



Technical University of Vienna
Information Systems Institute
Distributed Systems Group

OPELIX: a model and system for information commerce

Manfred Hauswirth, Mehdi Jazayeri,
Ivana Podnar, Elisabetta di Nitto and
Andreas Wombacher
M.Hauswirth@infosys.tuwien.ac.at
M.Jazayeri@infosys.tuwien.ac.at
I.Podnar@infosys.tuwien.ac.at
dinitto@elet.polimi.it
wombach@ipsi.fhg.de

TUV-1841-01-10

September 28, 2001

Information commerce (i-commerce) is a special variant of e-commerce in which the goods being offered for sale are digital and thus amenable to processing by computers. Trading of digital goods opens up new business opportunities and also poses interesting software engineering challenges. We present a general model of information commerce that extends the traditional customer-vendor model with an arbitrary number of layers of intermediaries that process other vendor products and services and offer new products and services of their own. Examining the interactions among the participants in this model leads to the requirements for an infrastructure to support information commerce applications and systems. We present a general, flexible, component-oriented architecture for information commerce systems. The architecture supports all business phases of i-commerce and may be deployed and distributed in a heterogeneous environment according to a user-defined configuration. We present the design rationale for the OPELIX system and our implementation experience.

Keywords: Information commerce, e-commerce, software architecture, distributed systems

OPELIX: a model and system for information commerce*

M. Hauswirth,
M. Jazayeri, I. Podnar
Distributed Systems Group
Technical University of Vienna
Vienna, Austria
[mh,mj,ip]@infosys.tuwien.ac.at

E. Di Nitto
Dipartimento di
Elettronica e Informazione
Politecnico di Milano
Milano, Italy
dinitto@elet.polimi.it

A. Wombacher
Integrated Publication and
Information Systems Institute
Fraunhofer-Gesellschaft
Darmstadt, Germany
wombach@ipsi.fhg.de

ABSTRACT

Information commerce (i-commerce) is a special variant of e-commerce in which the goods being offered for sale are digital and thus amenable to processing by computers. Trading of digital goods opens up new business opportunities and also poses interesting software engineering challenges. We present a general model of information commerce that extends the traditional customer-vendor model with an arbitrary number of layers of intermediaries that process other vendor products and services and offer new products and services of their own. Examining the interactions among the participants in this model leads to the requirements for an infrastructure to support information commerce applications and systems. We present a general, flexible, component-oriented architecture for information commerce systems. The architecture supports all business phases of i-commerce and may be deployed and distributed in a heterogeneous environment according to a user-defined configuration. We present the design rationale for the OPELIX system and our implementation experience.

1. INTRODUCTION

The use of the Internet as a platform has had a profound influence on electronic commerce and has created new business opportunities and business models which require supporting infrastructures and models. The simple customer-vendor model of the early days of e-commerce has been augmented by a large number of intermediaries which increases the complexity due to the higher number of roles and interactions. An e-commerce infrastructure must support actors in finding and interacting with each other and also during the business process. In most cases the actors belong to different organizations, behave according to diverse business models and are assisted by their various information system infrastructures.

*This work was supported in part by the European Commission under contract IST-1999-10288 (OPELIX).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE 2002 May 19–25, 2002, Buenos Aires, Argentina
Copyright 2002 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

Satisfying these requirements poses significant challenges to software engineering technology. One of the primary challenges is to meet ease of evolution. This traditional software engineering requirement is particularly relevant in the highly dynamic Internet environment when businesses need to adjust to rapidly changing business conditions. This situation requires that the software should be rapidly evolvable and exist in many configurations and versions. Another important implication is that the software must be clearly tied to the business model so that changes in the business model can be easily traced to, and help to evolve, the software.

The OPELIX (Open Personalized Electronic Information Commerce System) project [23] presented in this paper targets these requirements and enables enterprises to produce, sell, deliver, and manage information products over the Internet. An example of an information product is digitized music, such as a particular song or symphony. Such products do not incur duplication or distribution costs to the provider, maybe subject to copyright and may be combined with other similar products (e.g., all songs by a particular singer). The paper starts with the domain model for information commerce (i-commerce) in Section 2 which describes an information marketplace in which actors can trade information products (intangible products [20]). Section 3 then discusses the driving factors and requirements for the design of the OPELIX infrastructure which together with the domain model lead to the architecture of the OPELIX system presented in Section 4. We compare OPELIX with related work in Section 5 and present our conclusions in Section 6.

2. DOMAIN MODEL FOR I-COMMERCE

The OPELIX project focuses on *information marketplaces*, virtual, distributed locations in which *providers* offer *information* and *services* as goods and customers can find, evaluate and buy these goods. The traded products are intangible goods as defined in [20] which offers many new business opportunities and interaction patterns but also requires special treatment in several respects such as security and copyright protection of the products [20].

The domain model identifies business actors involved in marketplace business processes, describes their interactions, and the artifacts that are used and produced during these interactions. The goals for our domain modeling procedure were to understand information marketplaces and to identify potential intermediary services in this environment. We have employed the use case driven modeling approach defined in [17] and UML as the modeling language. This section presents an overview of our information marketplace

domain model and its business process model. Detailed descriptions of the models are given in [19] and [12].

Figure 1 relates a customer, a provider, and an intermediary performing a generic service in a use case diagram that describes the information marketplace. A *provider* generates and offers products or services to customers and intermediaries, delivers them according to the negotiated business terms, and may require payment for them. An *intermediary* offers products or services to customers, providers, and intermediaries. A concrete business model can involve any number of any of these roles but at least must consist of a customer and a provider.

In our model an intermediary acts as a connection between customers and providers. A customer uses the services of an intermediary to simplify interactions with providers. We model this interaction by the use case Perform service. For example, an intermediary can help a customer find a dependable provider by advertising products on the provider's behalf.

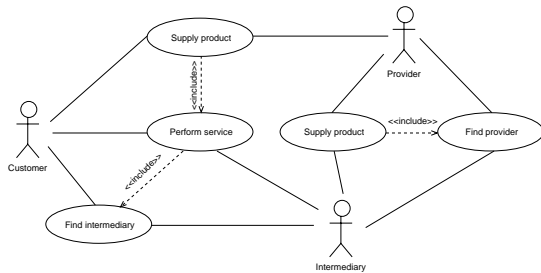


Figure 1: Use case diagram relating the actors of the information marketplace

After locating the provider the customer can choose to conduct business directly with the provider (Supply product use case that connects customer and provider). A customer may also choose to communicate only with an intermediary. In this case the intermediary delivers products, handles payment, etc. on behalf of the provider (Perform service use case). The two remaining use cases Find provider and Find intermediary model the initial process of locating providers and intermediaries.

