An Architecture for Information Commerce Systems*

Manfred Hauswirth, Mehdi Jazayeri,
Zoltán Miklós, Ivana Podnar
{mh, mj, zm, ip}@infosys.tuwien.ac.at
Distributed Systems Group
Technical University of Vienna, Austria

Elisabetta Di Nitto
dinitto@elet.polimi.it
CEFRIEL
Milano, Italy

Andreas Wombacher
wombach@darmstadt.gmd.de
IPSI, GMD
Darmstadt, Germany

Abstract

The increasing use of the Internet in business and commerce has created a number of new business opportunities and the need for supporting models and platforms. One of these opportunities is information commerce (i-commerce), a special case of e-commerce focused on the purchase and sale of information as a commodity. In this paper we present an architecture for i-commerce systems using OPELIX (Open Personalized Electronic Information Commerce System) [11] as an example. OPELIX provides an open information commerce platform that enables enterprises to produce, sell, deliver, and manage information products and related services over the Internet. We focus on the notion of information marketplace, a virtual location that enables i-commerce, describe the business and domain model for an information marketplace, and discuss the role of intermediaries in this environment. The domain model is used as the basis for the software architecture of the OPELIX system. We discuss the characteristics of the OPELIX architecture and compare our approach to related work in the field.

1. Introduction

The Internet and the Worldwide Web have spurred the generation of a large amount of information and made possible its easy distribution over space, time, and in many forms and modes. But the ease of publishing and the universal availability of information have transformed the Internet into a complex, even chaotic environment. Due to an increasingly large number of information sources the producers and potential users of information are disconnected with no natural way of finding out about each other. Moreover, the users of information face another substantial problem: once the required information source is located, there is no simple way of assessing its dependability. Conversely, information providers are discontent with their means of reaching potential customers.

As a solution to these problems we envision an *information marketplace*, a virtual location where providers offer *information* and *services* as goods and customers can find, evaluate and buy the desired information. In this context, information is the commodity being traded. Service is the facility offered to the user and performed by a service provider.

The information marketplace is an environment for information commerce (i-commerce). I-commerce is a special case of e-commerce focused on the purchase and sale of information. In i-commerce terms, information is considered a product that can be represented as bits of data (e.g., weather forecasts, daily news, stock quotes, etc.). Information products are intangible and have no physical presence. I-commerce is also known as "trading of intangible goods" [9]. Yet, information products have intrinsic value and may be traded just as tangible goods. Compared to tangible products, they are costly to produce but cheap to reproduce. It is easy to redistribute and update such products. This implies that copyright issues become extremely important and require special consideration.

The information marketplace offers a setting for the introduction of new innovative services. In such environment we envision the growth of intermediaries, business actors that offer services to both customers and providers. Search engines are a very simple form of intermediary: they help customers in finding providers. Portal sites are another kind of intermediary: they classify providers and help customers in searching based on some classification. We envision many other services that intermediaries may perform. Indeed, a possibly unlimited number of intermediaries may exist, each adding value to an information product, or performing a service for either customer and/or provider.

In this paper we present the requirements of an information marketplace by defining a domain model that identifies business actors, and products and services provided by specific actors. We use the domain model as a starting point for designing the architecture of an information marketplace software system. Next we present the characteristics of the OPELIX architecture, and discuss its design and implementation decisions. Finally, we discuss and compare the OPELIX architecture with related initiatives and platforms. The paper is organized as follows: Section 2 discusses the role of intermediaries in the information marketplace. In section 3 we

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

describe the domain model that was used as the basis of the OPELIX architecture. OPELIX's architectural principles are presented and discussed in Section 4. Section 5 relates our approach to other work in the field. The final section summarizes lessons learned and lists issues for further work.

2. Intermediary

By examining a simple transaction on the Internet that conforms to an information marketplace business process, we can identify a customer looking for a product and a provider offering that product. After locating the provider the customer sends a request and waits for provider's response. This simple business transaction was used since the early days of Web. However, the activity "find a provider" has significantly changed due to tremendous Internet growth. To deal with the overwhelming increase in the number and variety of providers, a number of services appeared that help the customer in finding providers. Search engines and portals are examples of such services. To take into account the role of such sites in the interactions between customers and providers, we introduce another information marketplace actor called *intermediary*. The intermediary, in this case, adopts the activity formerly performed by the customer and enables simple discovery of potential providers. Finding a provider is only one of the services an intermediary may perform. We envision the emergence of new intermediary services that will become part of the information marketplace business processes.

