Choreographing security and performance analysis [*]

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

L. Kloul
PRiSM, Université de Versailles
45, avenue des Etats-Unis
78000 Versailles
kle@prism.uvsq.fr

M. Maidl
Siemens AG, CT IC3
Otto-Hahn-Ring 6
81739 München
monika.maidl@siemens.com

ABSTRACT

We present a novel method of assuring security and performance demands on systems based on automated analysis of UML model descriptions. Analysable content is extracted from the UML models in the form of process calculus descriptions. These are analysed to provide strong guarantees of satisfactory security and performance. The results are reflected back in the form of a modified version of the UML model which highlights points of the design which can give rise to operational difficulties. A design platform supporting the methodology, *Choreographer*, interoperates with state-of-the-art UML modelling tools such as Poseidon. We illustrate the approach on an example provided by our industrial partner.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: To do...; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

UML, performance modelling, communications protocols, static analysis

1. INTRODUCTION

The design of modern networked applications provides a difficult engineering challenge for those application developers who work with the programming models and APIs of fully-featured development platforms. The ever-present wish to make services and applications accessible over the network strains developers' abilities to build secure systems which will stand against a wide range of possible attacks. Another challenge is meeting the performance requirements demanded by users. Applications which are built from reusable high-level components provide real productivity increases for the development of secure, feature-rich systems but such components rest on layers of middleware and must cooperate with security infrastructures on virtual machines and physical machine networks. From an application performance point of view, the effect of these layers of support and coordination is to dampen the responsiveness of the system and cap its peak efficiency. In this way, security and performance concerns are often at variance and developers struggle to satisfy both requirements while also working with ever-changing software development kits, implementation platforms, management technologies, and APIs.

In the design of challenging applications the greatest profit is gained by considering security and performance requirements early in the design phase, identifying problems before they are built into the system implementation. If such problems are not identified at the design stage then they might be found by users when the application is in deployment, perhaps leading to operational problems with significant financial or legal consequences.

To help to address the problems of designing secure systems which provide strong guarantees of adequate performance we have designed and implemented a sophisticated UML-based design platform. This design platform, named Choreographer, interoperates with standard UML tools such as the Poseidon architecture and guides developers through a design methodology centered on a UML project. Choreographer directs developers in the application of three categories of utilities: extractors, analysers, and reflectors. Together these provide access to powerful analysis techniques which operate on the UML models which they have created, elevating the analysis to the level of the UML model. The Choreographer design platform has a flexible and extensible plug-in architecture which supports its extension with additional analysis tools, offering other qualitative or quantitative analysis possibilities.

The contribution of this paper is to present a UML-based methodology for integrated security and performance analysis, supported by a well-engineered tool and set on the formal foundation of dedicated process calculi with custom analysers. We describe the UML-based methodology which Choreographer supports and discuss the implementation of the Choreographer platform itself. We give an example of its use on a typical UML project.

The paper is structured as follows: the methodology is described

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

WOSP '05 Palma de Mallorca, Illes Balears, Spain

in Section **??**. The Choreographer analysis tool is presented in Section **??**. The example application is a web-based micro business, described in Section **??**. This is followed by a UML model and its associated performance and security models in Sections **??**, **??** and **??**. We discuss related work in Section **??** and conclude in Section **??**.

2. METHODOLOGY

The methodology which we seek to support uses a range of UML diagram types to express the security and performance considerations of the system. As a principle, we use standard UML notation: there are no notational extensions or additional diagram types. This decision has two beneficial consequences. First, a UML modeller using this methodology does not need to learn any supplementary notation. Second, we are able to use standard UML tools such as Poseidon [**?**] to edit the UML diagrams which we use.

We use class diagrams, collaboration diagrams, sequence diagrams and state diagrams to describe the system under study in UML terms. Additional diagram types may be used in the UML project which is accepted as an input to Choreographer. These can be added as descriptive material documenting other aspects of the system, such as user modelling, and will not interfere with the analysis process.