The most interesting and complex use case is Perform service (where service is meant as a generic service). The services and products an intermediary offers can be manifold. It can provide search and retrieval services, advertise products or services, group, or aggregate information products, or provide negotiation or payment services. The underlying idea is that customers, providers, or intermediaries can delegate certain functionalities to specialized intermediaries so that they do not have to address certain issues themselves.

During their interactions customers, providers, and intermediaries produce, use, exchange and modify the following main artifacts:

Request: defines what a service or product of interest to a party is; it is issued by a customer or intermediary and sent to a provider or intermediary.

Offer: defines a service or product of a provider or intermediary (including legal terms and prices); issued by a provider or intermediary towards a customer or intermediary.

Order: if a party is satisfied with an offer (possibly after a negotiation phase) an order is placed with the offering party;

it is issued by a customer or intermediary towards a provider or intermediary.

Product: a good (service, information) which is traded in a business model; it is sent by a provider or intermediary to a customer or intermediary.

The sequence of interactions among customers, providers, and intermediaries define the phases of the business process. A typical business model consists of a combination (of a subset) of the following phases:

Matching: Parties publish descriptions of the available products to enable other parties to discover products of their interest and browse through available offers. Offers may be legally binding or not. The matching phase exists in 2 flavors: targeting (the provider initiates the interaction by sending offers to potential users) and searching (the customer initiates the interaction by issuing a request). Typical implementations include publishing on web servers (passive), mail/push distribution (active), or active searching and matching (robots, mobile agents). This phase was renamed from advertising as presented in [12] to matchmaking since it covers a broader notion than implied by the term advertising.

Negotiation: Once a product of interest is found, negotiating the business terms and possibly the properties of the product can start. Independently of the concrete negotiation process this phase must end with an agreement between the involved parties to continue with the succeeding phases. If no agreement can be reached at all the business process aborts. However, negotiation and matching can trigger each other mutually: If a party declines an offer it can request new offers or the party issuing the original offer can send new offers.

Ordering: After an agreement on the product and the business terms has been reached, a party may order the product. If the agreement is legally binding, we call it a *contract*.

Payment: If a product requires payment, then monetary values must be exchanged. We consider payment from a high-level point of view due to the arbitrary ways it can be done: It may involve credit card interactions, a bonus point system, micro-payments, or electronic money transfers, and heavily depends on the applied payment model such as rates, pay-per-use, or flat fees. Since these models involve very different concerns we address the conceptual superset and assume that the applied payment system secures payment transactions.

Delivery: In this phase the involved product is delivered to the requesting parties. Security in this phase heavily depends on whether products are tangible or intangible. Security for tangible goods is provided by non-electronic means whereas for intangible goods additional security issues apply, which require special consideration. For example, intangible goods such as programs or documents may be duplicated and sold without the consent of the copyright holder. The security problems of intangible goods and an approach to address them are presented in [20].

The phases described above are the building blocks for OPELIX's incremental business process model in which the provider gradually delegates phases (i.e., functionality) to the intermediary. The benefit for the provider in these models is that it can delegate parts of the process and pays the intermediary for the service(s) it provides. The customer may also benefit because the models may allow him/her to compare prices and products, combine them, or simply order

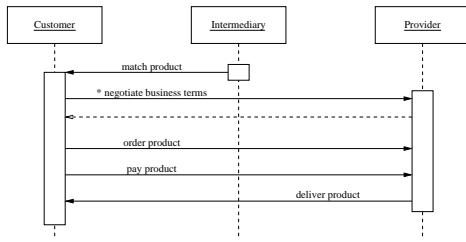


Figure 2: The A model

them at a single location.

The model was developed in the course of the OPELIX project to represent business models and analyze their properties of interest. For example in [12] this model is applied for security assessment of all representable 3-party business models. An incremental approach was chosen for the model to simplify the presentation without constraining its general applicability. In the following we briefly describe 3 possible business process model configurations to illustrate the model's underlying concepts. A complete description of OPELIX's business process model is given in [12].

In the simplest case all interactions occur directly between the customer and the provider. We call this the *direct model*. This model is used frequently today but is likely to diminish in importance because it requires the full set of functionalities for all phases at the customer and the provider, yielding “heavy” applications. It may necessitate considerable installation efforts on the customer side if the customer does not have the possibility to exploit the required components at the provider, for example via a light-weight web interface which is not possible in all cases. Another disadvantage of the direct model is that the customer must contact all providers directly and cannot delegate tasks to an intermediary. The provider is in full control of the whole process but at the cost of having to provide all required functionality. The sources of revenue are clear since only the provider and no intermediaries are involved.

In the *A model* shown in Figure 2 the intermediary takes over the matching phase from the provider. To be able to do this the intermediary needs marketing information from the provider such as a description of the provider or individual products or a product catalog.

In the next model—the *AN model*—the intermediary provides negotiation service in addition to matching. For the negotiation service the provider must supply the intermediary with an additional information—the *pricing and discount model*. This model should enable the intermediary to negotiate with the customer in a meaningful way on behalf of the provider. Depending on the complexity of this model, negotiation can range from simple discounts for ordering a high number of products up to sophisticated models based on customer history, customer classification, etc. The choice depends heavily on the amount of information a provider wants to disclose to the intermediary.

Following the incremental approach the other models are defined in a similar way with intermediaries that take over more and more functionality of the provider. Depending on the business model the sequence of phases may differ from the sequence in the incremental model as described above. For example, payment may follow the delivery phase, or a

product might be delivered to a party without prior matching, negotiation, and ordering, on the basis of a party's profile and payment is performed after the party accepts the product. In principle any sequence of the presented phases is possible or phases may be skipped. However, such models can easily be transformed into the incremental model without changing their properties. Also the number of intermediaries involved may differ: One intermediary may be used for all phases or a dedicated intermediary may be used for each phase. For example, one intermediary may be in charge of all phases except for payment which could be done via the services of a credit card company.

So far we have implicitly assumed in the model that the intermediary takes over functionality from the provider. But in fact the model is symmetric so that the similar incremental models also can be derived from the above if the intermediary takes over functionality from the customer and acts on behalf of the customer. All concepts described above also apply to the symmetric model. For example, if the intermediary took over the matching phase from the customer (*A' model*), the customer would provide matching criteria to the intermediary. Additionally, both models may be mixed which provide the possibility to model very complex business interactions. For example, the customer may delegate matching and payment to two different intermediaries and the provider may delegate matching and delivery to two other ones. If additionally the sequence of phases is changed this scenario clearly models a very complex though not yet existing business case which, however, may enter daily business soon.

The rationale for introducing a new model as described above are that in fact we found no general model we could use for the i-commerce domain. Traditional e-commerce models exist but do not address the special requirements of i-commerce, for example, the new possibilities of combining information products into a new one through automatic processing, the special security requirements, the possibly very short life-time of intangible goods, and the new interaction patterns which are possible in the 3-party model. A detailed discussion of these motivating issues is given in [19].

