End-to-end integrated security and performance analysis on the DEGAS Choreographer platform [*]

M. Buchholtz
Informatics and Mathematical Modelling
The Technical University of Denmark
Lyngby, Denmark
mib@imm.dtu.dk

S. Gilmore V. Haenel
Laboratory for Foundations of Computer Science
The University of Edinburgh
Scotland
stg@inf.ed.ac.uk,
valentin.haenel@gmx.de

C. Montangero
Dipartimento di Informatica
Università di Pisa
Pisa
Italy
carlo.montangero@di.unipi.it

ABSTRACT

We present a software tool platform which facilitates security and performance analysis of systems which starts and ends with UML model descriptions. A UML project is presented to the platform for analysis, formal content is *extracted* in the form of process calculi descriptions, analysed with the analysers of the calculi, and the results of the analysis are *reflected* back into a modified version of the input UML model. The design platform supporting the methodology, *Choreographer*, interoperates with state-of-the-art UML modelling tools. We illustrate the approach with a well known protocol.

Categories and Subject Descriptors

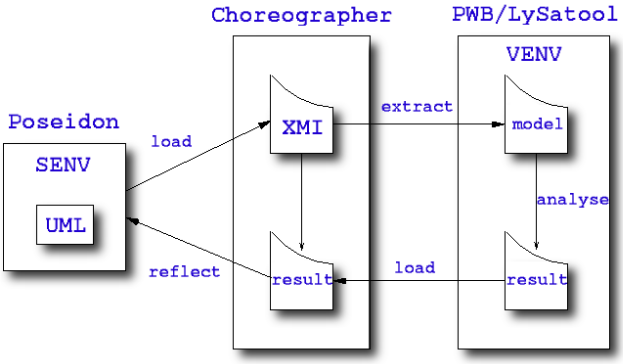
C.4 [**Performance of Systems**]: Modeling techniques; I.6 [**Simulation and modeling**]: Model Development—*Modeling methodologies*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; C.2 [**Computer-communication networks**]: Network Protocols—*Protocol verification*

General Terms

UML, performance modelling, communications protocols, static analysis

1. INTRODUCTION

Choreographer is an integrated design platform for coherent and consistent qualitative and quantitative modelling of software systems. It processes UML models as its input, and writes modified versions of these as its output. The architecture of the tool is to consider the interface to a specification environment (SENV) and a processing interface to a verification environment (VENV). Models which are input for analysis are channelled from the SENV to the VENV via software connectors known as *extractors*. The extracted formal content is passed to the VENV for analysis. The results of the analysis are recombined with the input model and channelled from the VENV back to the SENV via software connectors known as *reflectors*. In the specific configuration of the architecture which we discuss here the SENV is the Poseidon UML platform from Gentleware [1] and both the LySatool [2] and the PEPA Workbench [3, 4] are VENVs.



The qualititative analysis is deployed to investigate the security of the communication protocols used in the application. The analysis guarantees there are no successful attacks on the authentication of the communicated messages. In the case where authentication may be breached the analysis reports where the breach may occur.

The quantitative analysis which is provided is a performance analysis of the system model. This identifies components which are under-utilised or over-utilised indicating poor deployment of computational resources.

In the development of the Choreographer platform we were concerned to support not only the UML notation but the UML design process in order that UML develop-

---

[*] (Produces the permission block, copyright information and page numbering). For use with ACM_PROC_ARTICLE-SP.CLS V2.6SP. Supported by ACM.

ICSE '05 St. Louis, Missouri, USA

ers would be comfortable with working with the platform. That is, we devoted considerable effort in the design of the extractors to ensuring that the UML was being used as more than just a graphical syntax for the process calculi beneath.

2. ENGINEERING ISSUES

Our functional requirements for the Choreographer design platform were that it should provide access to the analysis procedures of the PEPA performance analysis tool (the PEPA Workbench in both its ML and Java editions [3, 4]) and the LySa security analysis tool (the LySatool [2]). In addition, it needs to interoperate with a fully-featured UML tool.

Our non-functional requirements on the platform were that we wanted to develop a professional quality tool in a constrained time, with a modest budget for developer effort. We also had the requirement that the tool should be available across platforms (in our case, Windows and Linux). We evaluated the generic IDEs of Eclipse and NetBeans and the Argo/UML, XDE, MagicDraw and Poseidon UML tools. We took the decision to build the Choreographer platform on top of NetBeans on the Java platform and have it interoperate with Poseidon. This decision was a complex engineering compromise between a number of conflicting tensions. Our choice went a considerable way towards addressing portability concerns but the portability issue was impacted also by the availability of the analysers and UML drawing tools we wanted to integrate with.

We wanted Choreographer to have two dimensions of portability. The first is the most obvious one, that it should run successfully on both Windows and Linux. This requirement for code portability has been successfully addressed. The second dimension of portability was that we wanted the Choreographer platform to interoperate with many UML tools via the standard XML Interchange format (XMI) for UML diagrams. Choreographer needs to deal with these because it reads from and writes into this import/export format. This data portability requirement was the more difficult problem, and one which we have not been able to solve perfectly. There are many versions of the XMI standard for UML, and different UML tools implement their chosen version to a more or less satisfactory extent. Some releases of the UML tools which we tried wrote non-well-formed XMI output, even according to their own criteria. Such inconsistency makes interoperation essentially a matter of writing a custom reader/writer pair for every version of every UML tool with which one wants to interoperate, which is the trap which standards such as XMI were intended to prevent developers falling into.

A configuration which we considered for Choreographer was XDE and Eclipse together. The XDE UML tool is provided as an Eclipse plug-in, so this is a natural coupling. We rejected this combination because the XDE tool is not available in a Linux release. We chose not to interoperate with MagicDraw because it is not freely available. We could not work with Argo/UML because it did not represent some aspects of the UML diagrams in the XMI format, thus crippling its use as an import/export model exchange format.

