Technical University of Vienna
Information Systems Institute
Distributed Systems Group

Towards a generalized payment model for Internet services

Michael Fischer and
Manfred Hauswirth
M.Fischer@infosys.tuwien.ac.at
M.Hauswirth@infosys.tuwien.ac.at

TUV-1841-01-03 August 15, 2001

Although at the moment electronic payment via the Internet has no major impact on e-commerce sites, it will be a prerequisite for new business models, such as pay-per-view, and necessary for the success and cost efficiency of future e-commerce applications. Many payment protocols and frameworks for various business domains exist. However, they are incompatible which makes it hard for service designers to "design for change:" Implementations are tightly-coupled with the applied payment mechanisms and cannot be adapted/changed easily. To improve this situation we must factor out expressive, common payment abstractions and come up with a generalized payment model which hides the payment mechanisms used, but offers a common, high-level interface for applications and supports a wide range of business models. On the basis of analyzing the standard payment scenarios and configurations this paper presents a component-oriented payment model, describes the security concerns of various configurations and their solutions, and evaluates the model with a concrete application in our OPELIX project. The contribution of this paper is in the decomposition approach for payment models and a generalized payment model which supports component-orientation, design for change, and assessment of security.

Keywords: Payment models, e-commerce

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

TOWARDS A GENERALIZED PAYMENT MODEL FOR INTERNET SERVICES*

Michael Fischer, Manfred Hauswirth†
Distributed Systems Group, Technical University of Vienna, Austria
[M.Fischer, M.Hauswirth]@infosys.tuwien.ac.at

Abstract Although at the moment electronic payment via the Internet has no major impact on e-commerce sites, it will be a prerequisite for new business models, such as pay-per-view, and necessary for the success and cost efficiency of future e-commerce applications. Many payment protocols and frameworks for various business domains exist. However, they are incompatible which makes it hard for service designers to "design for change:" Implementations are tightly-coupled with the applied payment mechanisms and cannot be adapted/changed easily. To improve this situation we must factor out expressive, common payment abstractions and come up with a generalized payment model which hides the payment mechanisms used, but offers a common, high-level interface for applications and supports a wide range of business models. On the basis of analyzing the standard payment scenarios and configurations this paper presents a component-oriented payment model, describes the security concerns of various configurations and their solutions, and evaluates the model with a concrete application in our OPELIX project. The contribution of this paper is in the decomposition approach for payment models and a generalized payment model which supports component-orientation, design for change, and assessment of security.

Keywords: Payment models, e-commerce

1. Introduction

Besides free Internet services and services funded by advertisement, e-commerce sites which require some form of payment for (part of) their services are the main segment of growth in the Internet landscape. At the moment the standard payment instrument are credit cards: When the

*This work was supported in part by the European Commission under contract IST-1999-10288, project OPELIX (Open Personalized Electronic Information Commerce System).
†contact author

customer decides to buy, he/she fills in a form with his/her credit card information which is then transmitted to the vendor's site via SSL [2]. Only very few sites offer electronic payment via a payment protocol such as SET [12]. Even fewer sites employ micro-payment protocols such as Millicent [5]. However, electronic payment is important for e-commerce applications because it enables highly relevant new business models. For example, pay-per-view business models are only possible with electronic payment instruments.

Assuming that electronic payment will be employed on a large scale in the near future and on the basis of the state-of-the-art in payment systems, a key problem surfaces: There will be a number of competing standards which are incompatible and require applications to be tightly coupled with them. This impedes component-oriented design for change and makes it hard for customers to interact with vendors who use different payment protocols. Additionally, vendors (or customers) may want to delegate payment handling to specialized third-parties who offer appropriate security and sufficient proofs of their transactions. Such payment intermediaries would have to support a wide range of different payment protocols and business models at reasonable costs to make a revenue out of their business. This is the setting we are assuming in the development of the OPELIX e-commerce platform [8] as shown in Figure 1a.