The OPELIX model subsumes all well-known e-commerce models which can easily be mapped onto our model. The *eshop model* and *portal* (for one provider) correspond to the *direct model*. A *(process) portal* [27] and the *associated partner model*, e.g., Amazon's, can be mapped onto the *A model* (in [12] we show that this model is problematic because it depends on trust and cannot be secured technically). Several others, such as *(process) vortex*, dynamically trading processes, *third-party marketplace*, *(value-adding) reseller*, or *virtual communities*, require special consideration since no simple 1:1 mapping can be defined for them, i.e., their definitions, as given in [27] for example, are fuzzy and thus they can be mapped onto several counterparts in our model which emphasizes more precise definitions. A discussion of all possible mappings is given in [12].

Figure 3 shows the class diagram that represents the domain model of the information marketplace.

The business processes as discussed above can be viewed as business relationships which we identified as the central concept in the domain model. A *Business Relationship* is defined by the aggregation of actors and artifacts that are involved in the business process. Customers, intermediaries, and providers are modeled as actors with changing roles as

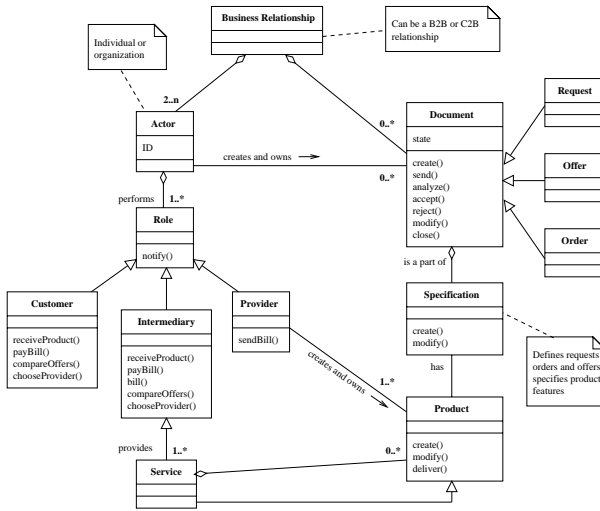


Figure 3: Information marketplace domain model

described in [28]. The classes Request, Offer and Order model the corresponding artifacts and are subclasses of the abstract class Document. Document's methods model the activities performed by customers, intermediaries, or providers that change the state of a particular document. An important part of each document is the specification of the requested, offered, or ordered information product. Thus Document is associated with Specification which specifies the product features related to Request, Offer, and Order.

Actor is associated with Document because an actor creates a particular document and sends it to another actor for processing. A customer can create requests and orders, a provider creates only offers, while an intermediary may create requests, offers and orders. A provider creates and owns a number of products which is represented by the association between Provider and Product. The association of Product with Specification models the fact that a product is described by its features.

The association between Intermediary and Service describes the basic characteristic of an intermediary which is to provide a service. Service inherits the properties of Product. Product's operations need to be overridden in Service since the processes of creating and modifying a service are quite different from those for creating and modifying a product. The number of actual services is currently unknown. Matching, negotiation, ordering, payment, and delivery are just examples of potential intermediary services. Further services of intermediaries such as classifying, combining, or mediating are discussed in [19].

Based on this model, we have concluded that communication and coordination between business actors, e.g., customers, providers, and intermediaries, is the key for the supporting system. To enable such communication, we need a standard language that ensures inter-operability. It should enable the exchange of requests, offers and orders among business actors. Each document should contain the information regarding involved parties and their roles in the business process, and describe the related information product and provided services. It is also crucial to enable the execution of the business process model that comprises prede-

defined actions. It is highly non-deterministic since business decisions and events trigger process actions. However, the process must be guided to comply with predefined business rules, which makes the supporting infrastructure design even more complex. The supporting system must also enable the composition of intermediary services. For example, an intermediary may decide to provide matching and negotiation to its customers. It therefore needs only those parts of the system that provide the desired functionality without losing inter-operability.

The presented domain model and its analysis have guided the definition of the architectural principles for the OPELIX platform. These principles and the architecture are addressed in the following sections.

3. DESIGN ISSUES

In Section 2 we have identified a number of phases that can take place in an i-commerce interaction. We have also argued that some of these phases may not be relevant in certain business processes and therefore they are skipped in such processes. Also, we have pointed out that the execution of each phase can be under the responsibility of different actors. An additional requirement that is not explicitly expressed by the domain model but represents a critical non-functional issue for the OPELIX platform is the ability to interoperate with different kinds of standards or existing COTS.

Based on the above considerations we have defined the main design goals of the OPELIX architecture as: (1) the ability to structure the system in terms of functional components autonomous from each other, each of which is in charge of a specific phase in the business model; (2) the ability to support loosely-coupled components; (3) the ability to run OPELIX installations ranging from a minimal set of components up to a full-fledged configuration; and (4) the ability to integrate different standards or products that provide specific services. In the following subsections we describe all above factors in detail.

3.1 Component orientation

The i-commerce environment is highly dynamic. Business conditions, business alliances, customer habits and preferences, and product and service offerings change rapidly. Businesses and their information systems also must be able to change rapidly to accommodate these changes. The only way we currently know how to do this is through the use of a flexible software architecture. The software architecture must reflect the business model so that it can accommodate changes to it relatively easily. For this reason, we have analyzed the i-commerce business processes presented in Section 2 for designing a component-based software architecture.

In the literature a component-based approach to software development is defined as a software development paradigm based strictly on the use of standard components, i.e., software is developed from efficient and generic components found in a standard software catalog [18]. In the OPELIX context we have associated a logical component with each of the main phases that characterize the i-commerce application domain. Moreover, we have defined a higher-level component that acts as a process support system by orchestrating the execution of the other components according to the adopted business process. Finally, we have identified a

set of lower-level components providing some basic functionality such as cryptography and storage support. All these components can be distributed and interact according to well-defined interfaces to provide proper decoupling and thus support clear separation of concerns among the components (see Section 3.2).

3.2 Loose coupling of components

Loose coupling of components has benefits both at development time and during system operation. At development time, it leads to clear separation among components and smooth integration effort. During operation of the system, it leads to the ability to dynamically reconfigure the system. These two aspects address two important requirements for OPELIX: the ability to integrate COTS for specific services and the ability to reconfigure the system depending on the business process being adopted. The second issue offers the possibility, for instance, that services are operated either by the intermediary or the provider or jointly by both of them.