The performance and security analyses which are supported within the methodology are fully automatic, so that there is no need for user intervention at any stage throughout the process between commiting the input model for analysis and inspecting the results.

Different models can be used for different purposes in the design of an application and so the methodology supported by our design platform allows modellers to either do a security analysis alone, or a performance analysis, or both. That is, the annotated versions of models which result from one run can be used again as inputs to Choreographer to perform a different type of analysis. The consequence of this is that a modeller using an established operational procedure to determine satisfactory levels of security (resp. performance) can use our design plaform to do performance (resp. security) analysis alone. They are not forced to adopt both of the kinds of analysis which we offer if they do not need both, or already have a preferred way to do one of them.

3. CHOREOGRAPHER

One feature of the methodology which we support with the Choreographer design platform tool is that modellers are able to express the models which are input to Choreographer in standard UML. The analysis process is initiated by invoking Choreographer on a UML project archive. The formal content of the UML model is stored in such an archive in an XML-based interchange representation (XMI). Software connectors termed *extractors* process the XMI representation of the input model and derive an analysable form of the model expressed in a process calculus. We use different process calculi for security and performance analysis: LySa [**?**] for the former and PEPA [**?**] for the latter.

Another key feature of the method is that the results of the analysis are reflected back as a modified version of the original UML model. The *reflectors* which do this are also available as software components which take the original UML project and the results of the analysers as inputs and write their results as complete UML projects in which the results of the analysis have been incorporated. The purpose of this is to ensure that the interpretation of the analysis results can be undertaken at the UML level and that the UML is not being used only as a model description language from which a process calculus representation is generated. Instead, reflecting the results back to the UML level centres the analysis on the UML description.

3.1 Design

The Choreographer platform is designed to support UML-centered development but is flexible enough to accommodate other modes of use in addition. These might simply be preferred by designers or developers who are using the platform or they might be needed to support a style of development favoured by the institution or software house which commissioned the development. Thus, a guiding principle of the design of Choreographer is that the processing of UML models should be made visible to the developer in order that the mapping between UML diagram elements and constructs of the process calculi beneath is transparent. This principle ensures that modellers have access to the representations which are needed to understand how their diagram elements are interpreted in the analysis process.

Exposing the functionality of the extractors and reflectors which map UML models to process calculi and analysis results back into the UML brings additional benefits for both implementors and users. We detail these in turn.

For the implementors the benefits of this design include having the functionality of the extractors open to inspection and enquiry. This greatly assisted with the problem of finding and correcting programming errors in the implementation of the extractors, with commensurate benefits to their implementation quality. The design of the analysis tools further supports the quality of the implementation of the extractors. The interface to each of the analysis tools is a formal language which is throughly checked by the tool before the analysis is carried out. Considering the performance analysis tools, models with deadlocks or missed synchronisations are rejected by the analysis tool, identifying errors in the input UML model or the implementation of the extractor which generated the process calculus representation.

For the users of the platform this has the benefit that it facilitates experimentation with models at the level of the process calculus, allowing a single UML model to give rise to a related collection of process calculus models. Here the UML level provides a much-needed layer of abstraction over the formal details. Such an abstraction would be useful in presenting a simplified—but consistent and accurate—view of the model to project managers or customers for whom the full complexity of the detailed process calculus representation would be a barrier to understanding.

3.2 Implementation

Sophisticated users demand more from applications than raw functionality alone. The functionality must be presented in an appealing package if it is to attract a substantial user base and build a community of enthusiastic adopters. The issues here include engineering concerns which complement the scientific ones. Ignoring the engineering problems would be an unwise practise because the benefit to be derived from scientifically well-founded analysis will not be obtained if the analysers are not used. Within the engineering concerns are portability issues, user interface design and ease of installation. Modest problems in each of these areas would not deter the most enthusiastic adopter but could prevent an interested potential evaluator from trying the Choreographer platform in the first place.

