SOFAT : a Scenario Oracle and Formal Analysis Toolbox

Loïc Hélouët[1]

IRISA/INRIA, Campus de Beaulieu, 35042 Rennes cedex, France
loic.helouet@irisa.fr
http://www.irisa.fr/triskell/perso_pro/helouet/LHpage.html

Abstract. Ce document est une description sommaire de l'outil SOFAT, un environnement de manipulation formelle de scénarios.

1 Introduction

Cette dernière décennie, les scénarios ont connu un intérêt croissant. Des langages comme les Message Sequence Charts [10], les diagrammes de séquence d'UML [5], ou les live sequence charts [6] ont été développés. Même si chacune de ces notations comporte ses spécificités propres, elles partagent toutes une idée commune: les éléments de base de ces langages sont des chronogrammes. Dans ce chronogrammes, les entités du système sont représentées par des lignes verticales symbolisant l"écoulement du temps, et les communications sont représentées par des flèches. Ces modèles sont clairs et très simples, ce qui en fait des candidats idéaux pour la capture d'exigences dans les phases amont de la conception de systèmes distribués. L'exemple de la Figure 1 montre un Message Sequence Chart.

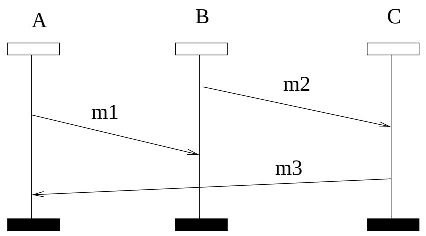

Fig. 1. Un exemple de bMSC

De toute évidence, ces dessins définissent des ordres partiels sur des ensembles d'événements dans des systèmes. Ces modèles non entrelacés sont très bien adaptés pour décrire la causalité et la concurrence dans un systèmes, mais ne permettent pas de définir des choix ou des boucles. Pour résoudre ce problème, la plupart des langages de scénarios propose des opérateurs de composition tels que la séquence, l'itération, et le choix. L'idée de composer ainsi des ordres partiels n'est pas nouvelle, et a déjà été proposée par Pratt [13]. Par contre, l'idée de décrire des comportements de systèmes distribués au moyens de compositions de motifs de communication clos plutôt que par des séquences communicantes est une idée récente. Cependant, les scénarios de base perdent rapidement leur simplicité lorsqu'ils sont composés. En effet, les événements concurrents ne se trouvent plus systématiquement dans le même diagramme, ce qui complique grandement la compréhension du comportement d'un système. De plus, la présence d'itérations implique que le système décrit n'est plus représentable par un ensemble fini d'états. Les opérateurs de composition donnent donc aux scénarios une expressivité énorme, et de nombreuses propriétés ne sont plus décidables que sur des sous-ensembles du langage. Cette complexité de la composition d'ordres partiels ne peut être traitée sans un ensemble d'outils permettant d'analyser et de simuler des descriptions données sous forme de scénarios, mais aussi d'identifier des classes de scénarios plus facilement manipulables.

Ce document décrit SOFAT, (Scenario Oracle and Formal Analysis Toolbox), un environnement pour l'analyse, la manipulation formelle de Message Sequence Charts. Ce document est organisé comme suit: la section 2 décrit les propriétés formelles détectées par l'outil, et la section 3 conclut. Une annexe fournit des exemples de manipulations formelles de MSC, et des copies d'écran d'une simulation.