By concentrating on the role of the intermediary, we can identify a number of services that either exist today or may be offered as new businesses. To help analyze intermediary services further, we can classify intermediaries in terms of their services. We have identified the following classes of intermediaries [8]:

Classifying (e.g., portals): these intermediaries provide a classification of available providers or products.

Filtering: these services provide a refined view of existing providers or products. For example, a digest service could provide a summary of information from other sites or a specialization service could provide specific information available from many providers.

Qualifying: such intermediaries provide a qualification service. For example, a particular service could rate the reliability of information products available from other providers. Such a service could be used by customers to choose what providers they use and by providers to improve their services.

Authenticating: these intermediaries authenticate the identities of customers, providers, and products.

Combining: these intermediaries combine the information products available from other providers. For example, an "evening planning service" could provide information about the starting time of an opera performance, the traffic conditions around the theater, and recommended restaurants.

Brokering (e.g., search engines): these intermediaries help customers and providers find each other.

Mediating: These intermediaries help customers and providers to conduct a business. For example, a mediator could offer anonymous transactions so that the customer and the provider do not learn one another's identity. The difference between a broker and a mediator is that the broker helps in identifying customers and providers but the mediator helps in carrying out the process of a transaction between a customer and a provider.

These intermediary services are not only useful functions but they also enable more sophisticated business models for enterprises. For example, *indirect commerce* is a business model where affiliate companies utilize web sites and/or push systems to get in touch with possible customers. *Added-value commerce* is a model where businesses sell higher-level information products by filtering or combining information sources.

3. Information marketplace domain model

The domain model identifies business *actors* involved in marketplace business processes, describes their interactions, and discovers *artifacts* that are used and produced during interactions. We have two major goals for our domain modeling procedure: to understand the context of information marketplace, and to identify potential intermediary services in the information marketplace environment.

Domain modeling is the initial step of the software development process that enables the understanding of the system environment and is primarily used for defining terms and enabling developer communication and cooperation. We have decided to employ the use case driven modeling approach defined in [7] as a "technique for understanding the business processes of an organization." In our case the organization is the information marketplace.

The use case driven modeling approach employs UML [2] use case diagrams for describing the context of the information marketplace. The use case diagram associates information marketplace actors and a number of use cases. Use cases are usually described by UML activity diagrams or sequence diagrams that model organizational business processes. They list the interacting actors, their activities, and artifacts produced during the business process, providing the input for the construction of the class diagram which communicates the domain model in a programmer-like style.

The reminder of this section presents the use case diagram and the class diagram that comprise the information marketplace domain model. The detailed description of the modeling procedure is given in [8].

3.1. Use case diagram

Figure 1 depicts the use case diagram that describes the information marketplace context. The diagram relates a customer, a provider, and an intermediary performing a generic service.

In our model an intermediary performs services for both customer and provider, and acts as a connection between them. 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 to find a dependable provider by advertising products on behalf of a provider.