To this end we worked to minimise engineering annoyances by building on portable, well-engineered implementation technology. Reasoning that we wanted to build an integrated development environment we researched a range of generic IDEs including Eclipse [**?**] and NetBeans [**?**]. Both of these build on the Java platform and provide generic user-interface and editor components, filesystem explorers, and other support for integrated development environments. Either could have been used for our purposes but we chose NetBeans because *insert plausible reason here*.

NetBeans associates *data loaders* with file types and the functionality implemented in a data loader determines the processing which can be performed on the file contents. We implemented data loaders for each of the modelling languages used in our development method: UML (specifically, in its XMI format), PEPA and LySa.

By associated the necessary functionality with a language in this way, the Choreographer design platform guides the user through the modelling process. UML projects can be opened to obtain their XMI content. Extraction can be applied to the XMI representation to obtain a PEPA or LySa model. Performance analysis can be applied to PEPA models. Security analysis can be applied to LySa models. Results are reflected back into the UML representation. Choreographer guides the user through the process (exraction, then analysis, then reflection) and prevents “mode errors” such as trying to apply security analysis to a PEPA model, as could occur in a less structured modelling approach.

programmed menus using NetBeans’ *XML layers*

4. THE WEB-BASED BUSINESS SYSTEM

The case study provided by our industrial partner is a web-based service to enable e-business based on a peer-to-peer authentication and communication paradigm. The objective of this system is to provide a support to micro web-based businesses that do not have by themselves the capability of developing proprietary solutions for e-business [**?**]. The system is accessible through both wired internet connections and mobile devices using standard protocols such as the wireless application protocol. The system will present the various services offered by the service providers according to a coherent layout and will be provide an interface for service access. While users should be able to process their transactions on a peer-to-peer basis, it is necessary to provide a central portal at which users register and can search for services. Hence the system naturally decomposes into three parts: the portal, services providers and customers (Figure **??**). The upper part of Figure **??** describes the part of functionality that involves the portal, while the lower part concerns the peep-to-peer functionality.

The portal

It enables remote data search and service navigation. Moreover it can constitutes the interface between the customers and the service providers during the on-line business transactions. The e-business data management provides access to distributed products and services catalogues. The portal allows an important number of concurrent accesses whereas providing an end-to-end security of the transactions.

The service provider

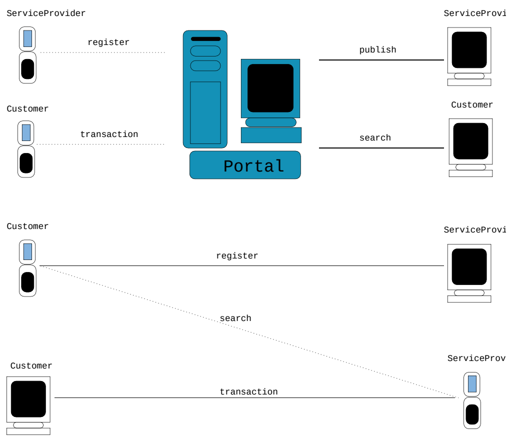

Figure 1: Architecture of the web-based business system

A new service provider joining the system has first to register itself to be known by the portal. A registered service provider has then the possibility to publish dynamically its services onto the portal. The list of its services can be accessed by any customer through the portal. Each provider will be able to modify its published services list by adding a new product, changing the characteristics of an existing one or removing a service from the list. At any moment, a service provider can quit the system by unregistering from the portal. The service provider can also handle transactions directly with customers who have registered at the service provider.

The customer

Like the service providers, new customers have to register at the portal before being able to use its services. The registered customers are informed by the portal about available services, the newly published services and the modified or removed ones. The user may perform on-line transactions via the portal to buy products he is interested in by selecting them from the list. The customers order requests are then routed by the portal to the appropriate service provider. Alternatively, a customer can choose to communicate peer-to-peer with a chosen service provider after registering directly at this service provider.

4.1 Performance and security requirements

In order to provide an attractive service it is important that the response time of the portal is limited: The high-level requirements state that a user request via mobile phone should be served within 10 seconds while a request via the internet should be served with in 5 seconds on average. The security of the system is also crucial: Correct authentication at the portal and between service providers and buyers is required to prevent misuse by identity theft. The user data, in particular the transactions between service providers and customers, and also information associated to the operations to a user on the portal, should remain confidential, and integrity should be preserved. As all these data are transmitted wireless or via the web, strong security measures are required to meet these requirements.