2 Analyse

Les Message Sequence Charts sont un langage de scénarios standardisé par l'ITU. Ils sont composés de deux niveaux de spécification. Au plus bas niveau, les basic MSC(bMSC par la suite) décrivent des interactions simples entre des éléments d'un système appelés instances. Les bMSC ne permettent cependant pas d"exprimer des alternatives ou des itérations. Il sont donc composés au moyen d'un deuxième niveau du langage, appelé High-level Message Sequence Charts (HMSC par la suite). Les HMSC permettent la définition de comportements beaucoup plus élaborés et proposent des opérateurs de séquence, de choix, d'itération. Cependant, deux interprétations de la composition séquentielle existent: la composition séquentielle forte, et la composition séquentielle faible. Intuitivement, lorsque l'on définit une composition séquentielle M1;M2, la composition forte impose que tous les événements de M1 soient exécutés avant les événements de M2. La composition séquentielle faible n'impose un ordre que sur les événements situés sur une même instance dans M1 et M2. Ainsi, les deux diagrammes de la figure 2 représentent le même comportement. La composition séquentielle forte impose des synchronisations arbitraires qui peuvent ne pas être implémentables par un système distribué, et limite le pouvoir d'expression des MSC à celui des langages réguliers. La sémantique officielle des MSC suppose que la composition séquentielle de deux bMSC est une composition faible. cependant, la composition faible donne aux MSC un immense pouvoir d'expression. En fait, les MSC ont l pouvoir d'expression des parties rationnelles [3], ce qui rend leur analyse très difficile. Par exemple, la vacuité de l'intersection du langage de deux MSC, ou l'inclusion d'un langage de MSC dans un autre ne sont pas des problèmes décidables. Heureusement, la plupart des résultats d'indécidabilité sur les MSC sont dus à des cas pathologiques du langage, et correspondent à des propriétés structurelles de MSC. Ces cas peuvent donc être identifiés. De récents travaux ont ainsi identifié des sous-classes de MSC autorisant un certain nombre de manipulations formelles. Ainsi, avant de tenter de faire du model-checking, de la comparaison, ou toute autre manipulation formelle de MSCs, il convient de s'assurer de l'appartenance des descriptions utilisées à une classe du langage autorisant cette manipulation formelle. L'outil SOFAT permet la détection des classes de MSC suivantes:

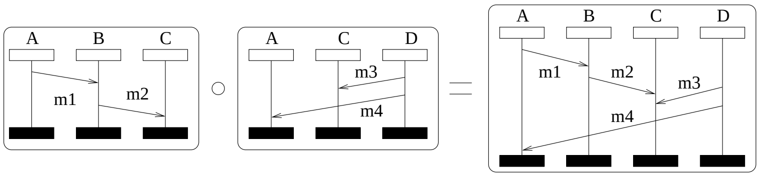

Fig. 2. Composition séquentielle

<u>MSCs bornés:</u> Les MSC bornés [1] sont une classe de MSC engendrant des langages réguliers (sans la restriction liée à la composition séquentielle forte). Pour cette classe de MSCs, le model-checking est décidable. Intuitivement, un MSC n'est pas borné s'il contient un cycle itérant un comportement dans lequel un groupe d'instances peut envoyer des messages à un autre groupe d'instances sans jamais attendre de message d'acquittement(voir l' exemple de la Figure 3. Cette situation, que l'on appelle aussi parfois divergence implique qu'un nombre potentiellement non borné de messages peuvent être en transit dans les canaux de communication du système, qui ne peut alors pas être représenté par un nombre fini d'états.

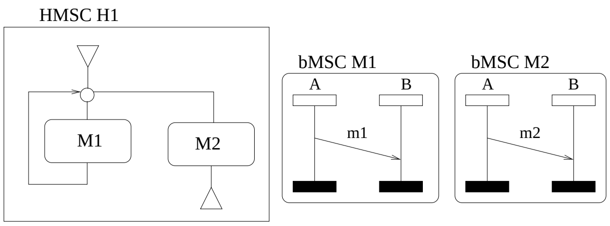

Fig. 3. Un exemple de MSC non borné

MSC Locaux: Un MSC est dit "à choix locaux" [2, 8] si et seulement si, pour chacun de ses noeuds de choix, une seule instance peut décider quel scénario doit être exécuté. Cette classe de MSC est très importante, car elle permet de décider de l'inclusion d'un HMSC dans un autre [4]. De plus, les MSC locaux sont très proches de comportements implémentables.

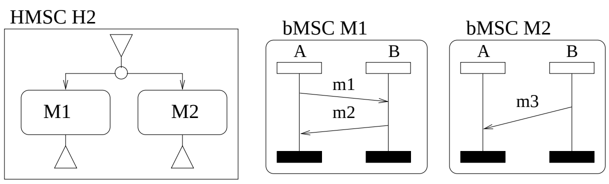

Fig. 4. Un exemple de MSC non local

MSCs confluents: Un MSC est dit confluent [12, 9] si des exécution parallèles de comportements sont exprimées en utilisant un noeud de choix. Considérons le MSC de la Figure 5. D'après la sémantique des MSC, une exécution de a peut être concurrente à une exécution de b. De plus, suivant la définition du choix et de la séquence faible, un a peut être observé après l'exécution d'un b. En fait, le langage décrit par ce HMSC est $a^*.b.a^*$. Ce type de HMSC peut être très ambigu, car une interprétation usuelle serait de considérer qu'un a ne peut plus se produire après un b. De plus, les MSC confluents produisent un branchement infini dans le système de transition sous-jacent à la sémantique opérationnelle des MSC. Sur notre exemple, lorsque l'on choisit un b, on choisit également le nombre (arbitraire) de a qui peuvent se produire par la suite, ce qui implique une infinité de transitions étiquetées par a dans le système de transitions associé à l'exemple. Enfin, la plupart des problèmes indécidables sur les MSC sont dus à cette classe.

