Choreographing Security and Performance Analysis for Web Services

Stephen Gilmore[1], Valentin Haenel[1], Leïla Kloul[2], and Monika Maidl[3]

[1] Laboratory for Foundations of Computer Science, The University of Edinburgh, Scotland
[2] PRiSM, Université de Versailles, 45, avenue des Etats-Unis, 78000 Versailles, France
[3] Siemens AG, CT IC3, Otto-Hahn-Ring 6, 81739 München, Germany

Abstract. We describe a UML-based method which supports model-driven development of service-oriented architectures including those used in Web services. 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.

1 Introduction

Web services must deliver secure services to users in order that financial and other confidential transactions can be conducted without interference. Off-the-shelf solutions are not available. Web services need to build end-to-end security from the point-to-point security afforded by standard network protocols. Even if a secure system can be created, scaling up to large user populations provides a steep challenge. The availability of many different forms of assistance (caching, stateless session beans, process isolation and others) means that the challenge of building scalable systems is complicated further by difficult-to-quantify approaches to system performance tuning.

We have developed a design platform, *Choreographer*, which seeks to assist with the development of secure systems with quantified levels of performance. To provide an accessible entry point for practising Web service developers the methodology which we support uses the UML. This is a novel feature of our work: we use a modelling language where a specification language or process calculus might more often be used to initiate the analysis. Many UML designs are not analysed either qualitatively or quantitatively. Here we provide support for both types of analysis, and illustrate the value of the analysis via an example.

We use 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 [1] to edit the UML diagrams which we use.

M. Bravetti et al. (Eds.): EPEW 2005 and WS-FM 2005, LNCS 3670, pp. 200–214, 2005.

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 used for other purposes in model-driven development, such as automatic code generation, and will not interfere with the analysis process. Our aim is to disrupt existing model-driven development approaches as little as possible while adding value to the UML modelling work which would be going on in any case.

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 platform 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.

The original contribution of this paper is to present a UML-based methodology for integrated security and performance analysis. The method is 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 describe its use on a typical Web service creation problem: a Web-based micro-business. We believe that the Choreographer software tool could also be used for high-level analysis of other service-oriented architecture questions such as the assessment of service discovery protocols but we do not demonstrate this in the present paper.

Structure of This Paper: The paper is structured as follows: the Choreographer analysis tool is presented in Section 2. The example application is a web-based micro business, described in Section 3. This is followed by a UML model and its associated performance and security models in Sections 4, 5 and 6. Related work and conclusions follow.

2 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 [2] for the former and PEPA [3] 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.

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.

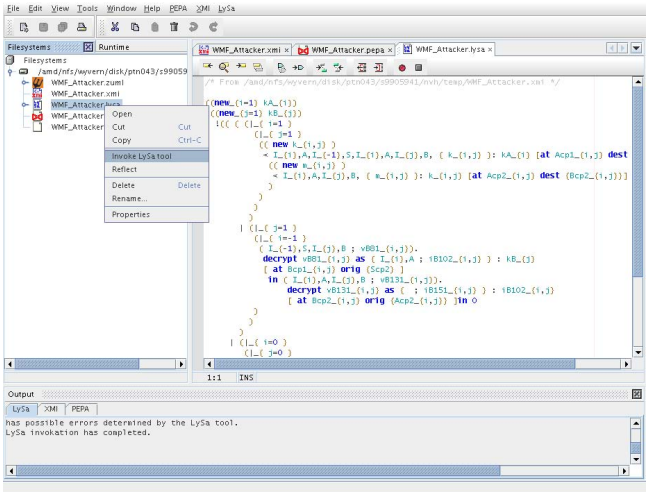


Fig. 1. The Choreographer user interface

In terms of its appearance, the Choreographer platform follows the conventional design of an IDE, as seen in Figure 1. The main design area divides into an explorer on the left, an editor on the right, and a message console beneath these. The explorer provides a view onto the local file system which is structured in order to group related documents into logical projects. The editor is language-aware with contextual modes: we have implemented editors for the process calculi which we use in the security and performance analysis process. The console is used to feed back to the user information about the progress of commands or analyses which have been launched from the application menus. Concise summaries of the analyses are printed into the console to allow information about the outcome to be obtained without having to initiate the reflection process and render the results in the Poseidon UML modelling tool.

3 The Web-Based Business System

The case study provided by our industrial partner is a business-to-business Web service to enable e-business based on a peer-to-peer authentication and communication paradigm. The objective of this system is to provide support to micro web-based businesses which do not themselves have the capability to develop proprietary solutions for e-business.