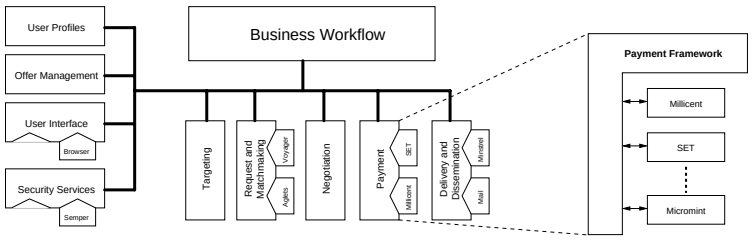

Figure 1a. OPELIX architecture

Figure 1b. Payment framework

OPELIX offers a number of specialized components which support all business phases [9] and are governed by a business workflow. The main design goals were component-orientation and interoperability. Component-orientation was targeted in two ways: We wanted to enable the use of components off-the-shelf (COTS), e.g., an existing SET implementation, and make OPELIX itself component-oriented to allow OPELIX users to configure the platform according to their requirements, i.e., only those components must be present which are actually needed. The support

of COTS is critical and led to the simple architectural pattern for their integration as shown in Figure 1b for the payment component.

Every OPELIX component which uses COTS provides an external interface to other OPELIX components and an internal interface to the COTS used. This allows us to abstract from the specific interfaces of COTS (such as concrete payment services) and offers a single interface for the other OPELIX components that they have to deal with. Additionally every component uses a common asynchronous event notification mechanism to which other components can subscribe. For example, one component can request a payment operation, continue with other tasks, and is notified upon completion of the operation. Other components which are also interested in this payment's completion can subscribe to the same event type and also receive data of interest, e.g., copies of the receipt. This mechanism facilitates highly dynamic interactions but still provides separation of concerns.

In this paper we focus on the payment component and present our component-oriented, generalized payment model which can be applied on top of existing payment mechanisms. It offers a set of uniform abstractions and provides a generalized, high-level payment interface for applications which supports "design for change." Section 2 describes our generalized model by presenting the Minstrel push system's (a subproject of the OPELIX project) payment model and introduces the abstractions and interactions of our approach. Section 3 then discusses typical payment configurations and positions our approach with respect to them. The security model of our approach is presented in Section 4. Section 5 then evaluates our approach and presents future work. This is followed by related work in Section 6. Finally Section 7 completes the paper with our conclusions.

2. Payment Model

Minstrel [6] is a push system for dissemination of content in OPELIX. It supports flexible business models such as pay-per-view, volume-based, time-based, pre-paid, or post-paid. Our generalized payment model stems from our work and experience with Minstrel's payment and business models [7]. We will use it as a case study in the the following because it illustrates all requirements of the generalized model and the problems which had to be solved. Minstrel is still exploited as a testbed for our implementation. Figure 2 shows the pay-per-view scenario we use in Minstrel as a UML use case diagram (this use case actually is the general use case for payment).