5. UML MODEL OF THE SYSTEM

6. THE PERFORMANCE MODEL

The on-line transaction system can be modelled using three PEPA components. The first component *Portal* models the behaviour of the interface between the service providers and the customers. The second component *Provider* models any provider registred in the system whereas the last component *Buyer* is used to model the behaviour of a customer.

Note that in this model, we assume that both buyers and providers are already know by the system; they have already registred themselves.

Component *Buyer*
In an on-line transaction, the system user starts by sending a request to the portal about a specific product he is interested in like books for example. This can be done by a simple clic on the icon titled "Books" in the main pages of available products provided by the portal. This is modelled by action type *new_request*. The response of the portal is to send to the customer the catalogue or list of books available with all characteristics. We model this using action type *get_product_list*. Once the customer has the targetted list, he can select all the items he wants (action *select_product*) and then go to the check out (action *check_out*). This last step allows the buyer to place an order for selected items. At any moment the customer can change his mind and stop the process. This is modelled using action type *restart*. Note that action type *get_product_list* has an unspecified rate in component *Buyer* because the rate is defined by the portal which will send the list of products at his rythm.

$$\begin{array}{lcl}
Buyer & \overset{def}{=} & (new\_request, r).Buyer_1 \\
 & + & (update\_request, \top).Buyer \\
Buyer_1 & \overset{def}{=} & (get\_product\_list, \top).Buyer_2 \\
Buyer_2 & \overset{def}{=} & (select\_product, r_1).Buyer_3 \\
 & + & (restart, r_2).Buyer \\
Buyer_3 & \overset{def}{=} & (select\_product, r_1).Buyer_3 \\
 & + & (restart, r_2).Buyer \\
 & + & (check\_out, r_3).Buyer
\end{array}$$

Component *Provider*
Once a service provider is registered, he may either send a request to the system to update the list of products or services he has published or receive an order from the portal. The former is modelled using action type *update_request* and the latter action type *transmit_order*. In the first case, he will receive the list of services he owns (action *get_own_list*) and can then make all the changes he wants using action types *add_product*, *delete_product* and *change_values*. Once he is finished with the updates he can leave the system (action type *quit*). In the second case, he will consider the customer order and do the necessary to satisfy the request. This is modelled using action type *process_order*.

$$\begin{array}{lcl}
Provider & \overset{def}{=} & (update\_request, s).Provider_0 \\
 & + & (transmit\_order, \top).Provider_2 \\
Provider_0 & \overset{def}{=} & (get\_own\_list, \top).Provider_1 \\
Provider_1 & \overset{def}{=} & (add\_product, s_1).Provider_1 \\
 & + & (delete\_product, s_2).Provider_1 \\
 & + & (change\_values, s_3).Provider_1 \\
 & + & (quit, s_4).Provider \\
Provider_2 & \overset{def}{=} & (process\_order, s_5).Provider
\end{array}$$

Component *Portal*
The portal manages both the buyers and the providers. All activities of component *Portal* are synchronizing activities, either with the buyers or the providers.

$$\begin{array}{lcl}
Portal & \overset{def}{=} & (new\_request, \top).Portal_1 \\
 & + & (update\_request, \top).Portal_3 \\
 & + & (select\_product, \top).Portal_1 \\
 & + & (restart, \top).Portal \\
 & + & (check\_out, \top).Portal_2 \\
 & + & (get\_product\_list, v_1).Portal_1 \\
Portal_1 & \overset{def}{=} & (get\_product\_list, v_1).Portal_1 \\
 & + & (select\_product, \top).Portal_1 \\
 & + & (restart, \top).Portal \\
 & + & (check\_out, \top).Portal_2 \\
 & + & (new\_request, \top).Portal_1 \\
Portal_2 & \overset{def}{=} & (transmit\_order, v).Portal \\
 & + & (select\_product, \top).Portal_2 \\
 & + & (restart, \top).Portal_2 \\
 & + & (check\_out, \top).Portal_2 \\
 & + & (new\_request, \top).Portal_2 \\
 & + & (get\_product\_list, v_1).Portal_2 \\
Portal_3 & \overset{def}{=} & (get\_list, v_2).Portal_3 \\
 & + & (add\_product, \top).Portal_3 \\
 & + & (delete\_product, \top).Portal_3 \\
 & + & (change\_values, \top).Portal_3 \\
 & + & (quit, \top).Portal
\end{array}$$