The service 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 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. Registration and searching for services can be handled by UDDI.

The system naturally decomposes into three parts: the portal, service providers and customers (Figure 2). The upper part of Figure 2 describes that part of the functionality which involves the portal. The lower part concerns the peer-to-peer functionality.

The Portal. The portal enables remote data search and service navigation. Moreover it 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 supports a significant number of concurrent sessions while providing end-to-end security of the transactions.

The Service Provider. A new service provider joining the system first must register at the portal. A registered service provider can publish its services onto the portal dynamically. 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.

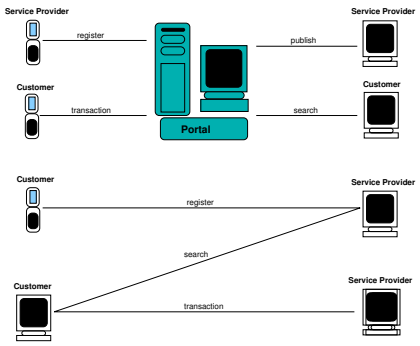


Fig. 2. Architecture of the web-based business system

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 with this service provider.

4 UML Model of the System

We turn now to our model of the above system. The performance model of the system consists of a collaboration between sequential object instances which undertake timed activities either individually, or in collaboration with other objects. Thus the UML diagram types which are used to describe this model are class diagrams (identifying the kinds of the objects in the system), state diagrams (detailing the behaviour of the objects) and collaboration diagrams (introducing an operational configuration of the system with named object instances collaborating on sets of activities).

Performance analysis of the system is conducted via the generation and solution of a continuous-time Markov chain (CTMC) representation of the system, thus the durations of all of the activities in the system are quantified by providing the parameter to a negative exponential distribution.

The state diagram which represents a buyer in the system is shown in Figure 3.

Other components in the model are not much more complex than that of the buyers. Figure 4 shows that the model of the service providers in the system have common synchronisation points with the buyers (reflecting exchanges which are not routed through the central portal in the system, for reasons of scalability). Where these synchronisation points occur, one of the interacting components specifies the rate of occurrence of the activity and the other passively co-operates with these activities.