A potential source of non-portability might have been the formal analysis tools which we used. These had been implemented in Java or the functional programming language Standard ML. However, we discovered that the Standard ML of New Jersey compiler which we used had very closely conforming versions for Linux and Windows, making the portability of these formal analysis tools essentially only a matter of working around small differences in the versions of the standard library for the two platforms. This level of minor tuning is also required for application development in the Java language, which has given more effort to ensuring cross-platform portability than perhaps any other programming language.

3. PROTOCOLS AND AUTHENTICATION

The usual remedy to protect network protocols from intervention by malicious attackers is to apply cryptography so that parts of the messages may be kept outside the control of the attacker. Many security properties such as confidentiality, authenticity, non-repudiation, etc. are of interest when considering whether a protocol is well-behaved or not. Here, we focus on checking an authentication property, namely that “messages protected by encryption should only be decrypted at the right places”.

The verification technique we use builds on the modelling of protocols in LySa, a process calculus in the π-calculus tradition, specifically tailored to model central aspects of security protocols [5]. A protocol is modelled in LySa in scenario with several kinds of principals: an *initiator* of the protocol, a *responder*, and a *server*, referred to as a trusted third party, a key distribution centre, a certificate authority, etc. Besides, there can be many principals acting as initiators and as responders.

To specify the authentication property that encrypted messages end up at the right places, the LySa process is annotated: each encryption and decryption point is named ℓ, ℓ', etc., and is furthermore annotated with its intended destinations and origins.

Our verification relies on a control flow analysis [5, 6] of LySa that tells whether the authentication properties are satisfied for all executions of the annotated LySa process, executed in parallel with an arbitrary attacker process. The analysis reports all possible breaches of the authentication properties in an error component ψ: a pair (ℓ, ℓ') in ψ means that something encrypted at ℓ was decrypted at ℓ' breaking the specified authentication property. The analysis computes over-approximations of ψ, i.e. it may report an error that is not actually there: [5] also illustrates why this is not a big problem in practice. To model security protocols in UML consistently, we have defined a specific profile [2]. The profile introduces stereotypes for core concepts like principals, keys, and messages, and for the concepts needed for the analysis.

To specify a protocol in UML so that the extractor citeForLySa can feed the analyser [2], the designer exploits the stereotypes in a class diagram presenting the structure of the protocol, with the intended communications, the involved messages, and the local information of each principal, like private keys, session keys, and temporary storage, and their operation to build and dissect messages. Besides, the structure of each message is specified in distinct diagrams, one per message type and includes the decorations needed to specify the authentication property. Conceptually, the information in the UML diagrams corresponds to what is modelled by a LySa process and the decorations specifying the authentication property corresponds to annotation of the LySa process.

The dynamics of the protocol is given in a sequence diagram, which describes the typical run on the protocol. Each message exchange is divided into three steps: 1. the sender packages the message, 2. the message is communicated, and 3. the recipient processes the incoming message. The operations are specified via post-condition constraints on the state of the principal. The places mentioned by the authentication properties are specified as notes associated to the messages in steps 1 and 3 above. These notes are placeholders that will support the notification of eventual errors resulting from the analysis. If the analysis reports an error being the pair (ℓ, ℓ') in ψ, the note introducing ℓ will be modified by the reflector to list ℓ', thereby signalling the error reported by the analysis.

4. PERFORMANCE EVALUATION

Despite impressive improvements in the computational power which is now available to end-users of computer systems, computer equipment is still expensive to purchase and maintain. Consequently, owners are often motivated to make the best use of their available resources. The analysis of computer systems through construction and solution of descriptive models is a hugely profitable activity: brief analysis of a model can provide as much insight as hours of simulation and measurement [7].

Simple models of a computer system can be constructed without any explicit notational support. However, as computer systems become more complex so do their models and the use of a high-level language to aid in their expression becomes necessary. Jane Hillston's Performance Evaluation Process Algebra (PEPA) [8] is an expressive formal language for modelling distributed systems. PEPA models are constructed by the composition of components which perform individual activities or cooperate on shared ones. To each activity is attached an estimate of the rate at which it may be performed.

Using such a model, a system designer can determine whether a candidate design meets both the behavioural and the temporal requirements demanded of it. That is: the protocol may be secure, but can it be executed quickly enough to complete the message exchange within a specified time bound, with a given probability of success?

Rather than composing process calculus models directly—although Choreographer also supports this mode of operation—we extract these from UML class, state and collaboration diagrams. For the purposes of performance analysis we extract a process calculus model in PEPA. The extractor for PEPA is documented in [9].

The relationship between the process algebra model and the CTMC representation is the following. The process terms (P_i) reachable from the initial state of the PEPA model by applying the operational semantics of the language form the states of the CTMC (X_i). For every set of labelled transitions between states P_i and P_j of the model $\{(\alpha_1, r_1), \ldots, (\alpha_n, r_n)\}$ add a transition with rate r between X_i and X_j where r is the sum of $r_1, \ldots, r_n$. The activity labels (α_i) are necessary at the process algebra in order to enforce synchronisation points, but are no longer needed at the Markov chain level.

Under conditions on the form of the model where every state is positive-recurrent, every such CTMC has a stationary probability distribution over the states of the chain. Knowing the rates associated with the activities of the system this stationary probability distribution can be obtained using procedures of numerical linear algebra such as Gaussian elimination, conjugate gradient methods, or over-relaxation methods such as Jacobian over-relaxation or successive over-relaxation.

Such a stationary probability distribution is rarely the desired end result of the performance analysis process but meaningful performance measures such as throughput and utilisation can be directly calculated from the stationary distribution. State-space generation and numerical solution is the computationally expensive part of performance analysis. The size of the state-space of the system is bounded by the product of the sizes of the sequential components in the model and thus modelling with continuous-time Markov chains is subject to the familiar *state-space explosion* problem.