The complete system: The behaviour of the actors of the online system and their interactions between each other are captured by component $web\_business$ which is defined as follows:

$$\begin{array}{ll}
web\_business & \overset{def}{=} (Buyer \underset{K}{\bowtie} Buyer \underset{K}{\bowtie} ...Buyer) \underset{L}{\bowtie} \\
 & \big((Provider|| \ldots ||Provider) \underset{M}{\bowtie} Portal\big)
\end{array}$$

where the synchronising sets are defined as follows:

$$\begin{array}{lcl}
K & = & \{update\_request\} \\
L & = & \{new\_request, get\_product\_list, select\_product, \\
 & & restart, check\_out, update\_request\} \\
M & = & \{update\_request, get\_own\_list, transmit\_order, \\
 & & add\_product, delete\_product, change\_values, \\
 & & quit\}
\end{array}$$

Remark: The use of action *update_request* in component *Buyer* ensures that during the updates of a product list by its owner, the buyers do not have access to this list. As all components of the model must synchronise on *update_request*, it will not be enabled unless all occurrences of component *Buyer* are in their initial state.

6.1 Numerical results

In this section we give an idea of the performance measures that we can compute in the context of such an application. We are mainly interested in the throughput of the portal. We consider a system composed of five buyers and one provider. This simple system allows us to already have an idea of the behaviour of the throughput in a system with a portal based architecture. All curves are plotted as a function of the arrival rate r of the requests of one buyer.

- Figure **??** depicts the total throughput of the portal in terms of buyers requests to get a product list and to select a product from a list, and the provider requests to get its own list. This figure gives also the throughput part related to the transmission of the orders to the provider. As we can see, the transmission of the buyers orders is a very small part of the throughput of the system. This may be explained by the fact that the buyers spend time selecting product. Moreover once the items selected, a buyer may decide to abandon or restart. Thus all buyers do not end up making the checking out process.

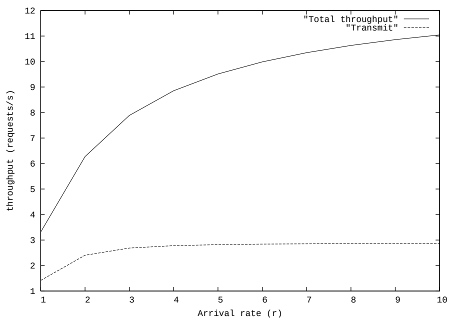

Figure 2: Total throughput

- Figure **??** shows the behaviour of the part of the portal throughput related to the provider requests (*get_own_list*). Unlike what we have seen in Figure **??**, this throughput decreases as the arrival rate increases. As we have more requests from the buyers, the portal spends more time dealing with these requests, and thus less time with the provider requests.

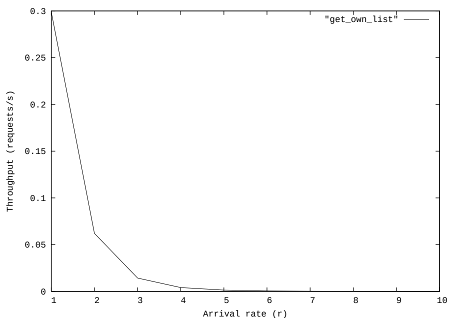

Figure 3: Throughput for provider requests

7. SECURITY ANALYSIS

The security of a networked service depends heavily on the ability of users to send confidential messages via wireless or Internet connections, and to confirm the identity of the partner in their message exchange. Cryptographic techniques are usually used both to ensure the confidentiality of messages.

