Modeling Business Media Platforms

Markus Greunz, Katarina Stanoevska-Slabeva
Institute for Media and Communications Management
University of St. Gallen
firstname.lastname@unisg.ch

Abstract

The high speed at which new businesses are developed can to a large extent be attributed to their ability to flexibly combine existing services into an integrated business platform. The development of such platforms requires the representation of the business architecture in a model which can be implemented in a software solution. However, aside from diagramming techniques little help is available. Hence, guidelines are needed on how to model the architecture of the prospective business platform which considers the necessity for flexible service combination. The objective of the paper is to present guidelines for modeling business media platforms which are based on both an underlying reference model providing a general architecture and the Unified Modeling Language (UML) as descriptive technique.

1. Introduction

Intent observers of the developments in the so called “new economy” surely have noticed the development, and just recently also the decline, of numerous new businesses. Of course, lots of those businesses will not survive but nevertheless an interesting part of this developments is the speed at which they have evolved. This speed has been fueled by flexible underlying software architectures on which either new enterprises have emerged or existing enterprises have expanded into new markets by quickly combining existing services. Grocery stores open banks and insurance companies, i.e. Tesco, or insurance companies open hospitals or car breakdown services [1]. New businesses have been rapidly assembled as loosely coupled sets of partnerships and services, with complex business processes spanning many organizations. To a large extent this has been enabled by flexible IT platforms that combine existing services in order to support new business processes.

We define platforms like these as media platforms. Media platforms enable the exchange of information or other objects such as goods and services [2]. According to the motivation for the exchange a differentiation can be made between business and knowledge media. In this paper the focus will be on business media. Business media are defined as media platforms that enable the exchange of goods and services and serve as the basis for economic value creation [3]. A business media platform is therefore synonymous to conventional e-commerce applications while media platforms in general have a wider scope including any form of information exchange which does not have to be economically motivated.

In contrast to conventional information systems, business media platforms do not merely support businesses but are a fundamental part of them. Therefore, it is of key importance to effectively transform the business architecture into a corresponding platform. An effective approach to achieve this goal must therefore start from requirements defined by the business architecture which describes how the enterprise is going to provide value to its customer. According to [4] the key elements of a business architecture are organization, processes and technology. In order to describe how these key elements are to be realized in an implementation it is necessary to capture them in a descriptive representation in the form of a model. However, aside from diagramming techniques, there is often little guidance towards reaching well structured system models [5]. Hence, guidelines are needed on how to model the architecture of the prospective media platform which correctly captures the three key elements. These guidelines aid the development team in identifying the key characteristics to be modeled as well as in how to represent them in a notation. Furthermore, the guidelines should lead to models that provide loose coupling between the business focused parts of the model (organization and processes) and the model of the technical aspects. This is important since business media platforms can be implemented on top of many different technologies or by assembling and integrating available services, which should not influence their core processes and organizational structure.

The objective of the paper is to present guidelines for modeling business media platforms. The approach presented is based on the Media Reference Model (MRM)
[2] which provides a general architecture of business media platforms and serves as the conceptual framework for the modeling domain. Following the proposed guidelines it shall be possible for analysts and developers to document the platform models and to communicate them among developers, managers, programmers and anyone else. For obvious reasons, the description technique used for the models should be simple and easily comprehensible, though, expressive enough for the task at hand. On the one hand it should be programming language independent and on the other hand it should support the implementation in a straightforward manner [5]. In order to meet these requirements we use the Unified Modeling Language (UML) [6] as the notation.

The paper is structured in the following way. The next section provides an introduction to the Media Reference Model and briefly describes its four layers. Section three describes an example business media platform which shall be used for illustrating the proposed modeling guidelines in the subsequent section. Section four gives a detailed explanation of how to use UML to model the aspects covered by each of the layers of the MRM. Finally, we discuss the presented work and future research directions.

2. The Media Reference Model

To abstractly structure business media platforms we use the Media Reference Model (MRM) [2]. It provides four views and four phases (see figure 1). Following, the views and phases are explained briefly as they provide the foundation for the further discussion. Further information about the MRM can be obtained from [2] and [7].