5. METHODOLOGY

The methodology which we follow is to first attempt a security analysis and then, if this is successful, progress to a performance analysis. The reasoning behind this methodology is that the security analysis rests on static analysis procedures which have a lower asymptotic complexity than the state-space generation and iterative numerical procedures which are needed for the performance analysis. Thus, ordering them in this way potentially gives a significant saving in the overall computation time by avoiding the performance analysis of an erroneous protocol.

Therefore, having described the protocol using a UML sequence diagram we apply the For-LySa extractor to generate a LySa model which we analyse with the LySatool. If the LySatool detects errors in the protocol, indicating that it is insecure, the results are reflected back to the UML level, so that we can view the results in the Poseidon tool. Having identified these flaws we can repair the protocol and continue with performance analysis. Here, we extract a PEPA process algebra model from the UML input. We solve this for its equilibrium probability distribution using successive over-relaxation (SOR), then reflect. The information returned from the analysis quantifies the percentage of time that the principals and the server spend in their local states, pointing to performance-related problems such as under- or over-utilisation, starvation, bottlenecks, or hotspots in the system. We can investigate the potential benefits to be obtained by improving the implementation of the activities in the system, thereby identifying the place or places where it will be most profitable to spend developer effort.

Evidently, it is possible to discover at this stage that the required improvements in the execution of the activities of the system might be infeasible to achieve, especially in the setting of weak computing devices such as smartcards or low-end PDAs or in a thin client context with intermittent or very narrow bandwidth connections between devices. If this is the case, then a developer working at the early modelling stage of the system development process would need to revisit the initial protocol design and perhaps redesign this to involve fewer message exchanges or reduce the amount of asymmetric cryptography used. This will initiate another cycle of security analysis and performance analysis in pursuit of the levels of security and performance demanded of the system.

6. RELATED WORK

Tool support for the automated analysis of security requirements in the UMLsec framework [10] is described and accessible at [11]. The relevant elements of the UML specification are translated in the input language of the model-checker SPIN and the dynamic property to be verified is translated in Linear Temporal Logic. The UML models are stored in a MDR library, and accessed via the generated JMI interface.

Work which is similar in spirit to our own approach is that of Petriu and Shen [12] where a layered queueing network model is automatically extracted from an input UML model with performance annotations in the format specified by a special-purpose UML profile [13]. We do not follow the same UML profile because it is not supported by our modelling tool. Additionally, the performance evaluation technology which we deploy (process algebras and BDD-based solution) is quite different from layered queueing networks.

Another performance engineering method which is similar to ours is that of López-Grao, Merseguer and Campos [14] where UML diagrams are mapped into GSPNs which can be solved by GreatSPN. We use different UML diagram types from these authors and, again, a different performance evaluation technology. Stochastic Petri nets and stochastic process algebras have different, but complementary, modelling strengths [15].

One feature of our work which is distinctive from both of the above is the role of a *reflector* in the system to present the results of the performance evaluation back to the UML modeller in terms of their input model. We consider this to be a strength of our approach. We do not only compile a UML model into a performance model, we also present the results back to the modeller in the UML idiom.

Acknowledgements: The authors are supported by the DEGAS (Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing. We thank Matthew Prowse for helpful discussions on PEPA extraction.

References

[1] Gentleware AG systems. Poseidon for UML web site, November 2004. `http://www.gentleware.com/`.

[2] Mikael Buchholtz. LySa — a process calculus. Web site hosted by Informatics and Mathematical Modelling at the Technical University of Denmark, April 2004. `http://www.imm.dtu.dk/cs_LySa/`.

[3] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.

[4] N.V. Haenel. *User Guide for the Java Edition of the PEPA Workbench—Tabasco release*. LFCS, Edinburgh, October 2003.

[5] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H.R. Nielson. Automatic validation of protocol narration. In *Proc. of the 16th Computer Security Foundations Workshop (CSFW 2003)*, pages 126–140. IEEE Computer Security Press, 2003.

[6] M. Buchholtz, C. Montangero, L. Perrone, and S. Semprini. For-LySa: UML for authentication analysis. In C. Priami and P. Quaglia, editors, *Proceedings of the second workshop on Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 92–105. Springer Verlag, 2004.

[7] Isi Mitrani. *Probabilistic Modelling*. Cambridge University Press, 1998.

[8] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[9] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. *IEE Proceedings: Computers and Digital Techniques*, 150(2):107–120, March 2003.

[10] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2004.

[11] Jan Jürjens. Umlsec webpage. Accessible at `http://www.umlsec.org`, 2002–04.

[12] D.C. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In A.J. Field and P.G. Harrison, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 159–177, London, UK, April 2002. Springer-Verlag.

[13] B. Selic, A. Moore, M. Woodside, B. Watson, M. Bjorkander, M. Gerhardt, and D. Petriu. Response to the OMG RFP for Schedulability, Performance, and Time, revised, June 2001. OMG document number: ad/2001-06-14.

[14] J.P. López-Grao, J. Merseguer, and J. Campos. From UML activity diagrams to stochastic Petri nets: Application to software performance analysis. In *Proceedings of the Seventeenth International Symposium on Computer and Information Sciences*, pages 405–409, Orlando, Florida, October 2002. CRC Press.

[15] S. Donatelli, J. Hillston, and M. Ribaudo. A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, North Carolina, 1995.

[16] M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *ACM Transactions on Computing Systems*, 8(1):18–36, February 1990.

7. APPENDIX A: DEMONSTRATION

As a running example in the demonstration, we will apply Choregrapher to analyse variations on the Wide-Mouthed-Frog authentication protocol, which was originally presented in [16].

This protocol describes key exchange between two principals (A and B) through a trusted server. A and B have no prior communication history with each other but both have previously contacted the server and have retained keys K_{AS} and K_{BS} respectively. The protocol has three steps.