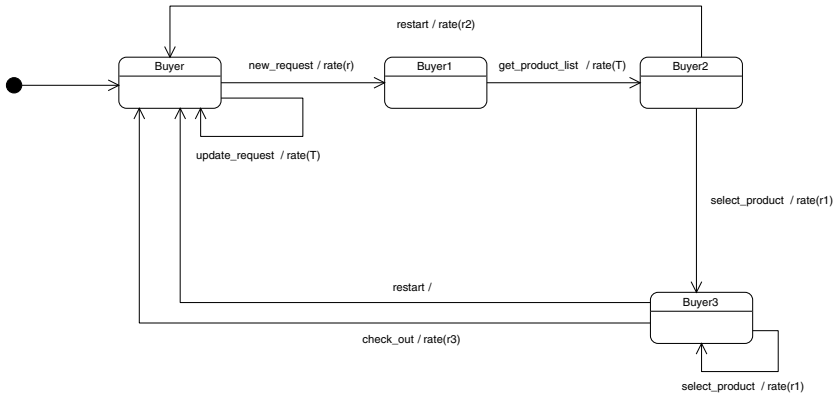


Fig. 3. State diagram of the Buyer in the Web-based micro-business model

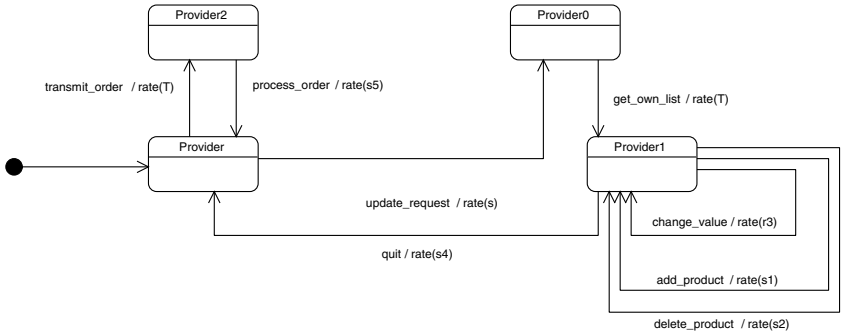


Fig. 4. State diagram of the Provider in the Web-based micro-business model

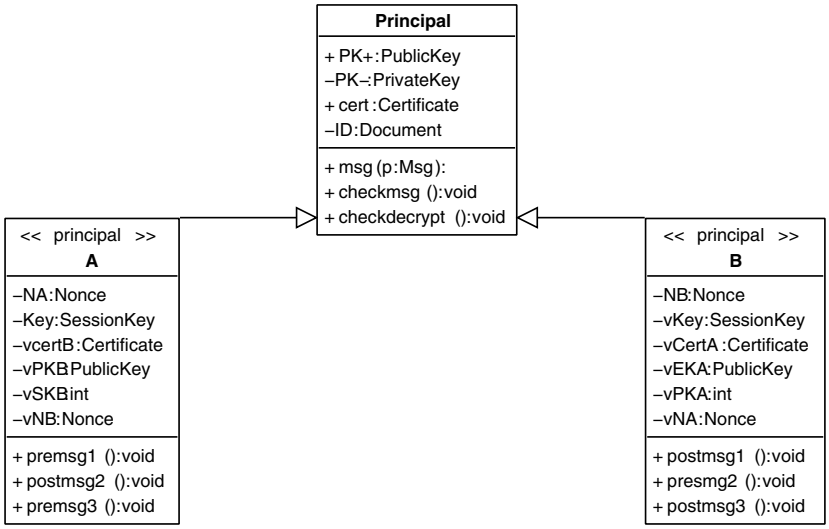


Fig. 5. The class diagram for the principals involved in secure transactions

In the UML design, security relevant information is specified by the ForLySa profile [4], which provides the means to annotate class diagrams and sequence diagrams with security-specific data. More precisely, ForLysa allows us to specify cryptographic security protocols with two participants (A and B) who typically exchange a new session key. Such protocols use cryptographic concepts like cryptographic keys and nonces, which are provided by two classes in the ForLysa profile: the class *Msg* for messages and the class *Principal* for participants of the protocol. The class *Msg* has attributes holding the sender and receiver of the message and the encrypted and unencrypted payloads of the message; the latter are objects of appropriate classes, and these classes contain methods for encrypting and decrypting data. The class *Principal* contains attributes for the private/public keys or symmetric keys associated with a principal, and specifies methods for sending and checking of messages.

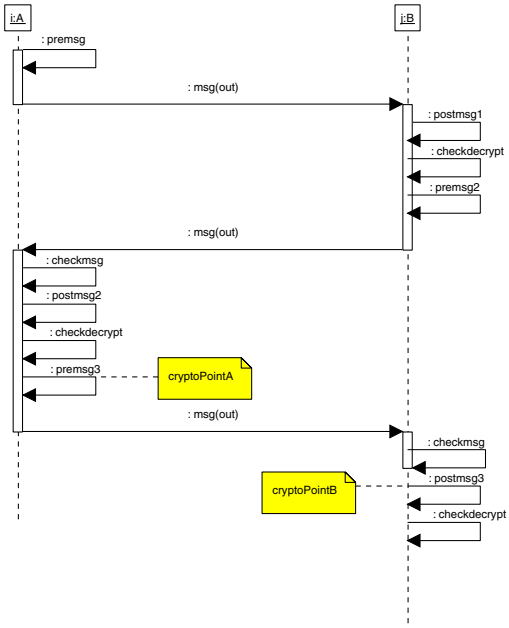


Fig. 6. The sequence diagram of the protocol for the principals involved in secure transactions

As an example, we show the UML design in Choreographer of the cryptographic security protocol described in Section 6, consisting of a class diagram and a sequence diagram. The class diagram, shown in Figure 5, specifies two principals A and B, as subclasses of *Principal*, which have attributes to hold the data generated or acquired during a run of the protocol.

The sequence diagram in Figure 6 describes the exchange of messages between A and B which defines the protocol. For each message, first the sender prepares and encrypts the content in method *premsg* by providing values for the attributes of a variable *out* of class *Msg*. When receiving a message, the recipient checks its contents (eg. correct addresses) with method *checkmsg*, then decrypts the encrypted parts with method *postmsg*. This assigns a value to the attribute which holds the decrypted content of the message. The decrypted part is then analysed in *checkdecrypt* where the receiver checks that the content has the required format. Figure 5 shows the call sequence for these methods, while the body of each method is specified by constraints which are not visible in the diagram.