The Community View describes the potentially interested business community which defines the structural organizational of the platform. The business community consists of a set of agents, i.e. users, that interact with the platform. Agents can be either organizations, humans, or artificial agents (i.e. software agents). The organizational structure is defined in terms of roles played by the agents, as well as protocols that define a number of rules that describe the sequence of possible actions that roles can take on the platform. In order to offer valuable business services to users of the platform the underlying processes of the implementation view are used.

The Implementation View describes the procedural organization of the platform in terms of processes. These processes are designed to implement the services offered to the roles and are constrained by the protocols. The processes themselves rely on a number of services that implement specific pieces of functionality needed for the successful execution of the process. These services are described in the Transaction View.

The Transaction View describes a set of services upon which the processes rely in order to provide the platform services to the community. There is a minimal set of services that has to be present in order to support a complete market transaction. In other words, for each phase in a market transaction there must be at least one supporting service providing the functionality that implements the minimal requirements of the specific phase. The phases are described in more detail further below. In order to combine the services of the different phases the technological infrastructure is used which is described in the infrastructure view.

The Infrastructure View describes the underlying technical infrastructure encompassing communication, transaction, and middleware services. It describes the functionality for the secure exchange of information as well as the transaction management that ensures the secure execution of business processes that encompass several services.

According to the definition of business media platforms agents interact with a platform in order to exchange goods. We call this exchange process a market transaction. The MRM distinguishes four phases of a market transaction [2].

In the Knowledge Phase the agents get information about the platform itself, the goods offered by suppliers, the community that is part of the platform, and other information that is necessary in order to perform transactions successfully. Access to the information is enabled by services such as information catalogs, directory services, and authentication services.

In the Intention Phase the agents signal their intentions, derived from the knowledge in the knowledge phase, and from their desires and goals. Offer, counter offer and demand are the prevailing form of expressed intentions in business media. Services like electronic product catalogs (EPC), order books and search engines support this phase.

In the Contracting Phase the agents negotiate selling contracts starting from the initial offers presented in the intention phase, i.e. by means of an EPC. The result of this phase is a legally binding contract which documents the agreed upon obligations of supplier and buyer in terms of traded products, prices, delivery and payment conditions. Negotiation services and auctions are examples for services supporting this phase.

In the Settlement Phase the agents have to act according to the negotiated contract. This involves the payment for the products or services purchased and the delivery of the products to the purchaser. The settlement is supported by payment and logistics services.

The views covered in the MRM reflect the key elements of business architectures – organization (community view), processes (implementation view) and technology (transaction and infrastructure view).

The paper is structured in the following way. The next section provides an introduction to the Media Reference Model and briefly describes its four layers. Section three describes an example business media platform which shall be used for illustrating the proposed modeling guidelines in the subsequent section. Section four gives a detailed explanation of how to use UML to model the aspects covered by each of the layers of the MRM. Finally, we discuss the presented work and future research directions.

2. The Media Reference Model

To abstractly structure business media platforms we use the Media Reference Model (MRM) [2]. It provides four views and four phases (see figure 1). Following, the views and phases are explained briefly as they provide the foundation for the further discussion. Further information about the MRM can be obtained from [2] and [7].

The Community View describes the potentially interested business community which defines the structural organizational of the platform. The business community consists of a set of agents, i.e. users, that interact with the platform. Agents can be either organizations, humans, or artificial agents (i.e. software agents). The organizational structure is defined in terms of roles played by the agents, as well as protocols that define a number of rules that describe the sequence of possible actions that roles can take on the platform. In order to offer valuable business services to users of the platform the underlying processes of the implementation view are used.

The Implementation View describes the procedural organization of the platform in terms of processes. These processes are designed to implement the services offered to the roles and are constrained by the protocols. The processes themselves rely on a number of services that implement specific pieces of functionality needed for the successful execution of the process. These services are described in the Transaction View.

The Transaction View describes a set of services upon which the processes rely in order to provide the platform services to the community. There is a minimal set of services that has to be present in order to support a complete market transaction. In other words, for each phase in a market transaction there must be at least one supporting service providing the functionality that implements the minimal requirements of the specific phase. The phases are described in more detail further below. In order to combine the services of the different phases the technological infrastructure is used which is described in the infrastructure view.

The Infrastructure View describes the underlying technical infrastructure encompassing communication, transaction, and middleware services. It describes the functionality for the secure exchange of information as well as the transaction management that ensures the secure execution of business processes that encompass several services.