1. Principal A sends a message to the server including the name of B and the new session key K_{AB}, encrypted under K_{AS}.
2. The server decrypts this and sends the name of A and the new key K_{AB} to B, encrypted under K_{BS}.
3. Principal A sends a message to B encrypted under K_{AB}.

According to the DEGAS methodology, we first attempt a security analysis. For the sake of demonstration, we try first with a flawed version, which we then correct and subject to performance analysis.

In more details, the demo will go through the following steps.

1. General description of the architecture of the tool and of the related methodology.
2. Overview of the interface of Choreographer, describing the main menus, as shown in screenshot.
3. Demonstration of how to 'load' a UML model (built in Poseidon) in Choreographer, and to make it available for analysis, as shown in the screenshot in Figure 1.
4. Overview of the running example UML model, with respect to security analyis. The essence is a sequence diagram, of which we offer no screenshot here, since it is very similar to the one shown below with the results of the analysis.
5. Demonstration of how to 'extract' a LySa model, and invoke the LySatool perform the security analysis, as shown in the screenshot in Figure 2.
6. 'Reflecting' the result of the security analysis back to UML as shown in the screenshot in Figure 3. The menus are available in both pop-up and drop-down form, showing the entries to invoke the extraction, analysis and reflection operations. During the analysis, information about its progress is shown for the knowledgeable designer (in the console in the lower part of the screenshot in Figure 3). The reflected UML model is a modified version of the input with additional annotations which place the analysis results onto the diagram at the appropriate places. We have circled the differences between the input and the result on the Poseidon screenshot in Figure 4.
7. Interpretation of the results of the security analysis, and identifications of the flaws; time permitting, intervention on the model to fix it (an already repaired model may be loaded instead, to streamline the presentation).
8. New security analysis, with no errors.
9. Overview of the running example UML model, with respect to performance analyis. The essence is a state machine diagram, for the sequential components, and a collaboration diagram to describe the operational instance to be investigated in the performance analysis.
10. Demonstration of how to 'extract' a PEPA model (Figure 5).
11. Checking the model (Figure 6) and performing the performance analysis. During the analysis, information about its progress is shown for the knowledgeable designer (lower half of the screenshot in Figure 7).
12. Reflecting the results in UML, as shown in Figure 7. The drop-down menu shows the entries to invoke these operations.
13. Also in this case, the reflected UML model is a modified version of the input with additional annotations which place the analysis results onto the diagram at the appropriate places. We have circled the differences between the input and the result on the Poseidon screenshot in Figure 8.
14. Interpretation of the results of the performance analysis, and identifications of possible improvements; time permitting, intervention on the model to assess one such improvement.
15. Wrap up.

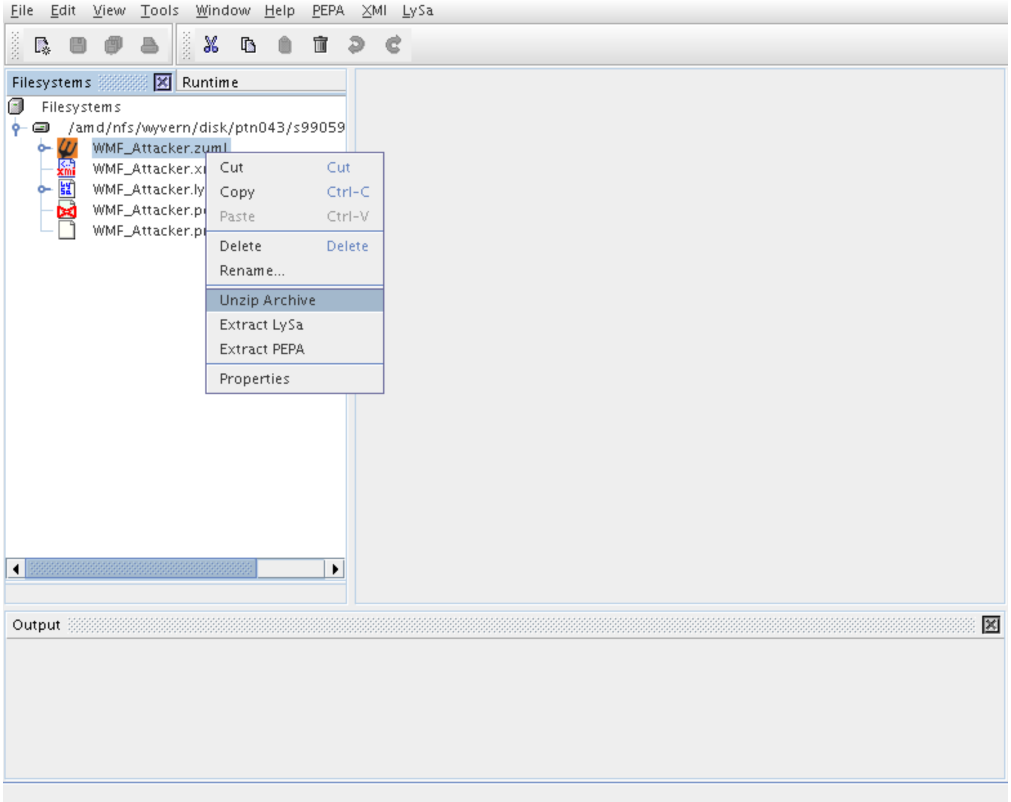
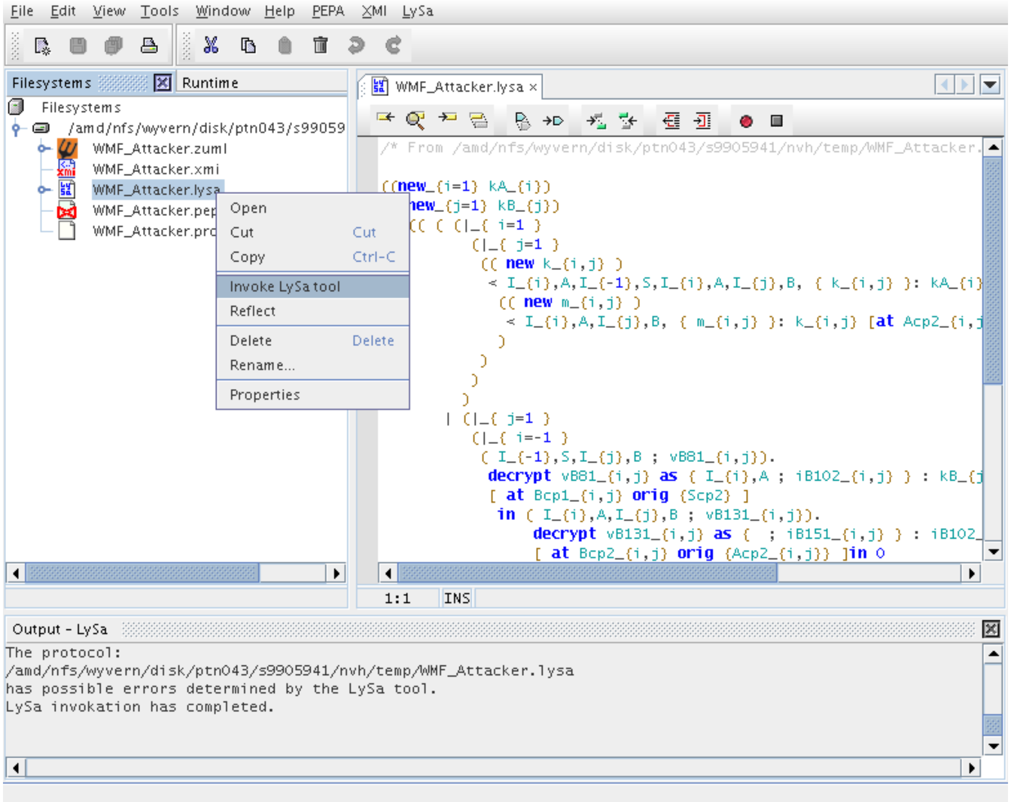