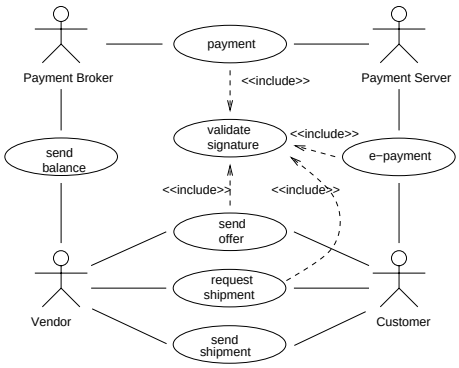

Figure 2. Payment use case diagram

The scenario involves the following actors: the *vendor* (*broadcaster* in Minstrel terminology) sends *offers* about information it wants to sell to *customers* (*receivers* in Minstrel terminology); on the basis of the received offers the customer can decide to request a *shipment* of the information; depending on the business model the customer may have to pay before or after the shipment is sent or may just have to present a proof of payment (*receipt*); to perform a payment the customer contacts a payment server (defined by the vendor or the customer's favorite one); it accepts certain kinds of electronic payment and issues signed receipts for successful payments; this server is operated by or cooperates with *payment brokers* (financial institutions that are responsible for the real-money transfer from the payment server's account to the vendor's account). We consider this not necessarily to be a single entity: It may be a bank, the customer's ISP who offers a charging service for third parties in a way as it is already done for paying via the telephone bill, or any other thinkable business entity/consortium. The identity of all parties can be guaranteed through a public key infrastructure.

The benefit for the vendors in this setting is that they do not have to take care of payment issues. However, it is not necessary to route all messages through a payment server since digital signatures ensure the proper operation of the protocol and can serve as evidences in case of arguments. The existing infrastructure of ISPs, banks, or credit card companies can be used to offer payment services to customers supporting their favorite payment scheme while guaranteeing payment to the vendor account through the certified identity of the payment server. Thus also the customer is freed from having the "correct" payment instrument

installed: One payment instrument is sufficient to pay at the payment servers which can mediate between all kinds of payment instruments.

2.1. 3+1 Party Generic Payment Protocol

This section takes a closer look at the sequence of actions during a payment interaction (for example, as performed in Minstrel). Figure 3 shows the sequence diagram for a payment in our model.

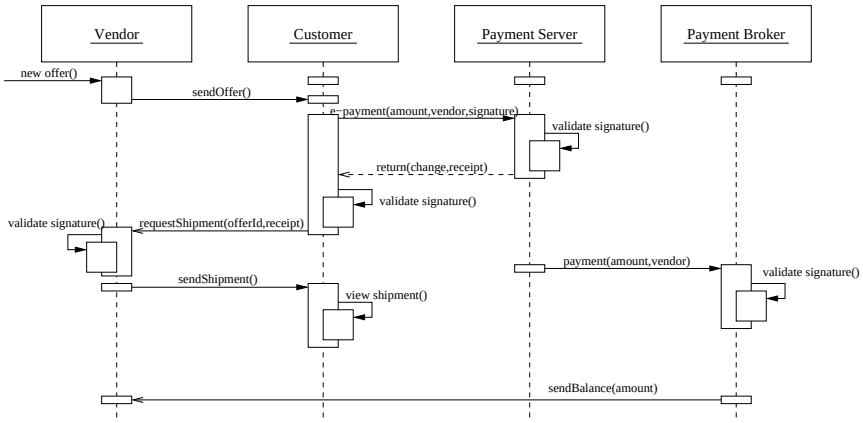

Figure 3. Payment sequence diagram

Initially the vendor sends an offer with price and payment information to the customer. This is not part of the 3+1 party generic payment protocol (3+1PGPP). If the customer accepts the payment terms and wants to buy, the customer authorizes the payment and sends a payment request using its default payment instrument to the payment server. Besides the information required for the payment instrument the request includes vendor information and a digital signature of the customer. The payment server validates the request to check the authenticity, performs the payment according to the payment instrument, and generates and returns a receipt to the customer upon successful completion. The receipt is generated by signing the customer-signature together with the amount and the vendor information (the security aspects will be discussed in detail in Section 4). In some business models, for example, in a time-based business model, this payment step may be skipped if the customer already has a valid receipt which may be presented to the vendor as proof of payment as described below.

The customer validates the receipt to proof the authenticity of the payment server and can now be sure that the server has accepted its

payment and the amount was/will be transferred to the vendor's account. This transfer can be done via an arbitrary payment protocol and employs similar authentication checks using digital signatures. Upon successful validation the customer sends a shipment request to the vendor which contains the offer_id, customer_id and receipt information (amount, id of payment server, serial number, expiration date, payment server signature) from the payment server's response. The vendor validates the shipment request by checking the payment server's signature which guarantees that the required amount was/will be paid to the vendor's account. Upon successful validation the vendor sends the shipment to the customer (this is not part of 3+1PGPP).