A decoupling approach that is being studied with growing interest by researchers and practitioners is the publish/subscribe or event-based paradigm [24] and exploits asynchronous, anonymous communication among components. According to this approach, components communicate by generating and receiving event notifications. A component usually generates an event notification when it wants to let the “external world” know that some relevant event has occurred in its internal state. The relevant aspect in this process is that the event does not contain the list of its addressees: the event is propagated to any component that has declared interest in receiving it by issuing a subscription, which is an expression on the content of the event. When an event fulfills the condition stated by a subscription then the event is “compatible” with that subscription. An event dispatcher or bus is in charge of receiving the events, evaluating the subscription, and propagating the events to their listeners. The event propagation is completely hidden from the component that has generated the event as well as the receivers. The anonymity of notifications has two important consequences. First, a component can operate in the system without being aware of the existence of other components. All it has to know is the structure of the event notifications that are interesting to it, so that it can issue the necessary subscriptions. Second, it is always possible to plug a component in and out of the architecture without affecting the other components directly. These two effects guarantee a high compositionality and reconfigurability of a software architecture.

In the OPELIX platform the publish/subscribe approach is used in combination with point-to-point communication to guarantee component decoupling when needed. A first use of the publish/subscribe approach in OPELIX aims at managing the deployment of the OPELIX components. In fact, since they can be independently (un-)installed, components need some registration mechanism that allows them to make the other components aware that they are available. Whenever a component is started, it issues a registration message that is received by all the currently active components. It turns, such components issue a response event to notify the new-comer that they are alive. By exploiting the publish/subscribe approach, listeners interested in the registration messages simply need to subscribe to them and wait for their occurrence, while originators of such messages can

subscribe to the response and publish the registration event without being aware of the identity of the listeners.

Another important use of the publish/subscribe approach is to externalize all relevant state changes occurring in a component. Thanks to this externalization, all components can make decisions to ensure the progress of the business process. For instance, the dissemination and delivery component can trigger the execution of all actions needed to execute a payment.

3.3 Customized installations

As we have mentioned before, the execution of all business phases is not mandatory but depends on the adopted business model. Indeed, the way a business phase is executed may depend on specific situations such as the device the user employs or the configuration preferences defined by an intermediary or a provider. Finally, the distribution of ownership of services among the various actors can change dynamically and is not fixed. Thus the OPELIX platform can be installed and executed in several different configurations. A configuration does not necessarily provide the complete OPELIX functionalities. Of course, whenever a configuration is incomplete, some of the services provided by the complete installation can be missing, but this does not prevent the correct behavior of the other services.

For instance, a customer may not want to install specific software and hardware for handling payment. In this case, he/she will still be able to exploit the OPELIX platform, but only can access information and services that are free-of-charge or that exploit payment methods that do not require any software/hardware to be installed at the customer’s site. As another example, consider the case of an intermediary who does not want to negotiate with customers. In this case it will not install the component devoted to execute negotiation and the system will disallow customers to negotiate. However, the intermediary could decide to install the negotiation component at any time thus enabling negotiation. The registration mechanism presented in Section 3.2 guarantees that these requirements are addressed since it allows each component to discover the current configuration of the system at run-time. Moreover, it enables dynamic recognition of new-comers and proper reconfiguration of the entire system.

3.4 COTS integration and standard compliance

As we have mentioned, one of our goals is to guarantee interoperability with standards and COTS. At the same time we want to ensure that the conceptual model underlying the OPELIX architecture is not affected by the specifics of concrete solutions, especially when several alternatives exist, for example, SET and Millicent for payment. Therefore we defined and use a design framework that keeps specific tools separated from the core of the system. The main idea is that each OPELIX component implements a software layer that decouples all the other components from the specific tools it encapsulates. For instance, Figure 4 shows how the payment framework can integrate different payment mechanisms still maintaining the same interface to the external world. The framework is in charge of mapping requests coming from the external world into the format that is understood by the specific tool accomplishing the request, and vice versa. Note that by introducing this level of abstraction different

tools can coexist in an installation. The way one of them is selected for executing a certain service is defined within the logic of the component that encapsulates them and the data presented to it.

4. THE OPELIX ARCHITECTURE

The domain and business process models for i-commerce summarized in Section 2, together with the factors described in Section 3 led to the architecture of the OPELIX e-commerce platform shown in Figure 4.

The figure shows the full configuration for a single OPELIX installation at one site, for example, for an intermediary. In a practical setting OPELIX would be installed at every participating party's site enabling interaction to do business. The presentation in Figure 4 is arbitrary—we might as well have chosen to show components as “equal” peers connected via a communication medium, but we decided to show the components in a logically structured way as we see their “position” in the architecture. Figure 4 already points out clearly two main architectural principles of the OPELIX architecture: component-orientation and direct, peer-to-peer-like inter-component communication. An OPELIX installation consists of a set of independent components that can communicate freely among each other via a hybrid client-server/event-based communication infrastructure which will be discussed in Section 4.1. This allows each component to request directly the services it needs to fulfill its task. All service requests are done via the network so the OPELIX components do not necessarily have to run on the same host. In fact we envision that a typical provider or intermediary installation would be distributed over several specialized hosts. For example, a company would have a central security site to enforce its security policy. However, OPELIX does not constrain the user to any particular configuration. Additionally the OPELIX infrastructure exploits component-orientation to be dynamic and changeable over time, i.e., components may join or leave the installation at any time. For example, this means that the payment component could be temporarily shut down to do a software upgrade and join the system again afterwards without having to stop the overall system. OPELIX notifies components of such events (see Section 4.1).

As already mentioned above Figure 4 shows the full configurations for an intermediary or provider. In many cases this configuration is too heavy, e.g., for most customers, or some components may not be need at all. Since OPELIX only requires some minimal set of components to be operable the infrastructure can be adjusted to the user's need in a wide range. The minimum configuration of a customer consists of the user interface and the communication infrastructure. A provider/intermediary site requires the offer management and security components and the obligatory communication infrastructure to be operable.

The components supporting the business phases were derived from the 5-phases/3-party business process model (Section 2). The Targeting component (of a provider or intermediary) is in charge of actively announcing offers to customers and intermediaries¹ according to their profiles. Offers are expressed in OPELIX's Business Offer Language (BOL) [30]

¹Conceptually an intermediary has a customer and a provider side. To simplify the presentation we will only use the terms customer and provider in the following and imply that all statement made for those also apply to intermediaries.

which is discussed in Section 4.2. Targeting exploits the services of the Delivery and Dissemination Component (DDC) to reach customers. Conversely, customers can actively inquire for offers via the Request-and-Matchmaking (RMM) component. Inquiries are specified as incomplete offers in BOL. The RMM component is presented in Section 4.3. The negotiation component allows providers and customers to negotiate in an offer/counter-offer way which should narrow down to a completely specified offer both sides accept. Both interactive and automatic negotiation is supported. Having agreed on an offer triggers delivery or payment depending on the business model. The ordering phase of the process model need not be supported by a dedicated component because we assume that reaching an agreement closes a deal, i.e., both sides have a digitally signed contract, and implies ordering. The payment component offers a high-level generalized payment interface [8] which abstracts from the concrete payment infrastructure used such as SET, Millicent, etc. The DDC component does the same for any kind of delivery services.