According to the definition of business media platforms agents interact with a platform in order to exchange goods. We call this exchange process a market transaction. The MRM distinguishes four phases of a market transaction [2].

In the Knowledge Phase the agents get information about the platform itself, the goods offered by suppliers, the community that is part of the platform, and other information that is necessary in order to perform transactions successfully. Access to the information is enabled by services such as information catalogs, directory services, and authentication services.

In the Intention Phase the agents signal their intentions, derived from the knowledge in the knowledge phase, and from their desires and goals. Offer, counter offer and demand are the prevailing form of expressed intentions in business media. Services like electronic product catalogs (EPC), order books and search engines support this phase.

In the Contracting Phase the agents negotiate selling contracts starting from the initial offers presented in the intention phase, i.e. by means of an EPC. The result of this phase is a legally binding contract which documents the agreed upon obligations of supplier and buyer in terms of traded products, prices, delivery and payment conditions. Negotiation services and auctions are examples for services supporting this phase.

In the Settlement Phase the agents have to act according to the negotiated contract. This involves the payment for the products or services purchased and the delivery of the products to the purchaser. The settlement is supported by payment and logistics services.

The views covered in the MRM reflect the key elements of business architectures – organization (community view), processes (implementation view) and technology (transaction and infrastructure view).
Furthermore, it considers explicitly the use of existing services in order to support the business processes as part of the technology aspect. This service-oriented view reflects the current developments as businesses are taking increasingly advantage of existing components and in the future may flexibly incorporate web services [8].

**Figure 1.** The Business Media Framework [2]

In its current state of development the MRM describes media platforms at an abstract level. However, it is of limited use in the development of platforms. In this paper we provide guidelines on how to model each of the views by means of UML diagrams which should allow to support a structured modeling process in the development of a platform. In order to illustrate the proposed guidelines the next section briefly introduces an example of a business media platform of which specific aspects are modeled in the subsequent section.

### 3. Example: The StreamCom platform

The results presented in this paper have evolved from the still ongoing StreamCom project [9]. The aim of this academic project is the development of a prototype for a platform for the secure online distribution of video streams with a guaranteed Quality of Service (QoS) support based on a pay-per-use schema. In addition to the technical implementation there is also a strong focus on finding guidelines of how to model platforms like this that can be followed in other projects.

The community view of the StreamCom platform consists of two roles: customer and stream provider. A customer can search for video streams in a catalog and select a stream to view. The stream provider is able to publish the streams he provides, he must broadcast them at the scheduled time, and has to provide decryption keys to the customers who have paid for viewing them.

The implementation view of the StreamCom platform has to define at least the following process: first the stream provider watermarks and decrypts a stream [10]; then the bandwidth for the scheduled broadcast time is reserved obeying the QoS reservation protocol; then he publishes the stream in some form of catalog obeying the catalogs publishing protocol; once the stream is published the customer can select a stream obeying the search/select protocol; then the customer needs to buy a ticket for it according to a macropayment protocol; this ticket contains payment units representing the price paid for the ticket; at the time the stream is broadcasted the ticket has to be presented and is used for providing micropayments to the stream provider for the time viewing the stream obeying the micropayment protocol [11]; for each received micropayment token the necessary decryption key is sent to the ticket owner; after the stream is broadcasted the money for the received micropayment tokens has to be redeemed by the stream provider using the ticket redemption protocol. As can be seen each role has to obey a number of protocols in order to execute the video stream distribution process.

The transaction view describes the services necessary to support the video stream distribution process. For the StreamCom platform these are a catalog service for publishing and searching streams, a ticket brokering service for purchasing tickets for streams, a QoS brokering service for establishing bandwidth reservation, a micropayment service for paying with tickets and a macropayment service for paying tickets and redeeming the money for the micropayment tokens.

Finally, the infrastructure view describes the communication infrastructure used for the StreamCom platform and other fundamental technology which are not listed here.

This example gives a brief overview of the architecture of a business media platform for video stream distribution. The following section provides more detail on what has to be captured in the single views of the MRM and how it can be modeled using UML. The focus will be put on modeling the bandwidth acquisition of the stream provider.