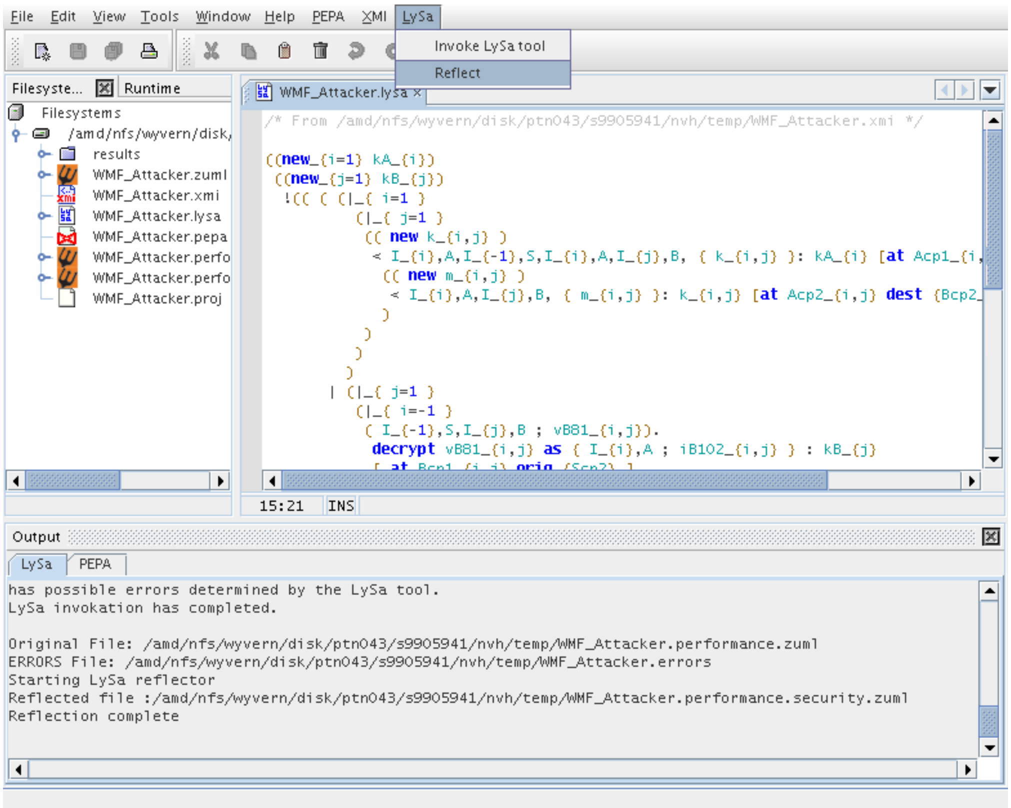
Figure 1: Opening a UML archive in Choregrapher



```
/* From /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.

((new_{i=1} kA_{i})
   new_{j=1} kB_{j})
   (( ( (|_{ i=1 }
          (|_{ j=1 }
           (( new k_{i,j} )
            < I_{i},A,I_{-1},S,I_{i},A,I_{j},B, { k_{i,j} }: kA_{i}
             (( new m_{i,j} )
              < I_{i},A,I_{j},B, { m_{i,j} }: k_{i,j} [at Acp2_{i,j
             )
           )
          )
         )
       | (|_{ j=1 }
          (|_{ i=-1 }
           ( I_{-1},S,I_{j},B ; vB81_{i,j}).
            decrypt vB81_{i,j} as { I_{i},A ; iB102_{i,j} } : kB_{j
            [ at Bcp1_{i,j} orig {Scp2} ]
             in ( I_{i},A,I_{j},B ; vB131_{i,j}).
                 decrypt vB131_{i,j} as {  ; iB151_{i,j} } : iB102_
                 [ at Bcp2_{i,j} orig {Acp2_{i,j}} ]in 0
```

```
The protocol:
/amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.lysa
has possible errors determined by the LySa tool.
LySa invokation has completed.
```