5 Performance Analysis

The performance analysis of the above UML project proceeds by extracting a performance model in Hillston's Performance Evaluation Process Algebra (PEPA) [3]. This extraction is performed automatically by the Choreographer design platform.

5.1 The PEPA Model

The objects whose behaviour is specified by state diagrams in the UML model give rise to PEPA components in the process algebra model. The first component, *Portal*, models the behaviour of the interface between the service providers and the customers. The second component, *Provider*, models any provider registered in the system. 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 known to the system: they have already registered.

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—for example, books. This can be done by a simple click 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 targeted 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 rhythm.

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

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 using 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 of the changes which he wants to 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 what is necessary to satisfy the request. This is modelled using action type *process_order*.

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

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.

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

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:

$$
\begin{aligned}
&Web\_Business \stackrel{def}{=} \\
&\quad (Buyer \underset{K}{\bowtie} \ldots \underset{K}{\bowtie} Buyer) \underset{L}{\bowtie} \big((Provider || \ldots || Provider) \underset{M}{\bowtie} Portal\big)
\end{aligned}
$$

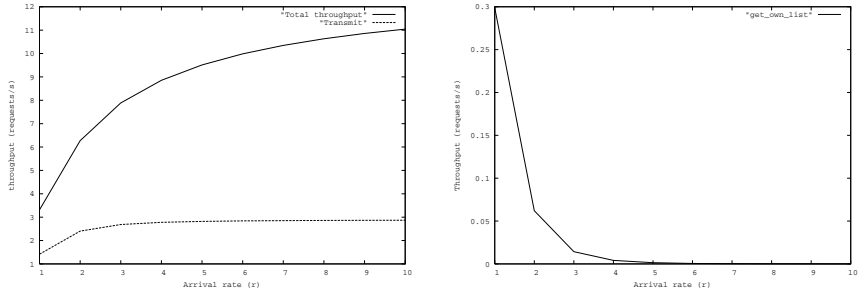
where the synchronising sets are defined as follows:

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

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.

5.2 Numerical Results

In this section we give an idea of the performance measures which 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 retain intellectual control 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.



(a) Total throughput (b) Throughput for provider requests

Fig. 7. Throughput computation

- Figure 7(a) depicts the total throughput of the portal in terms of buyer's requests to get a product list and to select a product from a list, and the provider's requests to get its own list. This figure also gives the throughput part related to the transmission of the orders to the provider. As we can see, the transmission of the buyer's orders is a very small part of the throughput of the system. This may be explained by the fact that the buyers spend the greater part of their time selecting products. Moreover, once an item is selected, a buyer may decide to abandon or restart. Thus not all buyers end up checking out with purchases.
- Figure 7(b) 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 7(a), 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.

6 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 and for authentication.

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 [5], and the main task consists of designing an appropriate protocol for *authenticated key exchange*. Such a protocol allows two or more participants to exchange a cryptographic session key in such 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 cryptographic concepts. Before describing the LySatool [2] 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. The protocol can be realised by the use of WS-Security, which provides all of the necessary mechanisms.

6.1 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 $cert_U$, e.g. X.509 certificates, which are signed by some trusted certification authority.

(1) A $\rightarrow$ B: A, $cert_A$
(2) B $\rightarrow$ A: $\{\text{B, NB}\}:\text{K}_A^+$, $cert_B$
(3) A $\rightarrow$ B: $\{\text{A, NB, K}_{AB}\}:\text{K}_B^+$

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.

6.2 LySa Model of the Protocol

The informal notation of the protocol used above leaves implicit a number of assumptions and does not completely describe actions such as 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 which potential attackers can perform.

For a formal analysis, these assumptions have to be specified. LySa provides a format for this, which is essentially a process algebra, enriched by cryptographic notions such as encryption and decryption, symmetric keys, public and private keys, allowing it to model authenticated key exchange protocols. More precisely, LySa is based on the π-calculus. The main difference from 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 (such as nonces), and to bind values to free variables. Each participant in the protocol (in our case A and B) is 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 an example, consider message (3), sent from A to B. 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 (A), the second the recipient (B), and the rest is the content of the message. The content in this case consists of only one, encrypted, part. The terms are either names such as A, B, and K_{AB}, or variables such as vNB which has been bound to the value of NB when A received message (2). Sending message (3) is preceded by generating a new session key K_{AB} which 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. Pattern matching is again 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^-.