But cryptography is not a magic wand to make everything all right. The main issue is that sending encrypted messages is only safe if only the authorized parties have the corresponding key. So data security becomes a key management problem [**?**], and the main task consists in designing an appropriate protocol for *a*uthenticated key exchange. Such a protocol allows two or more participants to exchange a cryptographic session key in a way that the participants are assured that only the intended parties obtain the session key. Confidentiality and integrity of data is then guaranteed by encrypting all data with the session key. The main tool for providing proper authentication in such a key-exchange protocol is again cryptography, and hence an analysis tool must be able to deal with cryptography concepts. Before describing the tool LySa [**?**] that is used by Choreographer, we first discuss the security requirements of the web-based business system, and show the key exchange protocol chosen for the project.

7.1 Security analysis for the web-based business system

In the case study, all communication should be encrypted to guarantee data confidentiality and integrity. This means that before starting a data exchange, a service provider and a customer or the portal and a user have to use a protocol for authenticated session key exchange.

For this protocol, there is a choice between using either symmetric cryptography or public key cryptography in a protocol for authenticated key exchange. When using symmetric key cryptography, the communication has to be conducted via a central server, and all users have to share initial symmetric keys with the server. The design goal of the project of providing peer-to-peer communication between service providers and customers would be violated if communication between users necessarily involved a central server. Moreover, initial distribution of secret symmetric keys is difficult to achieve in a practical way. Hence a protocol based on public key cryptography is used. In order to link a user identity U to a public key, it is essential to use certificates $cert_U$, which are signed by some trusted certification authority.

(1) A $\rightarrow$ B: A, $cert_A$
(2) B $\rightarrow$ A: $\{$B, NB$\}$:K_A^+, $cert_B$
(3) A $\rightarrow$ B: $\{$A, NB, $K_{AB}\}$:K_B^+

The aim of the protocol is to provide authenticated key exchange between A and B, i.e. after the exchange both A and B are assured that only they know the new session key K_{AB}. More precisely, correct authentication is achieved by the protocol if A can be sure that message (3) can only be decrypted by B, while B knows that message (3) can only be sent by A.

7.1.0.1 LySa model of the protocol

The informal notation of the protocol used above leaves implicit a number of assumptions and does not completely describe the internal actions taken by the principals like decrypting with a certain key, comparing nonces and checking certificates. Moreover it is crucial to specify the environment in which the protocol is executed, i.e. the actions that potential attackers can perform.

For a formal analysis, these assumptions have to be specified, and LySa provides a format for that, which is essentially a process algebra, enriched by cryptographic notions such as encryption and decryption, symmetric keys, public and private keys, so that it is possible to model an authenticated key exchange protocol. More precisly, LYSA is based to the π-calculus. The main difference to the π-calculus and the Spi-calculus is that there are no channels: messages can be arbitrarily intercepted and redirected. Moreover, pattern matching is used to check that a message contains expected values (like nonces), and to bind values to free variables. Each participant in the protocol (in our case A and B) are modelled by a separate process. Each message of the protocol corresponds to two actions: one performed by the sender who encrypts and sends the message, and one performed by the receiver, who decrypts the message, checks the content and might store parts of it.

As example, consider message (3) of the protocol, which is sent from A to B. The Lysa code for sending, which forms part of the process for A, is shown next; sending of messages is denoted by $\langle \ldots \rangle$.

$$(\text{new}K_{AB})\langle A, B, \{|A, vNB, K_{AB}|\} : \mathrm{K}_B^+\rangle$$

The first argument in the $\langle \ldots \rangle$ expression denotes the sender, the second the recipient and the rest is the content of the message, which in this case consists of only one part, which is encrypted. The terms are either names, like A, B, K_{AB}, or variables, like vNB, which has been bound to the value of NB when A received message (2). Sending of message (3) is preceeded by generating a new session key K_{AB} that nobody except A knows. This is modelled by restriction with the 'new' operator.