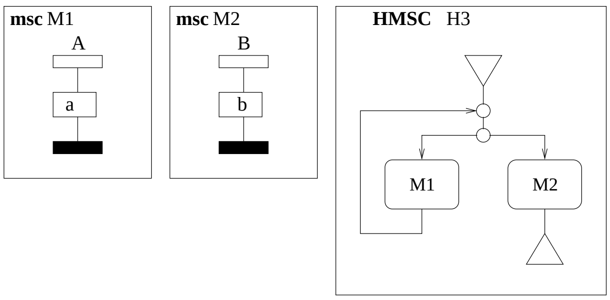

Fig. 5. Un exemple de MSC confluent

SOFAT permet de simuler tout MSC non confluent (indépendamment du fait qu'il soit effectivement implémentable de manière distribuée ou non). La simulation se fait en deux étapes: premièrement, le MSC est traduit en une grammaire de graphes, qui code une sémantique du MSC sous forme de structure d'événements [11]. Cette grammaire favorise une recherche efficace d'événements tirables en cours de simulation. Lors de la simulation, les états du système décrit sont des dépliages partiels de la grammaire de graphes, et les événement tirables sont les événements minimaux pour la relation d'ordre. Les événements tirables se trouvent dans l'état lui-même, ce qui favorise une recherche rapide. A chaque tir d'une transition, la grammaire est dépliée à la demande. Une description complète de l'environnement de simulation est proposé dans [7].

3 Conclusion

cet article a présenté les principales fonctionnalités de SOFAT, à savoir la classification et la simulation. Cet outil est en développement constant, car il est supposé être une boîte à outils intégrant un maximum de manipulations formelles de scénarios. Dans sa version actuelle, SOFAT accepte en entrée des Message Sequence Charts, mais pourrait être étendu sans trop d'efforts pour manipuler des diagrammes de séquence UML2 ou des Live Sequence Charts.

References

1. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. 10th Intl. Conf. on Concurrency Theory*, pages 114–129. Springer Verlag, 1999.
2. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Proc. of TACAS'97*, volume 1217 of *LNCS*, pages 259 – 274. Springer-Verlag, April 1997.
3. J. Berstel. *Transductions and Context-Free-Languages*. B.G. Teubner, 1979.
4. Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. In *Proc of ICALP 2002*, number 2380 in LNCS, pages 657–668. Springer Verlag, 2002.
5. Object Management Group. Omg unified modeling language specification, version 1.4, September 2001.
6. D. Harel and W. Damm. Lscs: breathing life into message sequence charts. Technical Report CS98-09, Weizmann Institute, Avril 1998.
7. L. Hélouët. A simulation framework for message sequence charts. In *Proc. of SDL'99: The Next Millenium*, pages 473–488. Elsevier, 1999.
8. L. Hélouët and C. Jard. Conditions for synthesis of communicating automata from hmscs. In *5th International Workshop on Formal Methods for Industrial Cr itical Systems (FMICS)*, Berlin, avril 2000. GMD FOKUS. http://www.gmd.de/publications/report/0091.
9. Loïc Hélouët. Some pathological message sequencecharts and how to detect them. In *Proc. of SDL'2001*, number 2078 in LNCS. Springer Verlag, 2001.
10. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva, September 1993.

11. C. Jard L. Hélouët and B. Caillaud. An event structure semantics for message sequence charts. *Mathematical Structures in Computer Science (MSCS) journal*, 12(4):377–403, 2002.
12. A. Muscholl. Matching specifications for message sequence charts. *LNCS*, 1578:273–287, 1999.
13. V. Pratt. Modeling concurrency with partial orders. *International journal of Parallel Programming*, 15(1):33–71, Mai 1986.