The Business Workflow component supports the definition of the user's business process in terms of OPELIX's Business Offer (BOL) Language [30]. This component is a special component that directs the interaction of the other components within certain bounds.

The components on the left in Figure 4 represent the non-functional services we require in OPELIX. The User Profile component holds information on other users (mainly customers) to personalize services and information goods to their requirements and needs. User profiles are provided in part by the users but can be augmented by provider-side information such as statistical information, implicit preferences derived from analysis of interaction patterns, etc. Currently this part of the user profiles is provided offline through logfile analysis. However, adding on-the-fly analyses is straightforward. Users have full control of all stored information in their profiles and can change it as they prefer to meet EU privacy regulations [7].

Typically vendors wanting to sell information already employ a DBMS or a content management system (CMS) so we did not include one in the architecture. The “interface” to these and any other kind of information stores is provided by the Offer Management component. It stores all user-defined offers and links information and services to offers via Uniform Resource Identifiers (URIs) which provides a high degree of flexibility. Offers are discussed in detail in Section 4.2.

The User Access component (UA) provide the user interface to the OPELIX and all its components. The individual interfaces of the OPELIX components are integrated by the UA and augmented by supporting user interface functionality, for example access to configuration options or user feedback. To comply to current user interface standards and to support remote access and distribution we tried to use a standard Java-enabled web-browser as far as possible.

The security component is in charge of all security concerns of the infrastructure. Specifically it offers a new copyright protection mechanism for textual data [26], secure (encrypted) and authenticated data transmission via SSL (including message origin authentication and message integrity), entity authentication, access control (to OPELIX), transaction authentication, key management and digital signature

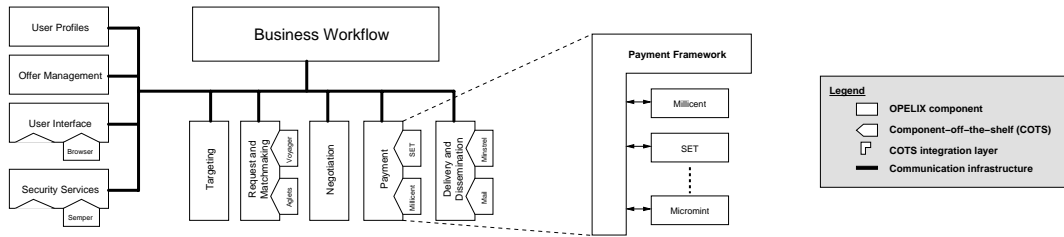


Figure 4: OPELIX architecture (full configuration)

services. In the following sections we discuss key components of the architecture.

4.1 Inter-component communication

Some communication aspects of the OPELIX architecture have already been discussed in Section 3. The components of the OPELIX platform communicate and cooperate via a common communication infrastructure. In principle any component can communicate directly with any other component to request its services. Two communication types can be found in OPELIX: (1) heterogeneous communication between different OPELIX components inside one installation, for example, when the DDC requests a signature check from the security component, and (2) homogeneous communication between peer components of different OPELIX installations, for example, if the customer payment component communicates with its counterpart at the provider to settle a payment. Additionally, both types of communication can be synchronous or asynchronous depending on the requirements of the components and the purpose of the communication. Synchronous communication is used in OPELIX whenever continuation without the result of an action is not possible or useful. For example, the DDC should wait for a content-signing operation if it is required to send signed content. Asynchronous communication—which in contrast to synchronous communication requires additional efforts for request/reply mapping—is applied otherwise. For example, if the DDC performs a delivery operation to 10000 subscribers the requester should not be blocked but be notified upon completion. The communication patterns can be 1-to-1 or 1-to-many and since the initiator of a communication can vary most components can act both as client and server (similar to peer-to-peer). The communicating components may reside on different hosts even if they belong to the same installation. If communication is done via the Internet then additional security requirements, network delay and bandwidth must be addressed.

The communication technologies that are used in OPELIX are: (1) the Java Event Distribution Infrastructure (JEDI) [3] for intra-site notifications (asynchronous heterogeneous communication); (2) Java RMI for service requests and communication between components of one OPELIX installation (synchronous heterogeneous communication); (3) XML via HTTP for homogeneous communication (between identical components in different OPELIX installations); (4) all messages are XML documents. Our decision was based on the following goals: (1) communication should be open and rely on standards; (2) the infrastructure must be lightweight to be usable on the client-side; (3) communication must be efficient (large data sizes, asynchronous where pos-

sible, etc.); and (4) the best communication paradigm for a certain communication requirement should be used. The choice of XML-based messages was obvious since XML is one of the base technologies of OPELIX. For homogeneous communication we decided to use XML via HTTP because these technologies are widely-used, open standards. For heterogeneous communication we considered RMI, SOAP and XML RPC, but decided to use plain RMI because SOAP and XML RPC impose additional installation requirements on customers while this communication type is invisible to the outside world.

4.2 Business Workflow

A major requirement derived from the analysis of Section 2 is the need to support a wide range of potential business processes in a flexible way. To meet this requirement we developed the Business Offer Language (BOL) [30] which allows the OPELIX user to describe the business process. The Business Workflow component interprets this declarative specification and interacts with (or directs) the other OPELIX components to execute the business process as defined by the user. In this way the OPELIX infrastructure can be parameterized to virtually any business process.

BOL's design is based on several assumptions and prerequisites that are presented in the following. BOL is designed around the concept of *information products* which denotes any type of electronic information that can be exchanged (plain information, payment, certificates, etc.). *Actions* on information products are defined as message exchanges between the trading partners. BOL allows the user to declare such information products, describe the trading partners and define the actions. The workflow itself is expressed through constraining actions by conditions on the information product, the process state and time. So an action may be valid in certain process states and at certain times within the workflow execution and invalid otherwise. BOL provides a minimal set of actions on information products to enable both parties to express their interest in an information product. In case of an agreement this imposes an *obligation* to perform a specific action on an information product. Enabling both parties to express their agreement to an exchange is part of this minimal set. This defines which actions become obligatory in a certain process state. BOL distinguishes between *explicit operations* which result in a message exchange and *implicit operations*. The exchanged messages can be derived from the specification. Thus the model allows to determine the possible communications from the specification.