Input of a message is denoted by $(\ldots)$; we show the receiving action associated with (3), which is performed by process B:

$$(A, B; x).\text{decrypt } x \text{ as } \{|A, NB; vK|\} : \mathrm{K}_B^-$$

An incoming message is matched with an output, whereby the terms before the semicolon have to match while the variables after the semicolon are bound to values after successful matching. Accordingly, the first term denotes the sender and the second term denotes the recipient of the message. Encrypted terms are bound to a free variable and decrypted in the next step. Again pattern matching is applied to the content of an encrypted message. In the example, B only accepts the message if the first argument is A, and the second is the nonce NB which B has chosen for message (2). Note that B has to decide with which key to decrypt the message; for message (3), this is the private key K_B^-.

It does not suffice to code only one session between A and B, because attacks might require parallel sessions in which A or B (or both) participate. LySa offers the possibility to parameterize the protocol by n to code $n+2$ participants $I_{-1}, I_0, I_1, \ldots, I_n$, where $I_1, \ldots, I_n$ are the legitimate participants, I_{-1} is a server (not present in our example) and I_0 models an attacker who masks as legitimate participant. As described, the protocol consists of two processes: the process for A and the process for B. In the LySa model shown in Figure **??**, every participant I_i can act either as A or B. Moreover, the replication operator ! indicates that any pair of participants perform an unlimited number of possibly concurrent sessions. The first line introduces the public/private keys of some certification authority, which are used to encrypt and verify certificates. In the second line, the public private key pairs $PK+_i$ $/PK-_i$ of all participants I_i are specified. The 'new' operator gurantees that these keys are not know to attackers. The attacker can read all messages and has hence access to all unencrypted parts apart from those restricted by 'new', and if in possession of the corresponding key, can also decrypt messages. Since the public keys are assumed to be publicly known, we have to make sure that potential attackers have access to them. This is done by sending public keys in the clear in the last two lines of the protocol. The attacker built into the Lysa model has the usual powers of the standard Dolav-Yao attacker [**?**], i.e. can use all information obtained from messages sent between participants to compose messages which can be sent to any participant. This means that a participant cannot be sure that the sender of a message is the one ocurring in the first argument of the message, as the attacker has access to all names and can hence fake the sender and recipient part of a message.

7.1.0.2 Security analysis with Lysa

The analysis performed by the LySa tool is to ask whether for multiple runs of the protocols between a number of participants, and in the presence of a standard (Dolev-Yao) network attacker, correct authentication is guranteed. LySa has been designed to verify correct authentication, and can also check confidentiality of data. The analysis of correct authentication is based on the use of assertions, which annotate the points in the protocol at which encryption and decryption takes place ('cryptopoints'). At an encryption point these assertions specify the destinations where it is believed that the complementary decryption can occur. At a decryption point the assertions specify the points where it is believed that the complementary encryption occurred.

For the key exchange protocol of the web-based business system, the LySa assertions specify that message (3) is correctly authenticated. More precisely, sending of message (3) is annotated with [at $a3_{i,j}$ dest $b3_{i,j}$] while receiving of message (3) has annotation [at $b3_{i,j}$ orig $a3_{i,j}$], see Figure **??**.

Hence, the assertions state correct (mutual) authentication of the communicating parties. The LySa tool checks whether an attacker is able to impersonate a legitimate participant and hence violate correct authentication. If the analysis shows that all assertions are correct in the presence of an attacker, we learn that the protocol guarantees correct authentication.

We have analysed the key exchange protocol for the web-based business system with LySa and shown that it provides authenticated key exchange. Moreover, we experimented with variants of the protocol and showed that omitting data from messages in the protocol makes it insecure. As an example, we show an attack that is possible when omitting the name A in message (3):

(1) A $\rightarrow$ B: A, *cert*$_A$
(2) B $\rightarrow$ A: {B, NB}:K_A^+, *cert*$_B$
(3) A $\rightarrow$ B: { NB, K_{AB}}:K_B^+

After A has started a regular session with B, the attacker I starts a parallel session with B, and afterwards sends the response of B instead of the second message in the first session. Then the intruder intercepts the response of A in the first session and uses it as message (3) in the second session.