Figure 2: Invoking the LySatool on a security model



```
/* From /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.xmi */

((new_{i=1} kA_{i})
 ((new_{j=1} kB_{j})
  !(( ( (|_{ i=1 }
          (|_{ j=1 }
           (( new k_{i,j} )
            < I_{i},A,I_{-1},S,I_{i},A,I_{j},B, { k_{i,j} }: kA_{i} [at Acp1_{i,
             (( new m_{i,j} )
              < I_{i},A,I_{j},B, { m_{i,j} }: k_{i,j} [at Acp2_{i,j} dest {Bcp2_
             )
           )
          )
         )
       | (|_{ j=1 }
          (|_{ i=-1 }
           ( I_{-1},S,I_{j},B ; vB81_{i,j}).
            decrypt vB81_{i,j} as { I_{i},A ; iB102_{i,j} } : kB_{j}
```

```
has possible errors determined by the LySa tool.
LySa invokation has completed.

Original File: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.performance.zuml
ERRORS File: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.errors
Starting LySa reflector
Reflected file :/amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.performance.security.zuml
Reflection complete
```

Figure 3: Reflecting the results back to UML

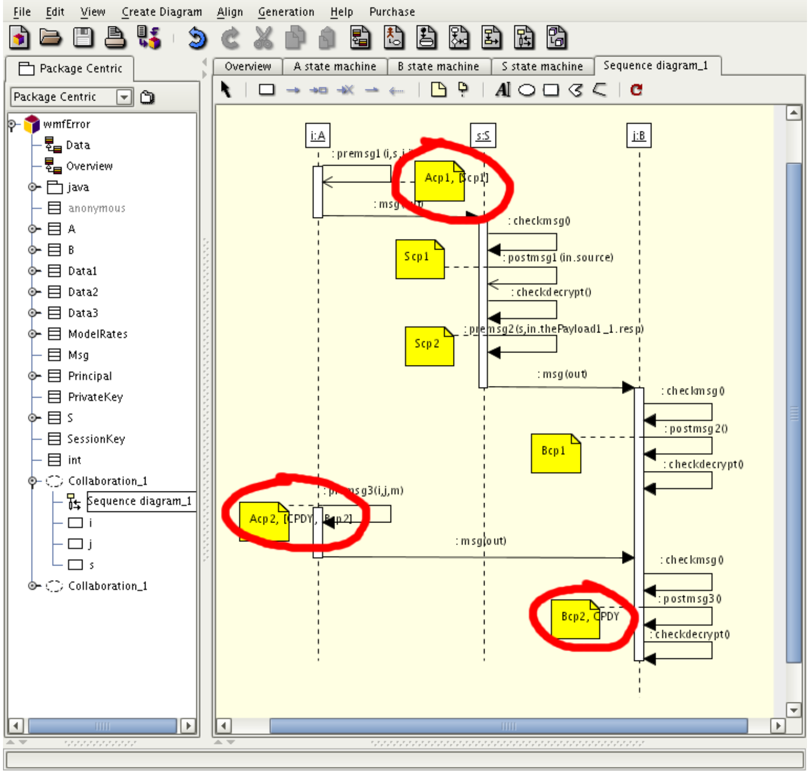
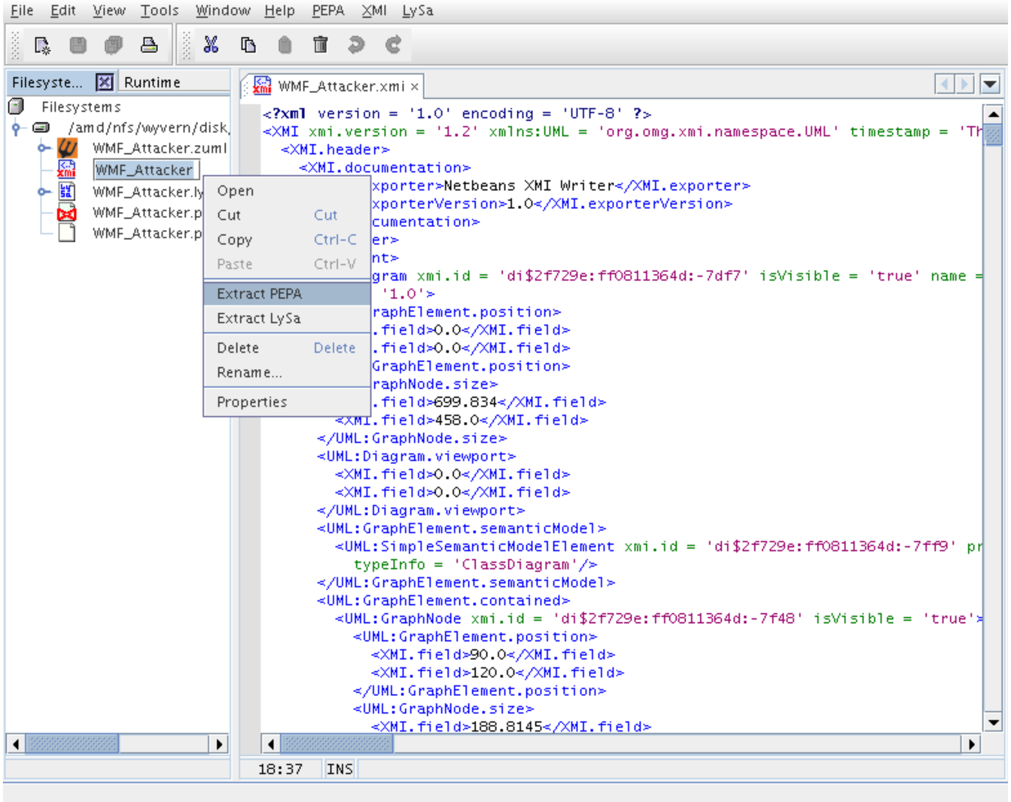
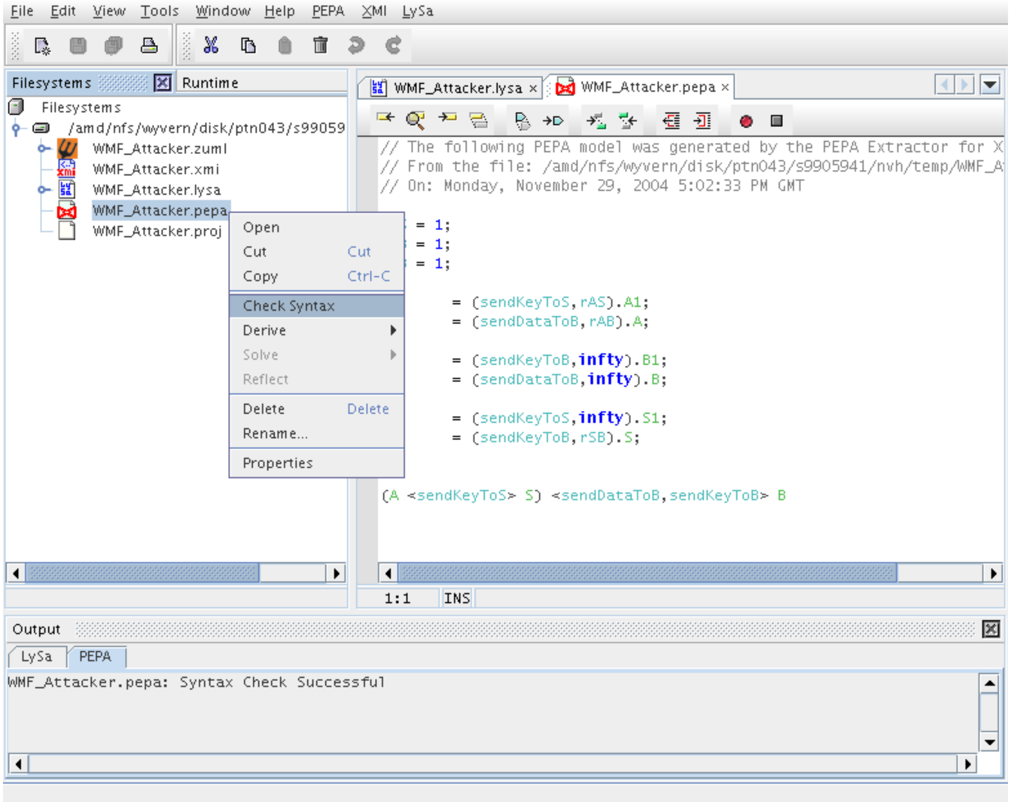