To provide an idea of how the BOL works in practice we discuss a simple example: a customer has to pay for an

information product he/she requested from an information vendor's web site. The relevant parts of the BOL specification are given in the example below. For processing, this specification is mapped onto the BOL DTD and is provided to the interpreter as an XML document.

```

roles: customer, vendor;
goods:
  payment(amount: NUMBER): customer → vendor;
  information(url:STRING): vendor → customer;
rules:
  → deliver(customer, vendor, payment(a));
  delivered(customer, vendor, payment(a)) →
    request(customer, vendor, information(url));
  ...
substitutions:
  delivered(customer, vendor, payment(a), t) ⇒
    promised(vendor, customer, information(url));
  ...

```

First the involved roles (customer, vendor) are declared followed by a specification of the exchanged goods (payment, information) and the direction of the exchange. The goods are parametrized, for example, the payment with an amount. For each product three standard actions can be performed: request by the receiver, promise and deliver, by the provider. Request and promise are used to agree on the product parameters which renders an obligation for the provider to perform the deliver action with all parameters bound to the agreed values. As stated above, actions can be constrained by rules which implicitly define the business process. A rule's conditions can refer to actions performed on specific goods and may yield an explicit state change (reflected by predicates requested, promised, delivered). For example, the rule in the above BOL example defines that the customer may request the information after having delivered the payment. Additionally, implicit state changes that are not a consequence of executing one of the actions, i.e., that occur without exchanging a message, can be specified by substitution rules. For example, the substitution rule in the above BOL example means that if a payment is received by the provider, this implies its promise to deliver the information. In case this information has been requested earlier an obligation occurs. This is BOL's way to express conditional obligations.

The Business Workflow component interprets such BOL specifications, communicates actions between the trading partners, and triggers the execution of other OPELIX components. In particular, the business workflow component maintains the states of ongoing processes, maps the BOL's information product specifications to the encoding for communicating with other components, interprets and enforces the rules specified in BOL, determines potential successive actions based on the rules and the current state of a process, computes the next state based on a selected action, triggers other OPELIX components, and synchronizes state changes with remote Business Workflow components.

BOL specifications are transformed into an execution model by analyzing the declarations of information products and rules. If an action is selected the next state is computed and the remote Business Workflow component is informed about the state change. The trading partner receives the action, checks its validity, computes its next state, derive the next potential action and communicates this action back to the first trading partner. Here the same checks occur and

the next state is computed. Based on the new state the next potential actions can be derived and visualized in the user interface. The user may now choose how to continue or automate these decisions. For robustness and traceability reasons states are stored persistently.

A complete description of BOL is given in [30].

4.3 Request and Matchmaking

As described in Section 2 the matchmaking service can be offered by providers or by intermediaries. In the second case the specific intermediary collects and returns all results (*offers*) for a single customer. Portals are already offering such service, but they have 3 limitations: (1) results are not specifically customized to the customer, (2) searches typically are done only on the contacted intermediary (not propagated to others), and (3) none of the existing tools supports *combined searches*, i.e., searches where the overall result depends on what is found in the individual steps of the matchmaking process on several providers, for example, an opera performance, e.g., "La Traviata", a direct flight from the customer's home city (e.g., New York) to the city of the performance, and a five stars hotel in that city as a combined result assuming that each part is offered by different providers / intermediaries. The advantage for the customer is that he/she just sends out a single request to a single intermediary and the system takes care of satisfying it.

The main purpose of the Request and MatchMaking component (RMM) is to enable combined searches by analyzing and combining information provided by different providers / intermediaries. The user defines a query via the RMM which is then given to *searching agents* that in turn are able to produce a result by executing multiple queries on different sites, each of which provides part of the overall result. Searching agents can either query servers remotely or perform the queries on a remote server or they can move from server to server to collect results (offers). The final step is constructing a combined offer from other (possibly combined) offers found. Offers are described in XML and are structured according to some criteria defined in what we call *offer types*. Offer types are also defined in XML and are known to any party providing the matchmaking service. The customer can be guided in building a request that is compatible with the existing offers. The RMM translates user requests into a *Request Definition Language* (RDL) statement which is understood by the searching agents (the example below shows part of an RDL request for the opera example given above).

```

APPLY '//show[name = 'La Traviata' &and$ status = 'Available']?/'
(city/date/price)' AS 'performances'
AT 'tcp://site1.domain1.com:10000', 'tcp://site3.domain3.com:10000'
USING_POLICY 'complete'
THEN APPLY '//flight[arrival/date < %performances$show.date% &and$
arrival/place = %performances$show.city% &and$
departure/place = 'New York']?/'
(code/price|departure/date)' AS 'flights'
AT 'tcp://site2.domain2.com:10000', ...
USING_POLICY 'complete'

```

Agents interpret an RDL statement by executing the following steps: (1) identify all objectives to be fulfilled; (2) for each objective, query proper sources in our example, the sites providing information on theaters, airlines, and hotels; and (3) assemble the results. To speed up requests an agent may create *slave agents* to parallelize sub-parts of the request: Slave agents are sent out while the *master agent* waits for them to come back with their result on the send-

ing site. RMM consists of mobile searching agents which are currently implemented on top of Voyager and the “stationary” components at every site that hosts the agents and interacts with a site’s offer management component on the agents’ behalf. At the moment we are developing an additional software layer that will decouple the agent logic from the mobility platform to support multiple platforms. The RMM component is described in detail in [21].

4.4 Delivery and Dissemination

The Delivery and Dissemination component (DDC) offers a uniform interface to all services necessary to ship information such as offers or information products to customers. For example, Targeting exploits the DDC to send offers to customers, and the Business Workflow instructs the DDC to ship an information product after having received the required payment from the customer. The DDC is intended to comprise traditional services such as distribution via email or download from a web server via a username/password scheme and new approaches, for example push systems. For the OPELIX project we developed the uniform interface including support for integration of COTS as described above and then enhanced the Minstrel push system [8] for information dissemination to support a new distribution medium type in OPELIX.

Minstrel follows the component and communication model described in [11]. Its central central communication concepts are channels that offer topic-specific information, a hybrid broadcasting algorithm which is supported by a communication-transparent transport system to provide scalability to large numbers of customers. Customers can subscribe to channels of their interest or are subscribed to them by an OPELIX component, for example Targeting, and then can receive a continuous information feed (for one-time delivery email or download would be used). The hybrid communication algorithm actively notifies customers of the availability of new data (push) and the customer then can initiate the transfer of the possibly large payload information from its nearest Minstrel access point. A detailed description of the broadcasting algorithm is given in [10].

Minstrel is implemented in Java using servlets and works with any Java servlet-enabled web server (we use Tomcat) and the current version of the protocol employs human-readable XML messages via HTTP. It supports active push distribution, is scalable in terms of users and network bandwidth consumption and provides secure transmission and content authentication/integrity via SSL. It allows providers to send executable Java code which can extend the receivers capabilities while the customer is protected from malicious code by the JSEF security framework [13]. Additionally it integrates the generalized payment model [9] we developed for OPELIX (we evaluated a pay-per-view business model with Minstrel). A detailed description is given in [10, 8].