Appendix 1 : Exemples

La Figure 6 décrit l'architecture de notre boîte à outils. Elle est composée d'un analyseur lexical, qui permet de détecter des erreurs syntaxiques dans les fichiers d'entrée, mais qui effectue également une analyse des scénarios fournis. Une sortie textuelle indique si cette description est bornée, confluente, et si tous les choix sont des choix locaux. Si un MSC n'est pas confluent, alors sa sémantique peut être codée par une grammaire de graphe([11, 7]). Cette grammaire est ensuite utilisée comme entrée d'un simulateur graphique interactif. Si le MSC étudié est borné, la grammaire peut aussi être utilisée pour réaliser une exploration exhaustive de l'espace d'états du système décrit. Cette exploration peut ensuite être sauvegardée sous forme d'un système de transitions étiquetées.

Le prototype a été conçu en JAVA, et peut être exécuté sur n'importe quelle machine équipée d'un JDK postérieur à la version 1.2.x.

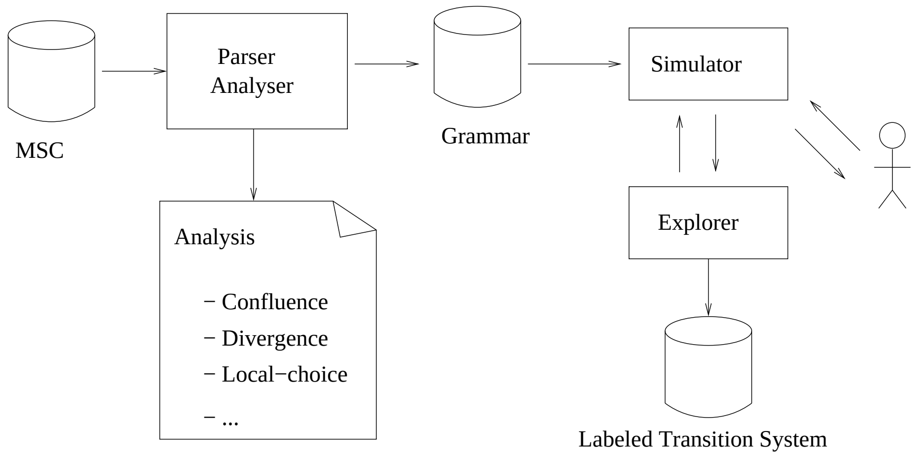

Fig. 6. La boîte à outils SOFAT

Regardons le MSC de la Figure 3. Comme précisé précédemment, ce MSC n'est pas borné, car une instance A peut envoyer un nombre infini de messages $m1$ avant qu'une seule réception ne se produise. Lorsqu'un cycle est trouvé, un graphe de communication est calculé pour le motif itéré par ce cycle. Un graphe de communication est un graphe dont les noeuds sont des noms d'instance, et qui comporte un arc de X vers Y si et seulement si X communique avec Y dans le motif considéré. Il y a une divergence si le graphe de communication associé à un cycle n'est pas fortement connexe. [1]. Ci dessous, nous montrons l'analyse de l'exemple Figure 3 par l'analyseur syntaxique. Notons que le graphe de communication associé au cycle 1,2 n'est pas fortement connexe.

```
kadupunch[14:56]%java slim unbounded.hmsc
Cycles found
Cycle :
1 2
A->B
Unbounded cyclic behaviour
No Confluence
All choices are local
```

Regardons le MSC non local Figure 4. Notons que ce MSC est borné, puisque son unique cycle répète un scénario dans lequel le message $m1$ doit être acquitté par le message $m2$ avant d'être réémis. La réponse de l'analyseur pour de MSC est la suivante:

```
kadupunch[15:03]%java slim nonlocal.hmsc
Cycles found
Cycle :
1 2
A->B
B->A
No Confluence
 Non local choice detected
```

Considérons à présent le MSC confluent de la Figure 5. Notons que puisque ce MSC ne comporte pas de communications, il ne peut pas être considéré comme divergent. La réponse de l'analyseur pour de MSC est la suivante:

```
kadupunch[14:56]%java slim confluent.hmsc
Cycles found
Cycle :
1 2
Node 1 is confluent
Confluence detected
 Non local choice detected
```

La Figure 7 décrit un protocole de communication très simple entre un émetteur $User\_A$ et un récepteur $User\_B$ qui utilisent un réseau $Network\_B$.

