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

Mikael Buchholtz [1], Stephen Gilmore [2], Valentin Haenel [2], and Carlo Montangero [3]

[1] Informatics and Mathematical Modelling, The Technical University of Denmark, Lyngby, Denmark. Email: mib@imm.dtu.dk
[2] Laboratory for Foundations of Computer Science, The University of Edinburgh, Scotland. Email: stg@inf.ed.ac.uk, valentin.haenel@gmx.de
[3] Dipartimento di Informatica, Università di Pisa, Pisa, Italy. Email: 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 and report on the experience of industrial users who have applied Choreographer in their development work.

Keywords: security analysis, performance analysis, process calculi, UML.

1 Introduction

The safety and reliability of networked software applications becomes a highly significant matter as such systems play an ever-increasing role in society and public life. Software systems win the trust of users by being secure against attack and by remaining available and responsive under increasing workload. Security and quality-of-service valuations such as these give rise to subtle and complex questions about these complex systems. Determining the answers to these questions necessitates careful modelling and analysis of these systems in well-founded formal calculi. Such reasoning is both too detailed and too arduous to be undertaken by hand and so modelling and design tools play a crucial role in designing and evaluating the computing applications of today and tomorrow.

Choreographer is an integrated design platform for coherent and consistent qualitative and quantitative modelling of software systems. The main idea is to cater for formal verification of system specifications through application of existing analysis tools and techniques. However, these analysis tools will largely be hidden from the developer, who only needs to related to the analysis at the

level of the system specification, which is already familiar to him. To this end, the Choreographer processes UML models as its input, and writes modified versions of these as its output. All that takes place between input and output will be performed fully automatic and, consequently, is of no concern for the developer.

The architecture of the Choreographer tool as illustrated in Figure 1 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*. The extraction, the verification, and the reflection are all automated and can therefore run without interference from the user of the Choreographer.

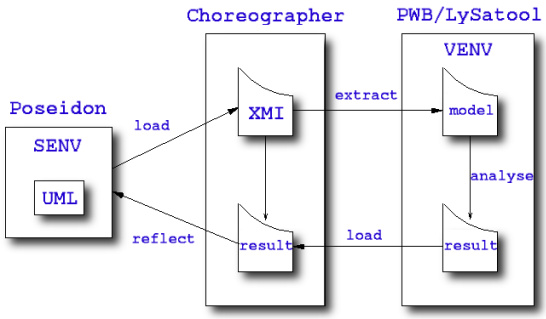

Fig. 1. Software architecture of the tool chain used by Choreographer

In this paper we discuss a specific configuration of the architecture in which the SENV is the Poseidon UML platform from Gentleware [1] and both the LySatool [2] and the PEPA Workbench [3, 4] are VENVs. The software tool chain which is formed when these are connected is also depicted in Figure 1.

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 developers 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.

Structure of this paper: In the next section we discuss the methodology behind the Choreographer platform, explaining how it has been used in practice. In Section 3 we describe the security properties which Choreographer can verify, and how this is achieved. In Section 4 we progress to a description of using Choreographer for performance evaluation based on the generation and solution of continuous-time Markov chains. Section 5 details the extraction and reflection operations which connect the UML input and output to the analysis routines beneath. Section 6 is a small example, making the foregoing descriptions concrete. Section 7 relates our experience of building the Choreographer platform. Section 8 reports on the experiences of our industrial users. A description of related work and conclusions follow.

2 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 re-design 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.

3 Security Analysis

For our security analysis we rely on techniques from data and control flow analysis. These are analysis techniques that automatically compute information about the entire behaviour of a software system including its behaviour when the systems is under attack. A trademark of these techniques is that they are automatic and complexity-wise efficient, which makes them well-suited as back-end analysis tools for Choreographer.