The payment server transfers the paid amount to the payment broker using an arbitrary protocol either immediately or when a certain threshold amount has been accumulated, or in regular intervals, or according to some other defined procedure they agreed upon.

This scenario can be mapped directly to a pay-per-view business model but can also be generalized easily to arbitrary other ones depending on the semantics assigned to the receipt and the additional data it carries. For example, in a time-based business model the payment would happen once in a while and the receipt would hold a timestamp until which the payment is valid; in a volume-based model it would hold a counter, etc.

3. Payment Configurations

The analysis of current payment protocols shows that most of the protocols are broker-centric, i.e., the broker is involved in the protocol during the payment operation. For example, NetBill [13] requires a server for billing and dispute handling, DigiCash [1] needs one to detect double-spending, and SET [12] must validate credit card data at a server. One exception is the Millicent [5] protocol in which the integrity of the electronic "currency" (scrip in Millicent terminology), can be validated by the vendor. Another, sometimes disadvantageous property of some protocols is that they define the payment and delivery process as an integrated transaction (e.g., NetBill). A limiting factor in many micro-payment protocols is the requirement to subscribe at the vendor's payment broker to obtain the virtual currency. Guarantee for payment is achieved by mapping real money onto cryptographic properties which is usually done by the broker. This guarantee could be given by any certified authority not only the vendor's payment broker.

Besides these problems the current payment approaches are useful in many settings but still do not fit into the modular design of the Minstrel

push system and our requirements for flexible business model support in OPELIX: (1) Minstrel's optimized distribution infrastructure decouples payment and delivery, i.e., an arbitrary number of delivery components may use a single payment component; (2) the business model and the associated payment model are not known a priori; (3) there is no interoperability between different payment schemes; and (4) the customer/vendor has little freedom in choosing the payment instrument/provider. These constraints motivated the need of a generalized payment framework as also suggested by [11].

The first natural steps towards a general model usually is the application of the architectural proxy pattern as done in the Universal Payment Application Interface (U-PAI) approach [10]. It can be used to implement a proxy/gateway for accessing services via different payment systems. Payment, evidence generation, accounting, etc. are handled by the proxy/gateway service. Although this approach facilitates the interaction with many different payment services it still limits component distribution aspects by requiring a "centralized" proxy/gateway.

To meet our goal of flexible business model support for a highly modular system we had to separate the delivery of goods from the payment and define the communication mechanism between these two components (which can be located at different network nodes). Our approach is to use signed receipts as a proof of payment in the communication among the business partners. Additionally the receipt serves as the central concept for implementing business models depending on the additional data it holds and the semantics assigned to this data. This configuration is depicted in Figure 4.

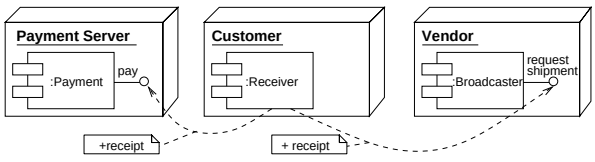

Figure 4. Generalized approach

The customer has an account at some payment server(s) (e.g., the ISP) which it uses to commit payments. In our model we use the receipt which is returned to the customer by the payment server as the only evidence for payment. Receipts are digitally signed to ensure their integrity. Shared keys for symmetric cryptography or public/private key pairs for asymmetric cryptography between the parties ensure the receipt's security properties. The interactions in this configuration were

already described in Section 2.1. Its security implications are discussed in further detail in the next section. The advantages of this configuration are the benefits of the proxy approach (interaction with arbitrary payment systems), combined with a modular design that decouples payment from any other functionality (delivery of shipments in our case) and support for arbitrary business models via the concept of extensible receipts.