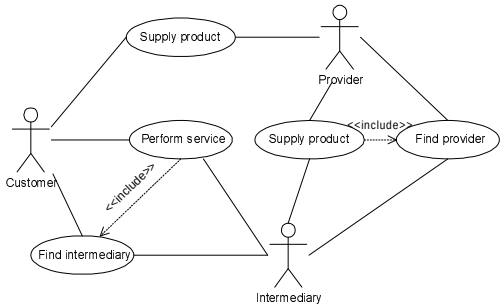

Figure 1: Use case diagram relating the actors of the information marketplace

After locating the provider, the customer can choose to conduct business directly with the provider. This situation is modeled by the use case Supply product that connects customer and provider. A customer may, however, communicate only with an intermediary. In this case, the intermediary delivers products and handles payment on behalf of the provider. Thus intermediary service grows in complexity: The intermediary interacts with the customer and performs a service, but also interacts with the provider. We model the interaction with the Supply product 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. In this initial modeling step we think of it as a generic service. The Intermediary can potentially offer an unlimited number of services. However, a typical business process consists of a combination of the following phases: advertising, negotiation, ordering, payment, and delivery [6]. The Intermediary can provide a service that enables one, or a combination of the listed business phases. For example, an intermediary can perform advertising on behalf of a provider. To be able to do advertising, the intermediary needs a description of provider products, i.e., a product catalog from the provider. A provider may, in another scenario, entrust both advertising and negotiation to an intermediary and supply the intermediary with the pricing and discount model that will guide the negotiation process. In another case, an intermediary may offer a number of payment methods, e.g., different micropayment protocols, as a service, or enable delivery via push systems. It can also combine products coming from several providers and autonomously offer combined products to end customers. These are just some of the examples of possible intermediary services that can become part of the information marketplace business processes.

To support the business process execution, the marketplace needs to facilitate inter-operability. The exchange of standard messages that are understood by all business actors is a prerequisite for automatic communication. In an information marketplace, the actors should be able to exchange requests, orders, and offers to negotiate business terms related to delivery and payment. They should also be capable of delivering products to their customers, and collecting revenue for their services.

3.2. Class diagram

The class diagram that represents the domain model of the information marketplace is given in Figure 2. It was created based on the information acquired during the use case analysis.

We are considering business processes of the information marketplace that relate a number of business actors. Business relationship is the aggregation of actors and electronic documents that are involved in the business process. Customer, intermediary, and provider are modeled as actors with changing roles. Classes Request, Offer and Order model the corresponding artifacts. They inherit the properties of the abstract class Document since these artifacts can be regarded as electronic documents. 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. Document is therefore associated with Specification which specifies product features.

Actor is associated with Document since an actor creates a particular document and sends it to another actor for analysis. A customer can create requests and orders, a provider creates only offers, while intermediary may create requests, offers and orders.

A provider creates and owns a number of products. An association between Provider and Product models this relationship. Product is described by its specification: Product is therefore associated with Specification.

The association between Intermediary and Service describes the basic characteristic of an intermediary: providing 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. Advertising, negotiation, ordering, payment, and delivery are just examples of potential intermediary services.

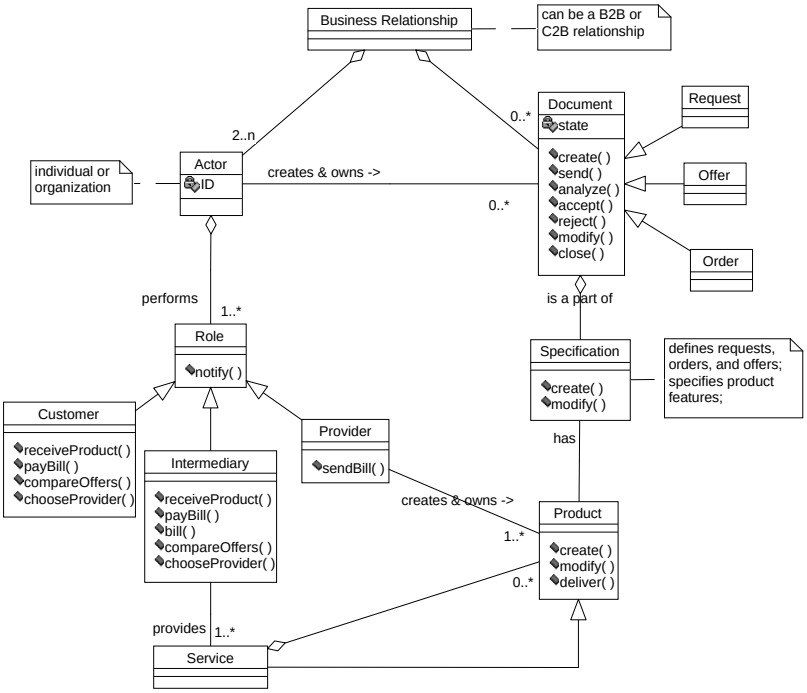

Figure 2: Domain model for the information marketplace

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 predefined actions. It is highly indeterministic since business decisions and events trigger process actions. However, the process must be guided to comply to predefined business rules, which makes the supporting infrastructure design even more complex.