5. RELATED WORK

The e-commerce application domain is currently attracting the attention of a plethora of researchers and practitioners who are proposing new approaches, tools, and standards aiming at solving several aspects of the business phases composing an e-commerce transaction. The existing proposals can be classified in the following categories:

- Standards and protocols to support interoperability

between B2B systems. They aim at defining protocols to enable exchange of various types of data concerning products and business transactions between two parties. This field is well-studied and exploits the experiences of many companies gained from the application of EDI. The objective is to build less expensive EDI-like systems on top of the Internet.

- Platforms enabling the development of specific e-commerce applications. These platforms may offer a set of components which can be parameterized or extended with user code to meet an application’s requirements. These components typically offer communication abstraction and typical functionality needed by e-commerce applications, for example, content management, advertisement, etc.
- Applications developed from scratch by using basic development tools. These systems typically target a specific, narrow application domain and offer a limited degree of re-usability. We do not discuss them further.

5.1 Interoperability between B2B systems

A number of existing approaches focus on interoperability mechanisms for e-commerce applications. In the following we overview them and relate them to the OPELIX approach.

OBI (Open Buying on the Internet) [22] is a specification aiming to “automate high-volume, low-dollar transactions between trading partners”. It can be seen as a replacement of traditional EDIs as far as interaction between buyer and seller organizations are concerned. The specification defines a standard purchasing process, a standard format for information about orders, and standard methods for transmission of orders and for managing security of the entire transaction. The specification assumes that an agreement exists between interacting parties that establishes all terms of a purchase. Differently from OPELIX, OBI is therefore mainly focused on managing transactional aspects in a well delimited context. Security and authentication are very critical aspects in this context and are described in detail in the specification.

RosettaNet is an XML-based standard for supply chain management in the information technology and electronic component industries. RosettaNet specifies Partner Interface Processes (PIPs) that describe business processes between trading partners. PIPs are XML messages that include a business document and a business process description. Current PIP specifications are based on peer-to-peer message exchange between e-business applications. To enable message exchange, RosettaNet defines business and technical dictionaries and an implementation framework. The directories define a common set of properties for PIPs. The RosettaNet Implementation Framework (RNIF) [25] provides exchange protocols for quick and efficient implementation of PIPs. RosettaNet is a standard focused on the communication between the two trading partners whereas OPELIX is a system that enables simple and flexible deployment of i-commerce services.

BizTalk [1] is a framework proposed by Microsoft that supports interoperability between applications by encapsulating all exchanged business data into *BizTalk documents*. Each of these documents contains information such as deadlines and retransmission requirements. The main components of the framework are the *BFC servers* that manage proper transfer of BizTalk documents between two parties which may

belong to different organizations. Each organization must have a BFC server that acts as an intermediary between the internal information system and the other BFC servers. Servers are responsible for (un-)wrapping business data in BizTalk documents if the application itself is unable to do so. Also, they are responsible for properly managing deadline expiration by discarding the corresponding documents, and for ensuring reliable communication. BizTalk is specifically focused on providing advanced services for transferring business data, while it does not make any assumption on the structure of data being transferred. So it can be viewed as complementary to many of the interoperability approaches we discuss in this section.

Universal Description, Discovery, and Integration (UDDI) [29] is a specification for distributed Web-based information registries that offer information about business services and their interfaces. UDDI registries can be viewed as meeting places where a business can discover other business services and also publish information about its services. UDDI specifies an XML schema that defines the information needed for a business service description. Part of the service description are the so-called "technical fingerprints" that specify a service's programming interfaces. A service wishing to interact with another service must conform to the specified interface. UDDI also defines the API for interacting with UDDI registries which uses SOAP via HTTP and XML messages as its communication mechanism. UDDI could be used in OPELIX for registering and describing the services offered by OPELIX providers and intermediaries.

Electronic business XML (ebXML) [4] is an initiative to "create a single global electronic market." ebXML standards try to combine the business process experiences of EDI with the flexibility of XML. Similar to other approaches, the ebXML technical architecture defines a messaging service and a registry for sharing the information between the trading partners. The ebXML *Business Process and Information Model* distinguishes ebXML from other standards. It enables a trading partner to describe its business process using the ebXML's *Specification Schema* in the form of an XML DTD, or UML diagrams. ebXML heavily relies on existing standards such as SOAP, UML, XML, and UDDI. UDDI can be used for interaction with the distributed ebXML registries. OPELIX could be extended to implement the ebXML standard messaging service and use its Business Process Model for sharing the process information between the trading partners.

A large number of XML-based frameworks for e-commerce are classified and compared in [31]. The approaches presented in this section are complementary to OPELIX. They offer a vocabulary and a registry service that enables a business to advertise its services and discover adequate business partners. The vocabulary defines the ontology that enables business partners to communicate by exchanging structured information. Most of the listed approaches provide the definition of the communication protocol and the corresponding message format. In principle OPELIX could exploit any of the above protocols and approaches to guarantee interoperability between providers and intermediaries.

5.2 E-commerce development platforms

E-Speak [6] is an infrastructure to build e-commerce applications designed upon the notion of a web service. It supports B2C and B2B business models through service reg-

istration, service discovery and interaction of dynamic Web services. E-Speak uses the concept of vocabularies for defining a service and builds its discovery functionality upon this concept. It also enables secure communication with firewall traversal capability. The business process supported by e-Speak is the following: a provider registers its service with a service directory, and a customer can find the registered service by querying the existing directory. The service directory has the characteristics of a marketplace since it acts as a broker, i.e., it serves as a meeting point for customers and providers. An example service directory implementation is provided in [5].

The main difference between the OPELIX and the e-Speak systems results from the different project goals. OPELIX offers the implementation of various services, such as advertising, negotiation, ordering, delivery, and payment, and flexible service composition. In e-Speak each service provider needs to develop its e-Speak compliant service using the Java e-Speak Service Interface (JESI) library. In OPELIX we focus on the implementation of different i-commerce intermediaries, while e-Speak offers support for the brokerage service only. In fact, OPELIX and E-Speak target the same domain but at different levels: E-Speak provides an advanced communication platform for e-business but the user has to build services from scratch whereas OPELIX offers all domain-specific components to start building i-commerce systems at a high level by parameterizing existing functionality. OPELIX includes a similar communication system as e-Speak as described in Section 4.1 and in fact e-Speak was one of our candidate communication platforms for OPELIX.