4. Security Model

This section describes the concepts to ensure the receipt's integrity and authenticity among the participants. Our approach is based on the technique described in [5] and uses shared secrets (e.g., a string of random characters), which must be negotiated once via a secure channel (e.g., SSL). The secrets are associated with unique ids (*vendor_id*, *customer_id*, etc.) so that they can be inferred from the id of a communication partner. Message security is achieved by sending the message plus its signature which is calculated by applying a secure hash function (e.g., SHA or MD5) to the message and the associated secret ($pdu = message + hash(message + secret)$). The addressee can verify the signature by recalculating the signature using the shared secret. The security stems from the fact that it is infeasible for an outside observer to generate the signature without knowledge of the secret or deduce the secret by knowing the message and the signature.

In the subsequent discussions of the payment process we assume the following setting: (1) The customer is already registered at the vendor's push system (this implies that the *vendor_customer_secret* for message authentication has been exchanged, the customer has a valid *customer_id*, and knows the *vendor_id*); (2) the customer is registered at a payment server, i.e., the customer and the payment server have agreed on a *paysrv_customer_secret* and the account's balance is sufficient for payment; (3) the customer has received an offer from the vendor containing an *offer_id* and a *price*; and (4) the vendor and the payment server have already authenticated themselves and agreed on the *paysrv_vendor_secret*.

To pay for an offer the customer first creates a unique fingerprint for this payment transaction by performing the following operation which yields the Vendor–Customer signature calculated by the customer (additional information describing the offer, e.g., a URL, may be included in the signature):

$$VC\_CustomerSign \quad = \quad hash(offer\_id + vendor\_customer\_secret)$$

Then the customer constructs a signed payment request (some payment protocols do this implicitly, e.g., Millicent)

$$PC\_CustomerSign = hash(vendor\_id + serial + price + VC\_CustomerSign + paysrv\_customer\_secret)$$

and sends the request, i.e., *vendor_id*, *serial* which is required for replay detection, *price*, *VC_CustomerSign* and *PC_CustomerSign*, along with the payment protocol information to the payment server. On successful completion of the payment transaction the payment server returns a receipt holding the following signatures to the customer:

$$PV\_PaysrvSign = hash(payserv\_id + serial + price + VC\_CustomerSign + paysrv\_vendor\_secret)$$

$$PC\_PaysrvSign = hash(PV\_PaysrvSign + paysrv\_customer\_secret)$$

The Payment Server–Vendor signature ($PV\_PaysrvSign$) will be sent to the Vendor by the Customer in the shipment request (see below).

Now the customer validates the Payment Server–Customer signature ($PC\_PaysrvSign$) by recalculating the signature using the shared secret. If the validation of $PC\_PaysrvSign$ succeeds, the customer can assume that $PV\_PaysrvSign$ is correct. If not, but $PC\_PaysrvSign$ is correct and the vendor complains about a bad $PV\_PaysrvSign$ signature, someone must cheat or security has been compromised and a dispute handling mechanism must be involved.

Having paid the customer requests a shipment by sending *customer_id*, *offer_id*, some information from the receipt (*paysrv_id*, *PV_PaysrvSign*, *serial*, *price*), and *VC_CustomerSign* to the vendor. The vendor uses this information and the associations between secrets and signatures as shown in Figure 5 to validate the integrity and authenticity of the request.

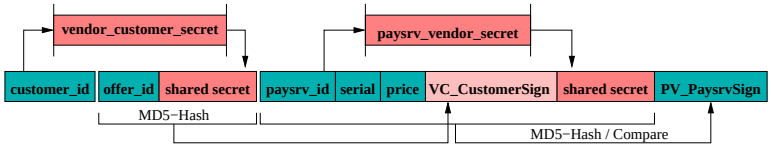

Figure 5. Payment validation at the vendor

The vendor first reconstructs the customer's signature via the *customer_id* by inferring the *vendor_customer_secret* which is then used to recalculate *VC_CustomerSign*. Then the vendor uses the *payserv_id* to infer *paysrv_vendor_secret* and uses it to recalculate *PV_PaysrvSign*. If the comparison fails then someone must cheat or security has been compromised and a dispute handling mechanism must be involved (same as above). Otherwise the payment (receipt) is valid. This means that the following condition holds:

$$PV\_PaysrvSign = hash(s_{pv} + hash(s_{vc} + k_{pc}) + k_{pv})$$

i.e., the Payment Server's signature of the receipt for the vendor to validate a customer request, where s_{pv} denotes the string of payment server–vendor properties, k_{vc} is the shared secret between payment server and vendor, s_{vc} denotes the string of vendor–customer properties (*offer_id*, etc.), and k_{pc} is the associated *paysrv_customer_secret*.

The above process ensures end–to–end security in terms of authenticity and non-repudiation between payment server and vendor by the use of shared secrets. The described method is sufficient for use within the Minstrel push system, since it is targeted on efficient delivery of low-value items by assuming a long term customer-vendor relationship where anonymity is not important. For higher security requirements it may be enhanced by using asymmetric cryptography for signatures to achieve privacy or to generate legal evidences.

5. Evaluation and Future Work

We have implemented a proof-of-concept version of the 3+1PGPP approach and have used it in Minstrel for the implementation of a pay-per-view business model together with our Millicent prototype implementation. As described above we have generalized and extended Millicent's [5] digital signature scheme to fit the requirements of 3+1PGPP. So far we have not found any limitations of our modeling approach. The key concept in our model is the notion of extensible receipts which we exploit to support arbitrary business models. Depending on the business model additional fields may be included in the receipt. At the moment we are working on a production-quality implementation of 3+1PGPP which will be released as part of the OPELIX platform.

Our current implementation is written in Java and uses HTTP as the transport protocol. At the moment we transport Java objects via HTTP and use Java's subclassing mechanisms for receipt extensions. To improve interoperability we will switch to XML documents which are exchanged via HTTP in future releases. The performance of the current implementation was evaluated in a lab setting (2 vendors, 10 customers, 1 payment server). Although this does not provide empirical figures to judge scalability we can argue that due to the fact that we do not need a central server and all calculations are done locally by the involved parties very good scalability can be achieved (as shown by other systems which have no central server as a bottleneck). However, this will have to be confirmed empirically by the implementation of a larger-scale case study in OPELIX. We also plan further evaluations in several respects: For example, we are interested in questions such as finding out how many payment servers are necessary to minimize transaction costs while still

providing convenient service, how unused receipts could be returned, or how dispute handling could be included. Also integration of further payment systems into the 3+1PGPP framework will be subject of future work.

6. Related Work

So far little attention has been paid to the issues of generalized payment frameworks and payment interoperability. The ebXML [3, 4] specifications provide a framework for creating consistent, robust, and interoperable eBusiness services and components within an integrated global eBusiness market. Unfortunately it does not specify any kind of evidence generation for payment services in detail.

SEMPER [11] supports different payment instruments by its *generic payment service framework (GPSF)* which provides a well-defined interface for higher-level services but still provides enough flexibility for supporting different payment models, e.g., check-like or cash-like. It could serve as an implementation platform for our approach. NetBill [13] is a transaction-oriented micro-payment system with a centralized server. The complete protocol has 3 phases (8 steps): offer and price evaluation; shipment request; and payment. The payment phase starts when the customer sends a *electronic payment order (EPO)* to the vendor. The EPO is a signed message containing price and item information. The vendor creates a signed invoice (price, item information, decryption key) and sends this information along with the EPO to the NetBill server. After successful completion of the payment transaction at the NetBill server, the customer receives the decryption key from the vendor (or the NetBill server in case of dispute). The disadvantage in the NetBill model is the centralized server because it maintains all customer and vendor accounts which limits scalability and is a high security risk.