The composition of intermediary services is the other premier requirement that the supporting system needs to address. For example, an intermediary may decide to provide advertising 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 architectural principles for the OPELIX platform. These principles and the architecture are addressed in the following section.

4. Architecture

The domain model [8] and the phases model [6] can be mapped onto a component-oriented architecture easily which resulted in the current OPELIX architecture. We tried to follow the domain model as far as possible but of course had to adjust the design to the specific needs of our industrial partners.

4.1. Design goals

The main design goals were component-orientation and inter-operability. Component-orientation was targeted in two ways: We wanted to enable the use of existing components off-the-shelf (COTS) and make OPELIX itself component-oriented to allow OPELIX users to configure the platform according to their requirements.

The support for COTS is critical since we did not want to re-implement existing functionality, for example payment services such as Millicent or SET, but exploit existing components. This led to a simple architectural pattern for the integration of COTS as shown in Figure 3 for the payment component.

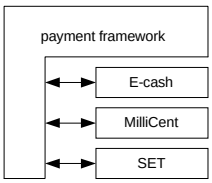

Figure 3: Framework approach of OPELIX

For every component (or service) which uses COTS the framework provides an external interface for other OPELIX components and an internal interface to the components used in a specific configuration. This allows us to abstract from the specific interfaces of concrete components (such as concrete payment services) and offer a uniform interface for

other components. Thus other OPELIX components only have to deal with a single interface for every component regardless of the COTS used.

At first we were considering to base our framework on existing component frameworks such as Enterprise JavaBeans [10] or CORBA [12]. However, we had to step back from this approach because we wanted to use the framework both on the client side and at the server side. At the server (provider) side typically enough computing and networking resources are available and (expensive) licenses are affordable. On the client side, however, we wanted to have as little prerequisites as possible that an installation would have to satisfy. It turned out that our requirements are much simpler than the wealth of functionalities offered by component frameworks, so we dropped this approach. Now the architecture requires Java and a web server at the server side (plus required COTS), and Java and a web browser at the client side.

The second main goal for the design of the OPELIX architecture was to have a flexible and extensible framework that could be easily tailored to the configurations needed by the user. For example, if a site did not want to support payment or advertising we do not require it to install unnecessary components. In contrast, we wanted to have a very minimal set of components to be required and allow a user to add further components as needed by his/her business application or to delegate such functionalities to intermediaries [6].

4.2. The OPELIX architecture

On the basis of the above considerations we developed an architecture composed of loosely-coupled cooperating components, communicating over a common communication infrastructure, which is intended to support intermediaries to establish a virtual marketplace where (1) information from providers is offered to customers; (2) intermediaries can (re-)combine/enhance these products to produce new ones which are also offered; (3) which allows customers to easily find and buy the data he/she is looking for; and (4) which enables providers to delegate the sale of his/her products to intermediaries.

The OPELIX architecture shown in Figure 4 (maximum configuration for an intermediary) uses a problem-oriented amalgamation of well-established architectural styles. This means that we have adopted and used those architectural styles that best address the concrete requirements.

The unifying principle of the whole OPELIX architecture is component-orientation: Every component should be exchangeable if its interface specifications are obeyed. This component-based approach facilitates the distribution of components in a network and supports integration of third-party components. Additionally, it provides a simple way for configurations tailored to the users' needs and adjusts the size and functionality of a site's installation to the requirements and resources. OPELIX components are independent of each other and can directly request services from the other components inside the same installation.

OPELIX provides a set of services that fulfill functional and non-functional requirements of i-commerce applications. Each component is devoted to accomplish a specific functionality and may rely upon or interact with other components of the architecture. The components have been identified on the basis of user requirements that where expressed as UML use cases and descriptions by the industrial partners in the OPELIX project.