(1)	A $\rightarrow$ B: A, *cert*$_A$		
		(1')	I $\rightarrow$ B: I, *cert*$_I$
		(2')	B $\rightarrow$ I: {B, NB' }:K_I^+
(2)	$\mathrm{I}_B \rightarrow$ A: {B, NB' }:K_A^+		
(3)	A $\rightarrow \mathrm{I}_B$: { NB', K }:K_B^+		
		(3')	I $\rightarrow$ B: { NB', K}:K_B^+

The result is that K is the new session key for the session A thinks she is conducting with B as well as for the session between B and I. This means that I can intercept messages encrypted by A with the key K_{AB} and make B believe that the message comes from I.

8. RELATED WORK

```
(new +- KCA)(
(new_{i=1} +- PK_{i})(
      /* process A */
(|_{i=1} |_{j=0\i} !(
        /* send (1) */
   <I_{i},I_{j},{|I_{i},PK+_{i}|}:KCA->.
        /* receive (2) */
   (I_{j},I_{i};v1_{i,j},vcertB_{i,j}).
   decrypt v1_{i,j} as {|I_{j};vNB_{i,j}|}:PK-_{i} in
   decrypt vcertB_{i,j} as {|I_{j};pB_{i,j}|}:KCA+ in
        /* send (3) */
   (new K_{i,j})(
   <I_{i},I_{j},{|I_{i},vNB_{i,j},K_{i,j}|}:pB_{i,j} [at a3_{i,j} dest {b3_{i,j}}]>.0
   )))
|
      /* process B */
(|_{j=1} |_{i=0\j} !(
      /* receive (1) */
   (I_{i},I_{j};vcertA_{i,j}).
   decrypt vcertA_{i,j} as {|I_{i};pA_{i,j}|}:KCA+ in
      /* send (2) */
   (new NB_{i,j})(
   <I_{j},I_{i},{|I_{j},NB_{i,j}|}:pA_{i,j},{|I_{j},PK+_{j}|}:KCA- >.
      /* receive (3) */
   (I_{i},I_{j};x4_{i,j}).
   decrypt x4_{i,j} as {|I_{i},NB_{i,j};vK_{i,j}|}:PK-_{j} [at b3_{i,j} orig {a3_{i,j}}] in
   0)))
| |_{i=1} <PK+_{i}>.0
| <KCA+>.0
))
```

Figure 4: Lysa code for the security protocol in the web-based business system

- Gorrerieri
- UMLsec
- performance + UML stuff

9. CONCLUSIONS

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.

10. REFERENCES

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

[2] Dieter Gollmann. *Computer Security*. Wiley, 1999.

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

[4] F. Nielson, H.R. Nielson, H. Sun, M. Buchholtz, R.R. Hansen, H. Pilegaard, and H. Seidl. The Succinct Solver suite. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988, pages 251–265. Springer-Verlag, 2004.

11. METADATA

[This section to be removed before the paper is submitted]

This is a draft version of the paper: comments are welcome. Full papers submitted to the WOSP 2005 conference should be up to 12 two-column proceedings pages in the ACM style. The submission deadline is December 3rd. Find this paper's LaTeX source code and compiled versions on-line at `homepages.inf.ed.ac.uk/stg/papers/DEGAS/CHOREOGRAPHER`

The ACM style forces section headings to be in Times font. The default LaTeX font, Computer Modern Roman, looks very poor with this, in my opinion. Therefore, we should use a PostScript font, and the obvious choice is Times, which we are using here. An alternative would be New Century Schoolbook but not if we are short of space. Because of the two column style we are using "sloppy" spacing between words, and have discouraged hyphenation with a high hyphen penalty. These can be changed if they are causing offence. The ACM style uses American letter paper, of course.

11.1 Papers not yet cited:

[**?**]