Ce protocole peut être décomposé en plusieurs phases, décrites par le HMSC. Premièrement, $User\_A$ demande une connexion au réseau. Cette demande peut être acceptée ou rejetée. Si la demande est acceptée, alors $User_A$ peut envoyer des données à $Network\_B$, qui les transmet à $User_B$. Ce comportement peut être répété un nombre de fois arbitraire. A tout moment, $User\_A$ peut aussi décider de rompre la connexion. Notons que suivant la sémantique de composition faible, les données transmises ne sont pas forcément reçues par $User\_B$ avant qu'une nouvelle émission ne se produise.

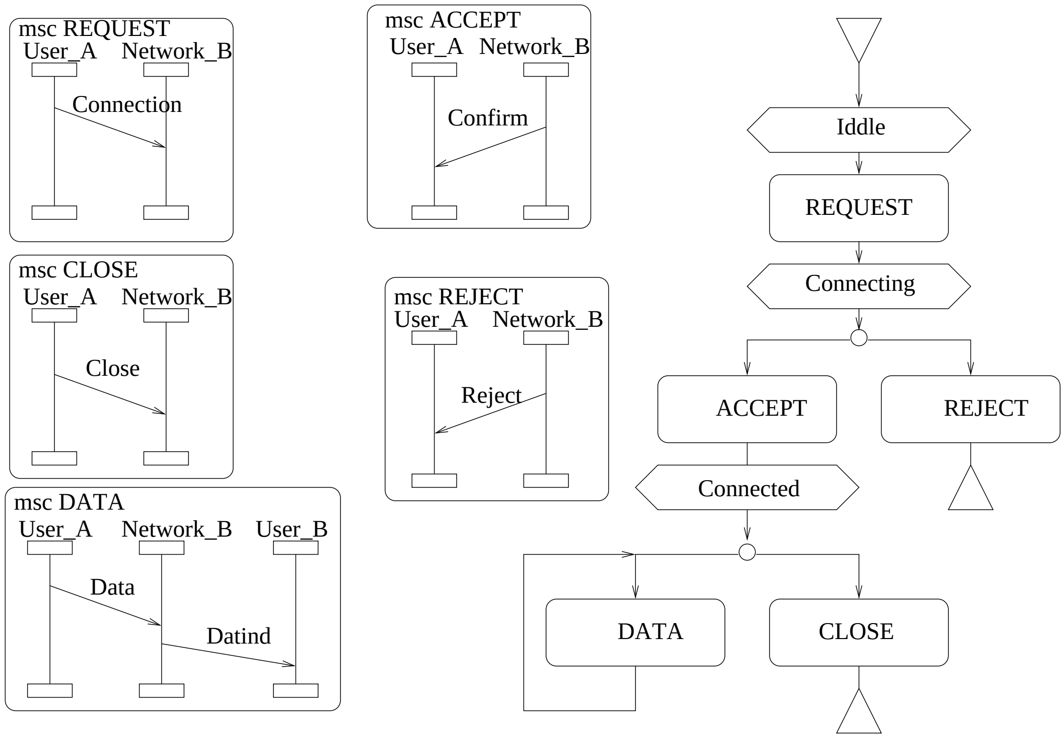

Fig. 7. Un exemple de protocole décrit par un HMSC

Le texte ci-dessous est une représentation textuelle (suivant la syntaxe de la norme [10]) du protocole décrit Figure 7.

```
mscdocument simple_protocol;

msc simple_hmsc ;
expr L1;
  L1: Request seq  (L2 alt L3);
  L2: Accept seq (L5) ;
  L3: Reject seq (L4) ;
  L4: end;
  L5: connect seq ( L6 alt L7);
  L6: Data seq (L5);
  L7: Close seq (L8);
  L8: end;
endmsc;

msc Request;
inst A,NB ;
    instance A;
        out conn to NB;
    endinstance;
    instance NB;
      in conn from A;
    endinstance;
endmsc;

msc Close;
inst A,NB ;
    instance A;
        out close to NB;
    endinstance;
    instance NB;
      in close from A;
    endinstance;
endmsc;

msc Accept;
inst A,NB ;
    instance A;
        in conf from NB;
    endinstance;
    instance NB;
      out conf to A;
    endinstance;
endmsc;

msc Reject;
inst A,NB ;
    instance A;
        in rej from NB;
    endinstance;
    instance NB;
      out rej to A;
    endinstance;
endmsc;

msc Data;
inst a,NB,B;
     instance A;
        out Dat to NB;
     endinstance;
     instance NB;
        in Dat from A;
        out Datind to B;
     endinstance;
     instance B;
        in Datind from NB;
     endinstance;
endmsc;

endmscdocument;
```