In more detail, the analysis techniques work by finding conservative over-approximations to system behaviour. That is, the analysis computes an over-approximation of the behaviour of a system under attack from any arbitrary attacker. With regards to security, this means that the analysis can *guarantee the absence* of attacks because they provide information about the entire behaviour of a system. However, because the analysis techniques are approximative they cannot guarantee the presence of attacks and may report warnings about possible attacks that in fact do not exist. In the following, we discuss a control flow analysis that guarantees authentication properties for encrypted network communication.

3.1 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. Cryptography may be applied to attain many different security properties such as confidentiality, authenticity, non-repudiation, etc. 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, which is a process calculus in the π-calculus tradition. LySa is specifically tailored to model central aspects of security protocols [5] such as (perfect) cryptography, nonces, network communication, etc. A protocol modelled in LySa will be analysed 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] of LySa, which is implemented in the LySatool [2]. The analysis tells whether the authentication properties are satisfied for all executions of the 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. However, [5] illustrates that this is not a big problem in practice.

3.2 Modelling Protocols in UML

To model security protocols in UML consistently, we have defined a specific profile [6]. 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 ForLySa extractor [6] can feed the LySatool analyser [2], the designer exploits the stereotypes in a class diagram to present the structure of the protocol. This involves first of all specifying the intended communications and the involved messages. The structure of each message type is specified in a distinct diagram that includes the decorations needed to specify the authentication property. Then, the local information of each principal must be introduced, like session keys or temporary storage, and their operations to build and dissect messages.

Then, the designer presents the dynamics of the protocol in a sequence diagram, which formally specifies a canonical run of the protocol. Each message exchange in the protocol is divided into three steps: 1. the sender packages the message, 2. the message is communicated, and 3. the recipient processes the incoming message. Each step is described by one or more UML stimuli in the sequence diagram, each associated to an operation of their target. Each operation is specified by pre- and post-conditions, for instance to specify how to decrypt part of a message, what to check in an incoming message, or what to store for later usage in the principal. The simple ad-hoc language [6] used in these conditions is presented to the designer with an informal semantics in term of the UML modelling concepts. This semantics intuitively reflects the precise one given by the translation in LySa.

Finally, the places mentioned by the authentication properties are specified as notes associated with the stimuli in steps 1 and 3 above, to provide the necessary hooks for the feedback from the LySatool. 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

Well-engineered, safe systems need to deliver reliable services in a timely fashion with good availability. For this reason, we view quantitative analysis techniques as being as important as qualitative ones. The quantitative 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]. 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].

4.1 Analysis process

We automatically generate a Continuous-Time Markov Chain (CTMC) from the PEPA model and solve it for its equilibrium probability distribution using procedures of numerical linear algebra such as the pre-conditioned biconjugate gradient method or successive over-relaxation implemented in the PEPA Workbench. 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, requiring the modeller to abstract in order to reduce model complexity.

4.2 Representing model components in UML

Markov chain modelling is based on finite-state representations of systems. The requirement to generate a finite state-space for the CTMC leads PEPA models to be structured as a concurrent composition of finite-state sequential processes. This led to a natural representation of the sequential process part of these models within the UML via the use of *state diagrams*, a variant of Harel's statecharts [10], together with a class diagram for each category of component. To represent a concurrent composition of those we used a *collaboration diagram* to specify an operational configuration of the system with some numbers of instances of each class of component synchronising over the activity names which they had in common. This diagram type provided the concurrent composition of the sequential components.

Class diagrams are used for other purposes in the model. A class with the reserved name *Rates* is used to store the values of the rate variables used in the model to quantify the time cost of performing any activity in the model. All activities are timed, and quantified by a rate variable which governs a negative exponential distribution, as used throughout Markovian modelling.

5 Extraction and reflection

Process calculus content is automatically extracted from input UML models and analysis results are automatically re-integrated into UML models. The categories of software tools which perform these operations are *extractors* and *reflectors*, which we describe briefly here.