Figure 4: Viewing the results in Poseidon



```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Th
  <XMI.header>
    <XMI.documentation>
            xporter>Netbeans XMI Writer</XMI.exporter>
            xporterVersion>1.0</XMI.exporterVersion>
            cumentation>
            er>
            nt>
            gram xmi.id = 'di$2f729e:ff0811364d:-7df7' isVisible = 'true' name =
             '1.0'>
            raphElement.position>
            .field>0.0</XMI.field>
            .field>0.0</XMI.field>
            GraphElement.position>
            raphNode.size>
            .field>699.834</XMI.field>
        <XMI.field>458.0</XMI.field>
      </UML:GraphNode.size>
      <UML:Diagram.viewport>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
      </UML:Diagram.viewport>
      <UML:GraphElement.semanticModel>
        <UML:SimpleSemanticModelElement xmi.id = 'di$2f729e:ff0811364d:-7ff9' pr
          typeInfo = 'ClassDiagram'/>
      </UML:GraphElement.semanticModel>
      <UML:GraphElement.contained>
        <UML:GraphNode xmi.id = 'di$2f729e:ff0811364d:-7f48' isVisible = 'true'>
          <UML:GraphElement.position>
            <XMI.field>90.0</XMI.field>
            <XMI.field>120.0</XMI.field>
          </UML:GraphElement.position>
          <UML:GraphNode.size>
            <XMI.field>188.8145</XMI.field>
```