The functional components (Targeting, Request and Matchmaking, Negotiation, Payment, Delivery and Dissemination) implement the phases described in [6]:

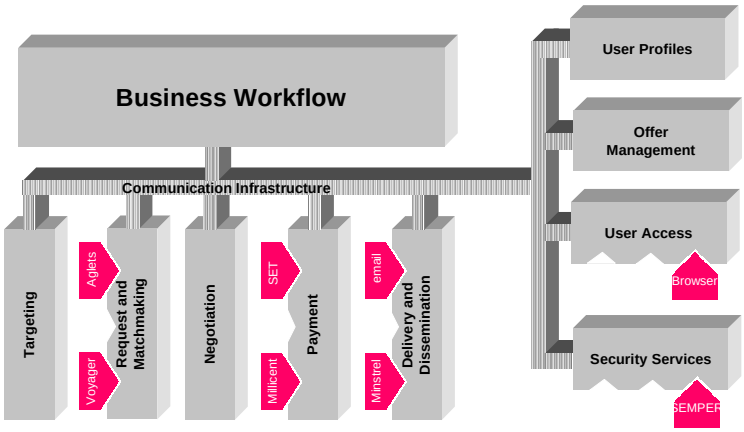

Figure 4: The OPELIX architecture

Targeting provides advertisements and offers to potential customers based on their previous actions or their preferences (user profile). It lets the intermediary define the rules to be used in marketing campaigns and relies on the Delivery and Dissemination component to transmit these data to the targeted customers. Additionally it provides the User Access component with the advertisements that might be interesting for a customer when the customer accesses the OPELIX system.

Request and Matchmaking allows the user to define a request as a set of constraints on product characteristics. The user can indicate which constraints are negotiable. On the basis of such specifications this component searches through available products and matches requests to offers. The searching process may span over different producers' sites, depending on the way content is managed and on the type of the request.

Negotiation supports parties in negotiating about requests and offers. For example, the customer can negotiate the payment method or the delivery date to be used. The outcome may be an agreement for a "contract." The final decision of accepting or rejecting the negotiation is up to the customer.

Delivery supports the delivery of products and advertisements to interested parties based on several paradigms (email, push, etc.) For the OPELIX project we focus on push delivery.

Payment provides an interface to different payment systems such as Millicent and SET.

The supporting components (User Profile, Offer Management, User Access, Security) provide additional management services for user profiles and offers, restrict access to the system and ensure the integrity and security of the platform:

User Access allows the user to access the system via authentication services provided by the Security Services component. It provides the top-layer user interface for any user type, enables the access to the user interfaces of the other components and collaborates with some of them to complete their functionality.

Offer Management supports administrators in managing offers and directories of content producers. It provides facilities to add value to existing content (for instance, through categorization or improvement in data presentation layout).

User Profiles stores user data and preferences such as personal data, list of products recently acquired, etc. Profiles can be accessed and modified by the intermediary and the customer.

Security Service offers a range of security related services to the other components: authentication to properly identify interacting parties and prove the origin of products; authorization to control user access to resources and data; non-repudiation ensures that parties cannot repudiate the actions they have performed in the fulfillment of a contract; and copyright services for data which cannot be water-marked (text, programs) [13].

The small plug-in components shown in Figure 4 denominate COTS that could be used with OPELIX.

4.3. Inter-component communication

The components of the OPELIX platform communicate and cooperate via a common communication infrastructure. The cooperation among the components is controlled by the Business Workflow which executes the user's business model defined in the Business Offer Language (BOL) [1] which was defined in the course of the OPELIX project. The Business Workflow component executes the workflow describing the structure of interaction between the intermediary and the other parties involved in a trade. It activates the other components of the system when their services are needed according to the state of the workflow being enacted. It also keeps track of and updates the status of the business process.

Two types of communication among the components inside the OPELIX system can be distinguished: (1) heterogeneous communication between different OPELIX components inside one OPELIX installation and (2) homogeneous communication between (components of) different OPELIX installations. This distinction does not preclude anything about the actual distribution of the components. The communicating components could actually reside on different hosts, although they may belong to one OPELIX installation.

Additionally communication can be synchronous or asynchronous depending on the requirements of the components and the purpose of the communication. The following sections discuss the types of communication which are relevant to OPELIX and the technologies we selected.