The transport format for UML content is XML in the XML Metadata Interchange format (XMI) used for exchanging UML models between UML tools. Our extractors and reflectors are implemented in the Java programming language using its native API for XML parsing. Before the XMI format of the model can be processed, it must first be retrieved from the archive format of the UML tools which we support (primarily the Poseidon [1] tool from Gentleware). We have written data loaders for the NetBeans platform which open these archive files to find the XMI content inside, and correspondingly close such archives.

The extractors traverse the object instance graph of the XML document following the UML metamodel structure to retrieve the diagram content of relevant type. This graph traversal involves following cross-references within the XMI content to find (for example, class diagrams referenced by a collaboration

diagram, or state diagrams associated with a class, or the local states within a state diagram).

The tree traversal performed by the extractors inspects the tree only, without modifying it. In contrast, the tree traversal performed by the reflectors modifies the tree to update states with additional analysis results, adding or modifying child elements of the model as necessary. Finally, this passes the modified XML tree to the output routines of Choreographer for serialisation and archival.

6 Example: checking a simple authentication protocol

As a simple example, we apply Choregrapher to analyse variations on the Wide-Mouthed-Frog protocol, originally presented in [11].

The 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}.

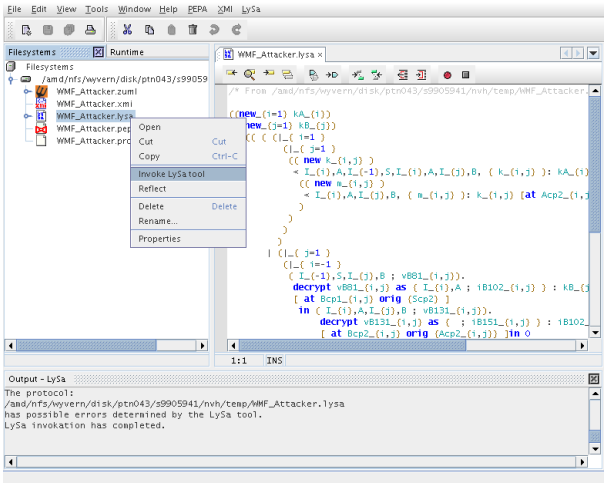

Fig. 2. Invoking the LySatool on a security model

The first step in checking such a protocol with Choregrapher is to formalise the protocol in a UML model, using primarily a sequence diagram to express

the protocol. We then open this UML model in Choregrapher and extract a LySa process calculus representation of the protocol, and apply the LySatool to analyse the flow of information through the protocol as shown in Figure 2.

The LySatool detects errors in the protocol, so we return to the Poseidon UML modelling tool to investigate the errors as described in the UML context. The output from Choregrapher is a modified version of the UML input. We have circled the changes in the screenshot in Figure 3 to indicate the differences from the input.

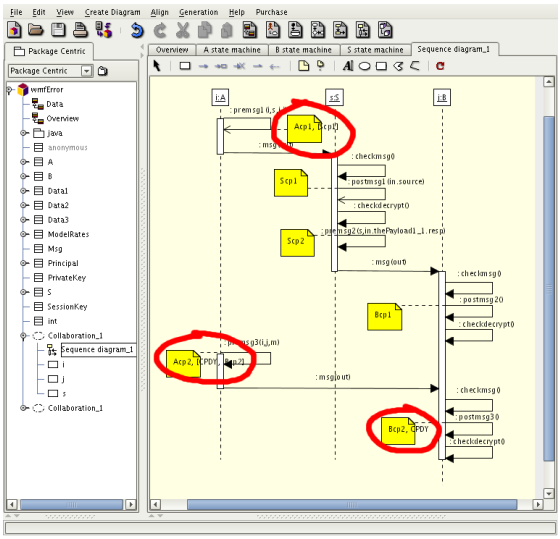

Fig. 3. Viewing the results from the LySatool in Poseidon

The LySatool identifies points in the protocol where the attached assertions may be violated. From this the modeller can modify the protocol description in UML and re-run the analysis until the analyser is no longer able to find errors in the protocol.