BroadVision's One-To-One [2] and on Intershop's Enfinity [15] focus on selling tangible goods. They support configurable business processes and provide customization for personalizing the system. This results in several different business models supported by both platforms, such as e-shop models, market places, and portals with special focus on personalization. In addition, both systems provide component-based extension capabilities similar to OPELIX. The difference between the OPELIX framework and these systems is OPELIX's focus on information commerce and its resulting focus on flexible business models. This means flexibility in terms of defining processes at product level whereas these two systems specify business rules at application level. Furthermore, OPELIX supports the concept of super-distribution, i.e., a buyer of an information good (legally) copies it, distributes it to others and those can also buy the product. Since a similar scenario is not possible with tangible goods, the existing shopping systems do not address this issue.

Other related platforms are Sun Microsystems's iPlanet [16] and iMediation's iChannel [14]. iPlanet is a suite of products to enable web-based e-commerce sites. Its functionalities are a subset of One-To-One's and Enfinity's. iChannel is a set of products to develop portals for (re-)seller's but there is not enough information available for evaluating it.

6. SUMMARY AND CONCLUSIONS

This paper has presented the domain and business process models for information commerce, and the software architecture for the OPELIX system that supports information commerce applications. The domain model uses the concept of intermediary to simplify the interactions among providers; in principle, each intermediary can provide a dis-

tinct and separate functionality needed to support and enhance customer-provider transactions. Layers of intermediaries can provide a complete i-commerce solution. The business process model consists of five independent phases that may be combined in different ways and assigned to customer, intermediary, or provider depending on the business model.

Analysis of the domain model and the business process model lead to a clean software architecture for the OPELIX system presented in Section 4. The dynamic nature of i-commerce requires ease of (re-)configuration and rapid evolution. To meet these requirements, the architecture is component-based and uses standard communication protocols. The components support all business phases of the process model, are intrinsically distributed, and may be reconfigured at runtime, for example to meet changing business or performance conditions. The components are loosely-coupled, use open standards to communicate, and provide interfaces for plugging in existing components such as for payment or security. A powerful business offer language supports the declarative definition of business models and the configuration of components into a complete system supporting the business model. The architecture combines a number of state-of-the-art technologies: mobile agents, event- and push-based communication, declarative business process definition language, XML based communication over HTTP, and a new copyright technique for digital text.

We have recently completed a prototype implementation of the architecture and have delivered it to two different companies. They are currently using it to implement two different i-commerce applications. Our experience in building the prototype implementation was surprisingly uneventful. In particular, since the components were built by different project members in different locations, we had expected a longer than usual integration effort. On the contrary, the integration of the components took almost no time. Although we cannot be sure of the reasons behind this unexpected success, some potential reasons are due to the architecture which supports clear separation of concerns into distinct components and uses the relatively uncoupled event-base communication. Another potential reason is the use of standard tools and technologies, such as Java, HTTP, XML, and their supporting tools. These technologies have rapidly become standard tools of the trade and have created a common framework and language, easing communication and interchange among implementers.

Acknowledgements

The authors would like to thank the OPELIX team, Harald Gall, and René Klösch.

7. REFERENCES

- [1] BizTalk website. Microsoft Corporation, 2001. <http://www.biztalk.org/>.
- [2] BroadVision website, 2001. <http://www.broadvision.com/>.
- [3] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, **27**(9), September 2001.
- [4] ebXML Consortium. ebXML Technical Architecture Specification v1.0.2, February 2001. www.ebxml.org/.
- [5] HP e-services village website, 2001. <http://www.hpespeak.com/esvportal/index.jsp>.
- [6] E-speak website, 2001. <http://www.e-speak.net/>.
- [7] Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, 1995. Official Journal of the European Communities of 23 November 1995, No. L281.
- [8] M. Fischer, M. Haberl, M. Hauswirth, and M. Umlauf. Minstrel: a component-oriented push system for Internet-scale information distribution, 2001. Distributed Systems Group, TU Vienna.
- [9] M. Fischer and M. Hauswirth. Towards a generalized payment model for Internet services. Technical report TUV-1841-01-03. Distributed Systems Group, TU Vienna, May 2001. <http://www.infosys.tuwien.ac.at/reports/repository/TUV-1841-01-03.ps>.
- [10] M. Hauswirth. *Internet-Scale Push Systems for Information Distribution—Architecture, Components, and Communication*. PhD thesis. Distributed Systems Group, TU Vienna, October 1999.
- [11] M. Hauswirth and M. Jazayeri. A Component and Communication Model for Push Systems. *ESEC/FSE '99*, September 1999.
- [12] M. Hauswirth, M. Jazayeri, and M. Schneider. A Phase Model for E-Commerce Business Models and its Application to Security Assessment. *HICSS-34*, January 2001.
- [13] M. Hauswirth, C. Kerer, and R. Kurmanowytsch. A Secure Execution Framework for Java. *7th ACM Conference on Computer and Communication Security*, November 2000.
- [14] iMediation website, 2001. <http://www.imediation.com/>.
- [15] Intershop website, 2001. <http://www.intershop.com/>.
- [16] iPlanet website. Sun Microsystems, 2001. <http://www.iplanet.com/>.
- [17] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.
- [18] M. Jazayeri. Component programming: a fresh look at software components. *ESEC '95*, September 1995.
- [19] M. Jazayeri and I. Podnar. A Business and Domain Model for Information Commerce. *HICSS-34*, January 2001.
- [20] D. Konstantas and J.-H. Morin. Trading digital intangible goods: the rules of the game. *HICSS-33*, January 2000.
- [21] E. Di Nitto, C. Ghezzi, M. Sabba, and P. Selvin. Using agents in performing multi-site queries. *Fifth International Workshop CIA-2001 on Cooperative Information Agents*, September 2001.
- [22] OBI consortium website, 2001. <http://www.openbuy.org/>.
- [23] OPELIX website. OPELIX Consortium, 2001. <http://www.opelix.org/>.
- [24] D. S. Rosenblum and A. L. Wolf. A design framework for Internet-scale event observation and notification. *ESEC/FSE '97*, September 1997.
- [25] RosettaNet Implementation Framework: Core Specification. RosettaNet Consortium, July 2001. <http://www.rosettanet.org/>.
- [26] M. Schneider and T. Keinz. Proof of Authorship for Copyright Protection in OPELIX. *Electronic Imaging & the Visual Arts (EVA 2001)*, March 2001.
- [27] A. P. Seth, W. van der Aalst, and I. B. Arpinar. Processes driving the networked economy. *IEEE Concurrency*, **7**(3), 1999.
- [28] C. Shapiro and H. R. Varian. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, Mass., USA, 1999.
- [29] UDDI Technical White Paper. UDDI Consortium, September 2000. <http://www.uddi.org/>.
- [30] A. Wombacher and K. Aberer. A language for information commerce processes. *Third Workshop on Advanced Issues of E-commerce and Web-based information Systems*, September 2001.
- [31] Y. Zhao. XML-based Frameworks for Internet Commerce. Department of Computer and Information Science, April 2001. Licenciate thesis.