4.3.1 Synchronous vs. asynchronous 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. In synchronous communication the mapping of requests and replies is implicit: A requester waits for the reply, so the reply can only be for that request.

Asynchronous communication is useful if continuation without the result of an action is possible or useful. For example, if the DDC performs a delivery operation to 10000 subscribers the requester would not wait for the completion but wants to be notified upon completion. Asynchronous communication requires explicit request/reply mapping, e.g., by means of request IDs.

4.3.2 Heterogeneous communication

Heterogeneous communication denotes communication between components of different types inside one OPELIX installation, for example, when the business workflow requests a delivery from the DDC. Although frequently communication is not over a public network, security must be addressed. For example, not everyone in an organization should be able to trace all communication such as price information. Heterogeneous communication can either be synchronous, for example, when the DDC requests encryption from the security component, or asynchronous, for example, the notification of completion of a requested service.

4.3.3 Homogeneous communication

Homogeneous communication denotes communication between components of the same type in two or more OPELIX installations; for example, communication between the customer payment component and the intermediary payment component to settle a payment. Since communication between two installations typically occurs over the Internet, network delay, bandwidth, data size, and security must be taken into account when choosing the communication paradigm. Homogeneous communication in OPELIX can be synchronous, for example, in the case of negotiation or contracting, or asynchronous, for example, in the case of Request and Matchmaking (RMM) or delivery.

4.3.4 Technologies used in OPELIX

The communication technologies that are used in OPELIX are: (1) the Java Event Distribution Infrastructure (JEDI) [2] 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 as open as possible and rely on standards; (2) the infrastructure must be light-weight to be usable on the client-side; (3) communication must be efficient (large data sizes, asynchronous where possible, 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 notifications we wanted to rely on an existing event-based infrastructure and decided to use JEDI since it satisfies our requirements, is light-weight and available from a project partner which allows us to adopt it to our requirements easily. For homogeneous communication we decided to use XML via HTTP because these technologies are widely used and are open standards which facilitates communication between OPELIX systems of different vendors if they adhere to the communication standards. Additionally, it puts minimal requirements on the availability of network support and security. For heterogeneous communication we considered RMI, SOAP [14] and XML RPC [16]. On the basis of an evaluation of SOAP and XML RPC we decided to use plain RMI. The main reason for this decision was that SOAP and XML RPC require considerable implementation efforts by the industrial project partners while this communication type is invisible to the outside world.

4.3.5 Communication patterns

Communication among OPELIX components in one installation is 1-to-1 and 1-to-many. The paradigm used – client-server, event-based, synchronous or asynchronous – depends on the requirements of the communication partners. For example, requesting a signature check from the security component is done synchronously in client-server style because the result of the security operation determines the further processing. Additionally, asynchronous communication would impose further security requirements such as authenticity or replay checks of the reply messages and mapping replies to requests.

Other operations such as requesting a payment from a receiver are done asynchronously although the (third-party) payment protocols used internally by the payment component may communicate synchronously with their counterpart. We use asynchronous communication if possible, so that requesting components are not blocked unnecessarily. Asynchronous communication in general is beneficial for long-lasting operations where the requester can receive a reply later.

We use event-based communication for asynchronous notification of (multiple) components. A component can subscribe to certain events and will be notified if a matching event occurs. For example, a component may request a delivery and multiple parties may be interested in its completion. So these components would subscribe to the according event. This also supports the goal of a highly flexible architecture because no assumptions on the number and kind of components present in the system are made. The only requirement is that a component should subscribe to the events it is interested in.

In the case of communication between OPELIX installations three communication partners exist: producer, intermediary, and consumer. They communicate in a point-to-point style using the client-server pattern. Each of them can act as a client or as a server. For example, in the targeting scenario a consumer would be a server and a producer would be a client; in a payment scenario the same consumer can be the client of the producer. External communication can be request-reply style, involve sessions, or even be 1-to-many depending on the purpose.