As described, the protocol consists of two classes of processes: the process for A and the process for B. In the LySa model every participant 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 attacker built into the LySa model has the usual powers of the standard Dolev-Yao attacker [6], i.e. they can use all of the information obtained from messages sent between participants to compose messages which can be sent to any participant.

6.3 Security Analysis with LySa

The analysis performed by the LySatool 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. The underlying technique is static analysis, more specifically the Succinct Solver Suite [7] provides the implementation of the solution procedures which are deployed to effect the analysis. 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 $a3$ dest $b3$] while receiving of message (3) has annotation [at $b3$ orig $a3$].

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 which 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, $\mathrm{K}_{AB}\}$: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 misuses 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) I_B $\rightarrow$ A: $\{$B, NB' $\}$:K_A^+	
(3) A $\rightarrow$ 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.

7 Related Work

With regard to the performance analysis of UML models there are a range of significant prior works which have similarities with the performance-related part of our work. In many cases, these map UML diagrams of various kinds to other analysable representations including stochastic Petri nets [8, 9], layered queueing networks [10], generalised

semi-Markov processes [11] and others. Some works are particularly noteworthy for their careful consideration of the role of the UML metamodel in the performance analysis process [12]. Our work has some similarities with the above, and many differences (different diagram types, different performance analysis technology). Two things are unique to our work here: an integrated technology for security analysis and the use of *reflectors* to reflect the results of the analysis back to the UML level.

Other methodologies based on UML have been defined in order to specify security aspects of designs. UMLsec by Jan Jürjens [13, 14] is a versatile profile that includes a wide range of high-level security concepts like secrecy, integrity, no-down-flow, fair exchange etc. and allows the user to specify hardware platforms such as LAN, smart card, Internet and others. It is however not possible to specify correct authentication, which is the main security requirement on the key exchange protocols which are part of the case studies that we have considered. As in the UML content processed by the LySa extractor, UMLsec protocols are specified by sequence diagrams, and the constraints used in the sequence diagrams are similar. However, the UML use supported by the LySa extractor provides a means to specify cryptopoints in sequence diagrams, which is an essential prerequisite for analysing correct authentication with LySa.

8 Conclusions

We have presented a novel method for analysing security and performance questions about UML-described systems which follow a modern, open design pattern. The classes of behaviours understood within the system are described by class and state diagrams. The interactions between object instances of these classes are described using collaboration diagrams and sequence diagrams. The Choreographer design platform automatically processes descriptions of systems structured in this way, and packaged as a UML project. Process algebra representations of the formal content of the diagrams are extracted and passed to efficient analysers which check performance and security properties. The results of these analysers can be inspected directly or reflected back through the Choreographer design platform in order to present all of the analysis at the UML level.

Through the use of the UML as an interface to the security and performance analysis process we hope that we have an accessible framework which could attract developers facing difficulties in engineering secure systems with high performance to consider formal analysis as a beneficial complement to their current design practices. There are many benefits to the use of formal modelling and analysis methods, not the least of which is the ability to display that due care and attention has been taken in the development of secure services which are to be used in business-to-business contexts.

Acknowledgements. The work described in the present paper was undertaken while the authors were supported by the DEGAS (Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing. The Choreographer design platform is a Java application which has been successfully tested on Windows and Red Hat Linux systems. It is available for download from `http://www.lfcs.ed.ac.uk/choreographer`.

References

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

[2] 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.

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

[4] 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, Rovereto, Italy, 2004. Springer Verlag.

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

[6] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 22(6):198–208, 1983.

[7] 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 of *LNCS*, pages 251–265. Springer-Verlag, 2004.

[8] 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.

[9] Juan Pablo López-Grao, José Merseguer, and Javier Campos. From UML activity diagrams to Stochastic Petri nets: application to software performance engineering. In *Proceedings of the fourth international Workshop on Software and Performance*, pages 25–36. ACM Press, 2004.

[10] D.C. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In *Proceedings of Tools'02*, number 2324 in LNCS, pages 159–177. Springer-Verlag, April 2002.

[11] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst. Performance analysis of time-enhanced UML diagrams based on stochastic processes. In Tucci [15], pages 25–34.

[12] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In Tucci [15], pages 35–45.

[13] Jan Jürjens. UMLsec: Extending UML for secure systems development. In *5th Intl. Conference on the Unified Modeling Language (UML) 2000*, LNCS 2460, 2002.

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

[15] Salvatore Tucci, editor. *Proceedings of the Third International Workshop on Software and Performance (WOSP 2002)*. ACM Press, Rome, Italy, July 2002.