At this point the user is able to continue with a performance analysis of the model. Again, the process calculus representation is extracted by Choregrapher from the UML model and processed by the analysis tool (in this case, the PEPA Workbench). The Workbench derives the reachability graph underlying the process algebra model, interprets this as a continuous-time Markov chain and computes the stationary probability distribution for this chain. The commentary from the Workbench on this calculation can be seen in the tabbed pane at the bottom of the screenshot in Figure 4. These results can again be reflected back to the UML level.

Viewed in the Poseidon modelling tool, the results of the analysis tell the user the probability of being in each of the local states of each of the components of the

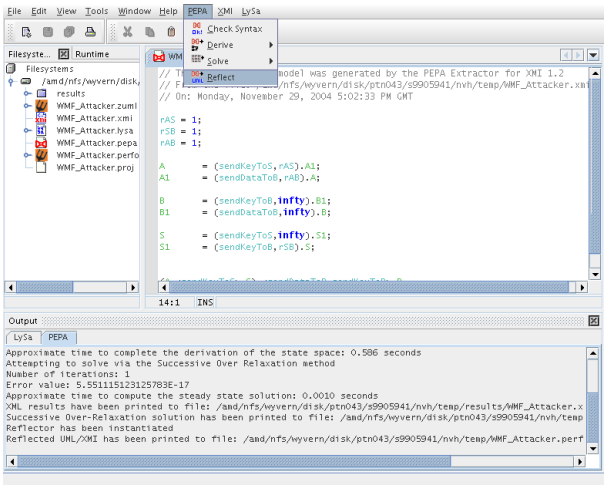

Fig. 4. Reflecting the results of performance analysis back to UML

process algebra model. Each such component has been described by a UML state diagram, and a UML collaboration diagram has described a parallel composition of a number of instances of these sequential components. A screenshot of the Poseidon modelling tool describing this information can be seen in Figure 5. The changes from the input state diagram are again circled. Each state now is tagged with a record of the probability of being in this state in the long run.

At this point the modeller is able to consider the consequences of these relative probabilities and to decide whether or not they indicate acceptable levels of performance with respect to these rates of performance of the activities of the model.

7 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

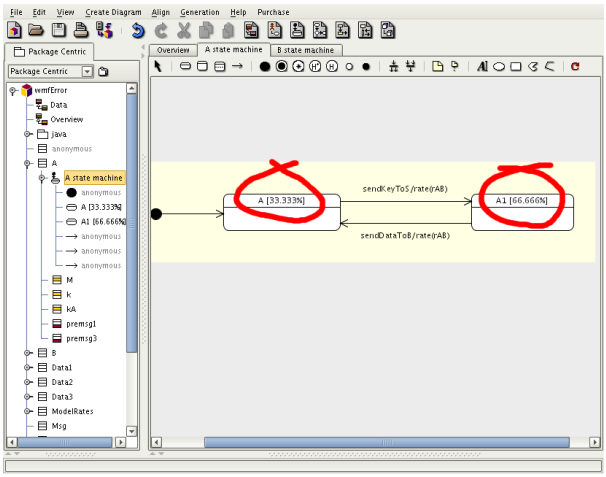

Fig. 5. A Poseidon screenshot of the modified performance model with the changes circled

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.

8 Experiences of the industrial users

The Choreographer design platform has been used by the industrial partners in our project on two separate developments. In the first, the partner was a large multi-national company designing a web-based micro-business portal. In the second, the partner was a small developer targeting telecommunications services designing a multi-player on-line role-playing game for mobile applications. Both used the Choreographer platform independently, consulting us when they had problems but otherwise operating without an expert on formal specification or verification on-hand.

The industrial partners had no previous experience of using the LySatool and the PEPA Workbench and their use of them was solely via the Choreographer extraction/reflection discipline. The most significant potential sources of error along the tool chain are i) in the UML constructs used in the input model; ii) communication of the UML model from the UML tool to the extractor; iii) model exchange between the extractor and the process calculus analyser; iv) in the use of the analyser; and v) from the analyser to the reflector. Almost all of the problems reported by our industrial users were of type i), ii) or iii).