5. Related work

The area of e-commerce has attracted much interest recently. eCo and ebXML are the most prominent projects that are relevant to our work. The eCo project [4] provides an architecture that will enable the interoperation of a heterogeneous set of e-commerce systems. The eCo architecture is not intended to represent any specific e-commerce system nor does it try to give a general model for e-commerce applications. Rather, this architecture is designed to represent those aspects of an e-commerce system that contribute to its interaction with a prospective trading partner. To achieve inter-operability, the eCo specification does not require that businesses agree on what they do or how they do it, it helps them describe what they do. The eCo architecture is a layered model representing an e-commerce environment.

ebXML [5] is an ongoing project of the United Nations (UN/CEFACT) and OASIS, with broad industrial support. The motivation of the project is to provide an alternative for systems based on the *Electronic Data Interchange* (EDI) standard. The vision of ebXML is to enable enterprises of any size to conduct businesses with each other through the exchange of XML based messages. Although the project goals are different from OPELIX, there are some similarities in the approach. ebXML also defines a mechanism for describing *Business Processes and Information Meta Models*. Companies submit their *Trading Partner Profile* to a repository, which enables them to find trading partners. Trading partners can optionally negotiate possible business scenarios. If the partners agree on business rules, they can conduct business electronically. The entire technical documentation of the project is still unavailable, so a detailed comparison of ebXML and OPELIX is not possible at the moment.

A comparison of architectures of current e-commerce platforms is beyond the scope of this paper. A detailed analysis and comparison is given in [15].

Currently available e-commerce platforms are not specifically designed for i-commerce. They can be used for selling information products or services as a special case. We know of very few systems that specially focus on trading intangible goods and offer a similarly rich functionality as OPELIX: Project MEDIA (Mobile Electronic Documents with Interacting Agents) [9] aims at building such a system based on mobile agents. MEDIA defines a model and a framework for trading digital documents. The design of the framework focuses on the protection of the intellectual rights of the document owner.

6. Conclusions

We have presented the architectural principles of the OPELIX platform, a representative of an emerging class of systems dedicated to information commerce. OPELIX provides an open information commerce platform that enables enterprises to produce, sell, deliver, and manage information products and related services over the Internet. The OPELIX architecture is open, flexible, customizable, and extensible. We presented the main components and communication mechanisms in OPELIX.

OPELIX is an ongoing research project that is currently in the implementation and integration phase. The validation of the approach will be done through case studies in cooperation with our industry partners.

Acknowledgments

We would like to thank the OPELIX team for providing a fruitful environment for discussions.

References

[1] Aberer, K., Wombacher, A., A Language for Information Commerce Processes, to appear in: 3rd International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, San Jose, California, USA, June 21-22, 2001.

[2] Booch, G., J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, Reading, Massachusetts, USA, 1999.

[3] Cugola, C., Di Nitto, E., Fugetta, A., The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS. To appear in IEEE Transactions on Software Engineering.

[4] eCo, http://www.commerce.net/eco, 2001.

[5] ebXML, http://www.ebxml.org/, 2001.

[6] Hauswirth, M., M. Jazayeri, 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, 2001.

[7] Jacobson, I., G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, Reading, Massachusetts, USA, 1999.

[8] Jazayeri, M., I. Podnar, A Business and Domain Model for Information Commerce, Proceedings of the 34th Hawaii International Conference on System Sciences, Maui, Hawaii, 2001.

[9] Konstantas, D., J.-H. Morin, Trading digital intangible goods: the rules of the game, Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[10] Matena, V., Stearns, B., Applying Enterprise JavaBeans: Component-based Developmnt for the J2EE Platform, Addion-Wesley, 2001.

[11] OPELIX consortium, OPELIX website, http://www.opelix.org/, 2001.

[12] Orfali, R., Harkey, D., Client/server programming with Java and CORBA, John Wiley & Sons, 1998.

[13] Schneider, M., Keinz, T., Proof of Authorship for Copyright Protection in OPELIX. EVA 2001.

[14] Simple Object Access Protocol (SOAP). http://www.develop.com/soap/.

[15] Shim, S.S.Y., V.S. Pendyala, M. Sundaram, and J. Z. Gao, Business-to-Business E-Commerce Frameworks, IEEE Computer, **33**(10), October 2000.

[16] Wallnöfer, H., XML-RPC Library for Java, http://classic.helma.at/hannes/xmlrpc/.