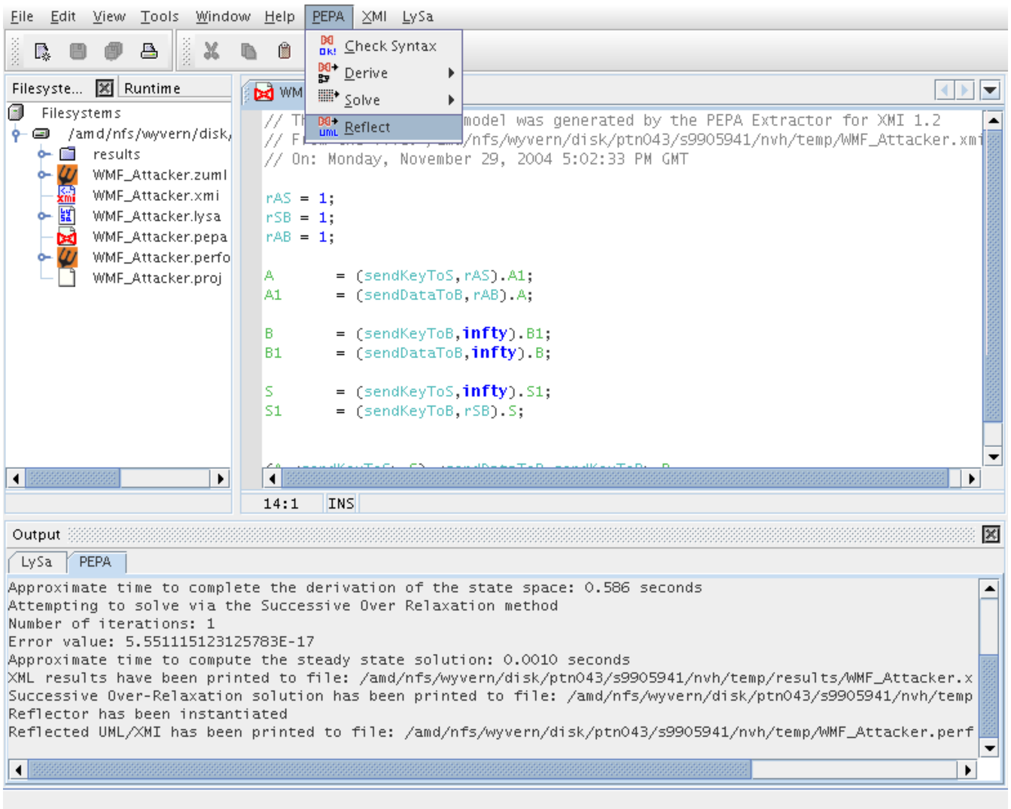
Figure 5: Extracting a PEPA performance model



```
// The following PEPA model was generated by the PEPA Extractor for X
// From the file: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_A
// On: Monday, November 29, 2004 5:02:33 PM GMT

    = 1;
    = 1;
    = 1;

        = (sendKeyToS,rAS).A1;
        = (sendDataToB,rAB).A;

        = (sendKeyToB,infty).B1;
        = (sendDataToB,infty).B;

        = (sendKeyToS,infty).S1;
        = (sendKeyToB,rSB).S;


(A <sendKeyToS> S) <sendDataToB,sendKeyToB> B
```

```
WMF_Attacker.pepa: Syntax Check Successful
```

Figure 6: Checking well-formedness of the model



```
// Th                    model was generated by the PEPA Extractor for XMI 1.2
// F                     /nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.xmi
// On: Monday, November 29, 2004 5:02:33 PM GMT

rAS = 1;
rSB = 1;
rAB = 1;

A        = (sendKeyToS,rAS).A1;
A1       = (sendDataToB,rAB).A;

B        = (sendKeyToB,infty).B1;
B1       = (sendDataToB,infty).B;

S        = (sendKeyToS,infty).S1;
S1       = (sendKeyToB,rSB).S;
```

```
Approximate time to complete the derivation of the state space: 0.586 seconds
Attempting to solve via the Successive Over Relaxation method
Number of iterations: 1
Error value: 5.551115123125783E-17
Approximate time to compute the steady state solution: 0.0010 seconds
XML results have been printed to file: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/results/WMF_Attacker.x
Successive Over-Relaxation solution has been printed to file: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp
Reflector has been instantiated
Reflected UML/XMI has been printed to file: /amd/nfs/wyvern/disk/ptn043/s9905941/nvh/temp/WMF_Attacker.perf
```

Figure 7: Reflecting the results back to UML

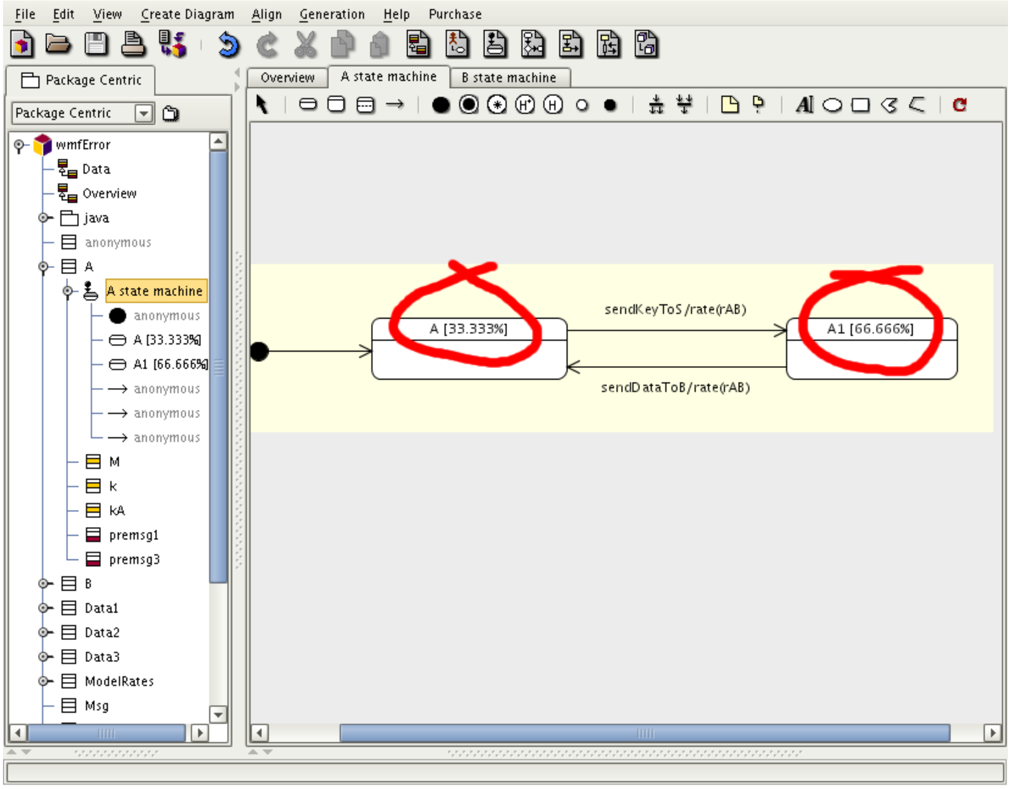


Figure 8: A Poseidon screenshot with the changes circled