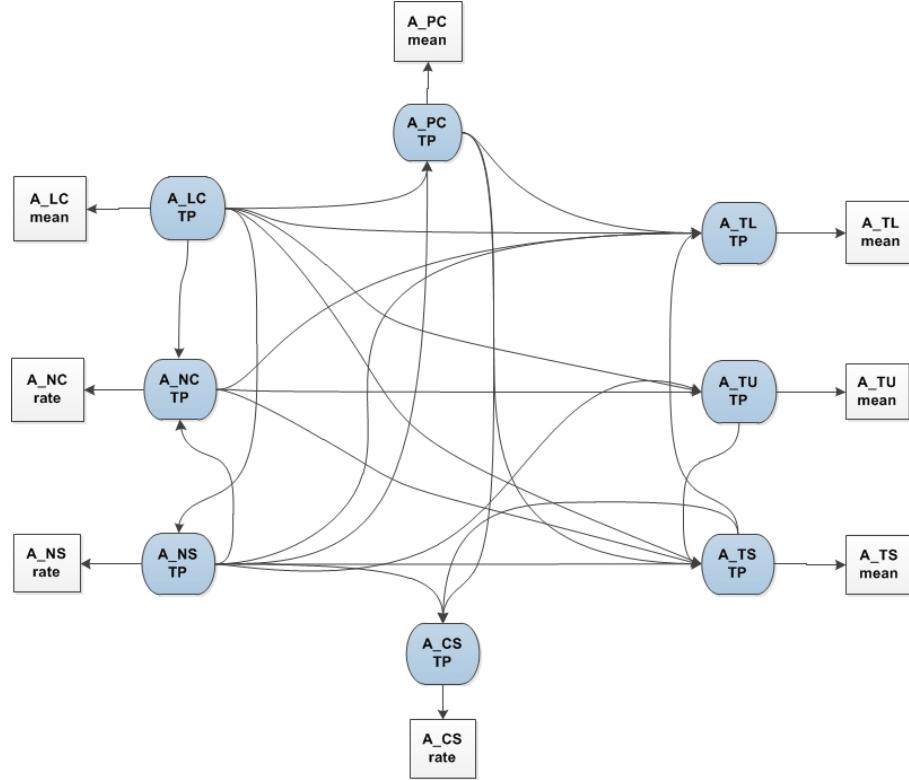
```

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 // PROBE RULES
216
217
218     gtrule Probe_Disconnected(inout S1, inout S2) = {
219         precondition pattern lhs(S1, S2) = {
220             Super(S1);
221             Super(S2);
222             neg find transitiveClosureOfSuperLinked(S1, S2);
223         }
224
225
226     gtrule Probe_AllSuperPairs(inout S1, inout S2) = {
227         precondition find SuperPairs(S1, S2)
228     }
229
230     gtrule Probe_Supers(inout S1) = {
231         precondition pattern lhs(S1) = {
232             Super(S1);
233         }
234     }
235
236     gtrule Probe_AttributeFreeClients(inout FC) = {
237         precondition pattern lhs(FC) = {
238             Registry(R);
239             Registry.FreeClients(FC) in R;
240             Registry.freeClients(Fc, R, FC);
241         }
242     }
243
244     gtrule Probe_AttributeConnectedClients(inout S_C) = {
245         precondition pattern lhs(S_C) = {
246             Super(S);
247             Super.Clients(S_C);
248             Super.clients(S_CX, S, S_C);
249         }
250     }

```

## 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	Start	Applied Rule							
		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 (0.1171)	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)