Le texte ci dessous est le résultat de l'analyse du protocole effectuée par l'analyseur de SOFAT. Le cycle détecté peut produire un nombre non borné de messages en transit, le protocole est donc non borné. Cependant puisqu'aucune confluence n'est détectée, ce protocole peut être simulé. Tous les choix du protocole (L1 et L5) sont locaux. Pour le choix entre *Accept* et *Reject*, l'instance prenant la décision est $Network\_B$,et pour le choix entre *Data* et *Close*, le scénario à exécuter est choisi par $User_A$.

```
kadupunch[14:36]%java slim sp.hmsc

Cycle :
5 6
A->NB
NB->B
Unbounded cyclic behaviour
No Confluence
All choices are local
```

Une grammaire de graphes a pu être générée par notre analyseur. Cette grammaire est utilisée comme modèle opérationnel pour une simulation. Puisque le protocole n'est pas borné, il est impossible de réaliser une simulation exhaustive. Cependant, une simulation interactive ou aléatoire peut souvent permettre de détecter certaine erreurs ou inconsistances dans une spécification. L'image de la Figure 8 est une copie d'écran prise durant la simulation du protocole. LA fenêtre de gauche contient une liste des événements tirables: un double clic sur un des événements permet de l'exécuter. La fenêtre de droit est une représentation de l'état courant, sous forme d'une structure d'événements. Les événements les plus haut sont les événements tirables. Les causalités entre événements sont représentées par un trait blanc. Enfin, deux événements qui ne peuvent pas apparaître dans une même exécution sont reliés par un arc de couleur bleue (on appelle ce type d'arc un conflit). Ces arcs proviennent des noeuds de choix dans le HMSC initial. Pour simplifier, les états représentés contiennent une représentation des événements tirables, des choix, et d'une certaine partie du futur proche. Lorsqu'un événement est tiré, un nouvel état est calculé, en dépliant la grammaire de graphes si besoin.

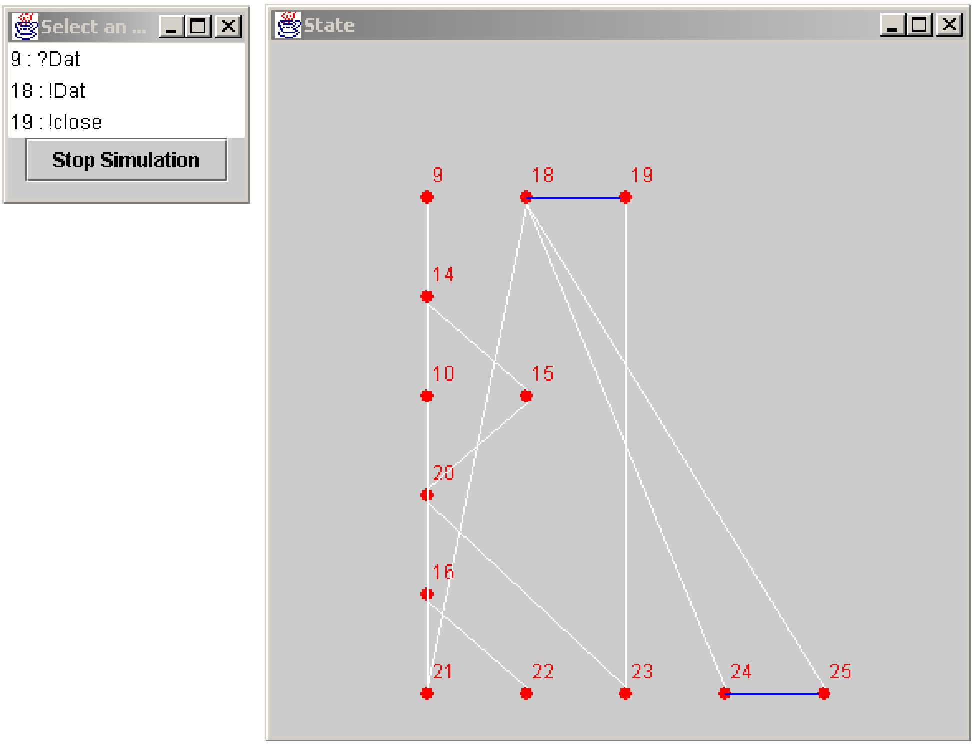

Fig. 8. Simulation interactive