Errors of the first kind included choosing the wrong type of connections between class instances in the collaboration diagrams, or omitting to include collaborations between instances of classes which needed to collaborate. Errors of the second kind included the UML tools writing non-well-formed archive files with missing or corrupt XMI content. Errors of the third kind are found because the process calculus analysis tools check the well-formedness of the model before continuing with the analysis. This is to ensure that as many problems as possible with the model are caught before a potentially expensive analysis process begins.

8.1 Reflections on the experience

Our anticipation of the difficulties for the industrial users was quite far removed from the actual difficulties encountered. The fact that many of the errors were

related to UML processing surprised us. We had assumed that the asymptotic complexity of the analysis procedures used in performance analysis would be a problem for models of industrial scale. The PEPA Workbench uses sparse matrices to store CTMCs internally but other representations such as multi-terminal binary decision diagrams (MTBDDs) allow the representation of much larger state spaces. Thinking that this would be a problem, we had previously developed a compiler from PEPA into the input language of the MTBDD-based model-checker Prism [12]. We developed a custom reflector for this tool which we tested with the PEPA extractors and reflectors to analyse our own UML models [13].

In fact, the state-space explosion proved not to be a problem for the use of Choreographer by our industrial partners. The models which they built were much smaller than we had anticipated. So much so, that using sparse matrices is perhaps not even necessary and dense matrices and direct solution methods might even have been applicable.

9 Related work

Tool support for the automated analysis of security requirements in the UMLsec framework [14] is described and accessible at [15]. 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 [16] 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 [17]. 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 CTMC-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 [18] 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 [19].

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 interms 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.

10 Conclusions

Strong and justified belief in the networked software applications is engendered via formal analysis using well-founded calculi and tools. Such apparatus for formal reasoning is often daunting to those who most need to make use of, and benefit from, formal analysis techniques, namely systems designers and software developers working on state-of-the-art systems. To this community, and their colleagues in project management and product development, a graphical notation such as the UML has much greater appeal than the blunt, cold formality of process calculi. By establishing a two-way connection between the UML and calculi such as LySa and PEPA, the Design Environments for Global ApplicationS (DEGAS) project has elevated the analysis process to the UML level, thereby bringing the benefits of the analysis without exposing the unfamiliar languages used.

It is not the case that an inexperienced modeller can use the Choreographer platform to verify any security property of interest or to compute any performance measure that they wish without needing any understanding of the abstraction, modelling and mathematical analysis beneath. However, we hope that we have gone some way to providing automated support for useful security and performance properties and to circumventing an unnecessary notational hurdle if this was acting as an impediment to the understanding and uptake of modern static and dynamic analysis technology.

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 his extraction algorithm for PEPA. The work reported here builds on a number of prior works by the members of the DEGAS project. It is a pleasure to thank the other members of the project for their contributions and comments on the work reported here.

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. D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Programming*, 8:231–274, 1987.
11. M. Burrows, M. Abadi, and R.M. Needham. A logic of authentication. *ACM Transactions on Computing Systems*, 8(1):18–36, February 1990.
12. D. Parker. *PRISM 1.3 User's Guide*. University of Birmingham, February 2003. http://www.cs.bham.ac.uk/ dxp/prism.
13. S. Gilmore and L. Kloul. A unified tool for performance modelling and prediction. In B. Littlewood S. Anderson and M. Felici, editors, *Proceedings of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP 2003)*, volume 2788 of *LNCS*, pages 179–192. Springer-Verlag, 2003.
14. Jan Jürjens. *Secure Systems Development with UML*. Springer, 2004.
15. Jan Jürjens. Umlsec webpage. Accessible at http://www.umlsec.org, 2002–04.
16. 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.
17. 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.
18. 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.
19. 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.