7. Conclusions

In this paper we have presented a systematic approach towards a generalized payment model for Internet-based services. We have described the use cases, the interactions which have to take place independent of the payment instrument used, and the artifacts and roles involved. Our approach decouples payment from applications in a way which enables the exchange of payment components and supports component-orientation also for payment. The 3+1PGPP approach supports interaction with arbitrary payment systems and facilitates the implementation of arbitrary business models via the concept of extensible receipts. Its component-oriented approach naturally supports distribution of the

components in a network which offers new business opportunities for specialized payment services and frees both vendors and customers from many problems in the domain of e-payment. Since our approach introduces new security concerns we have analyzed the possible security problems and presented feasible solutions. 3+1PGPP was validated with a pay-per-view business model case study in the Minstrel push system.

References

[1] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, **24**(2):84–8, 1981.

[2] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Network Working Group, January 1999. RFC 2246. http://www.ietf.org/rfc/rfc2246.txt.

[3] A. Dutton and D. Webber. Understanding ebXML, UDDI and XML/edi. XML.org, October 2000. http://www.xml.org/feature_articles/2000_1107_miller.shtml.

[4] Enabling Electronic Business with ebXML. White Paper. UN/CEFACT, OASIS, December 2000. http://www.ebxml.org/white_papers/whitepaper.htm.

[5] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. *Fourth International World Wide Web Conference* (Boston, Massachusetts, USA). Published as *World Wide Web Journal*, **1**(1). O'Reilly & Associates, Incorporated, November 1995. http://www.w3.org/Conferences/WWW4/Papers/246/.

[6] M. Hauswirth. *Internet-Scale Push Systems for Information Distribution—Architecture, Components, and Communication.* PhD thesis. Distributed Systems Group, Technical University of Vienna, October 1999.

[7] M. Hauswirth and M. Jazayeri. A Component and Communication Model for Push Systems. *Proceedings of the ESEC/FSE 99 – Joint 7th European Software Engineering Conference (ESEC) and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-7)* (Toulouse, France, September 6–10, 1999), pages 20–38, September 1999. http://www.infosys.tuwien.ac.at/Staff/pooh/papers/PushIssues/.

[8] M. Hauswirth, M. Jazayeri, Z. Miklos, I. Podnar, Elisabetta Di Nitto, and Andreas Wombacher. An Architecture for Information Commerce Systems. *Proceedings of ConTEL 2001 – 6th International Conference on Telecommunications* (Zagreb, Croatia, June 13-15, 2001), June 2001. http://www.infosys.tuwien.ac.at/Staff/pooh/papers/EC-Architecture/ConTEL.pdf.

[9] M. Hauswirth, M. Jazayeri, and M. Schneider. A Phase Model for E-Commerce Business Models and its Application to Security Assessment. *Proceedings of the 34th Hawaii International Conference on System Sciences* (Maui, Hawaii, January 3-6, 2001), January 2001.

[10] S. Ketchpel, H. Garcia-Molina, A. Paepcke, S. Hassan, and S. Cousins. UPAI: A Universal Payment Application Interface. *USENIX 2nd e-commerce workshop*, 1996. http://citeseer.nj.nec.com/ketchpel96upai.html.

[11] G. Lacoste, B. Pfitzmann, M. Steiner, and M. Waidner, editors. *SEMPER – Secure Electronic Marketplace for Europe*, volume 1854 of LNCS. Springer Verlag, Berlin, 2000.

[12] SET Secure Electronic Transaction LLC. *SET Secure Electronic Transaction Specification – Book 3: Formal Protocol Definition*, May 1997. Version 1.0. http://www.setco.org/download/set_bk3.pdf.

[13] M. Sirbu, B. Cox, and J. D. Tygar. NetBill Security and Transaction Protocol. *First USENIX Workshop on Electronic Commerce*, pages 77–88, October 1999. http://www.ini.cmu.edu/NETBILL/pubs/Usenix.html.