### 4. Modeling the views of the MRM using UML

According to Stritzinger [5], modeling is a task that covers all phases in the software life cycle. Each phase takes as input the model from the previous phase and adds some details until the model can be implemented. This means, however, that it is not defined how the modeling task itself can be structured.

Based on the MRM we propose to structure the modeling task according to the four views of the MRM. The characteristics of each view are modeled by suitable
UML diagrams. As the development progresses details are added to each of the view models. Therefore, in each development phase and in each iteration in the development project the whole model is continually updated giving at any time a consistent representation of the envisaged system at the current level of detail. As the views of the MRM provide a good separation of aspects to be considered for a platform it is not necessary to follow a fixed modeling order. This may be left to a development process model describing when to produce which model. For the following exposition a simple top-down approach is used which might be preferred especially in the analysis phase. Therefore, the following subsections describe how to model the single views in a top-down order starting with the identification and description of the relevant business community.

4.1. Modeling the Community View

Gathering system requirements usually starts from the perspectives of the users, i.e. the community, considering how they interact with the system in order to obtain value. Business media platforms can be accessed by a very heterogeneous group of agents that may follow very different objectives when using the platform. For example, buyers are interested in finding a product that satisfies their needs while sellers are interested in promoting and selling as many products as possible. Furthermore, the business may also be subject to monitoring by any third parties (i.e. governments may be interested in transactions involving weapons) whose user requirements also need to be captured. Other user groups may be possible depending on the specific business model.

The central notation in UML to capture functional requirements is the use case diagram. Use cases describe all the required interactions of users – which can be human users as well as software systems [6] - with the system to be constructed. A use case describes users of a system as actors. Use cases also include a set of preconditions which specify what must be true in the system for the interactions to be feasible. Furthermore, a set of postconditions states what must be true at the end of the interactions. [12]

Actors in UML are very similar to roles as defined in the community view of the MRM as each actor has a set of functionality to interact with the system. In order to correctly capture the community it is necessary to model all possible kinds of roles that may use the platform. These are usually at least sellers and buyers but may also include roles such as a platform operator, a trusted third party and others. The use case diagram illustrated in figure 1 shows the two basic roles of the example in section 3 – a customer interested in buying and viewing video streams and a stream provider selling the streams. It is also possible to define subroles, i.e. a customer may be a guest, a registered customer or a regular customer. Furthermore, for each role it is necessary to determine the functionality it offers to its agents by means of use cases. There are four basic use cases in any business media platform which follow the four phases of a market transaction. These generic use cases have to be refined for each phase. The use case diagram in figure 2 refines the settlement phase only in which the customer can view the stream and has to pay for it following the pay-per-use schema. The stream provider has to encrypt the video stream, send it, reserve bandwidth, and redeem the money from the micropayments. Eventually, the functionality of a role as defined in the use cases needs to be detailed to an extent that it can be expressed as a set of operations. Preconditions provide a way to indicate explicitly what is required by the roles in order to perform their actions. Postconditions indicate what must be true for a role after successfully performing a use case.

Figure 2. Use case diagram showing details of how the roles may interact with the platform in the settlement phase of the market transaction

The use case diagram should comprehensively define all roles of the user community as well as the functionality of each role. However, there are more aspects to be covered in the community view. These include the description of the agents and their characteristics, the model of the roles, the model of the data that has to be exchanged during the market transaction as for example
the product catalog or ontology, and other static aspects of the system domain. These aspects are best modeled by class diagrams. Following we briefly describe how to model some of these aspects. A more detailed discussion can be found in [13].

Agents interact with a platform playing a specific role. While the role defines what an agent can do on the platform it is also necessary to model the agents themselves as they capture the user’s data, i.e. user credentials, demographic data, etc. This data provides context information about the user of the platform which is important for authentication, personalization and the evaluation of business policy rules.

We use class diagrams to model agents and their characteristics as they describe static aspects of the system. This is comparable to modeling business objects representing the concepts of agent (i.e. a person or organization), knowledge (i.e. address, ...), intentions (i.e. purchase order, ...), contracts (i.e. selling contracts, service level agreements, ...) and supply (i.e. product, currency, ...).

Roles that have been identified in the use case diagram are also part of the static description of a platform. Therefore, they are taken over to the class diagram for further refinement (which is no problem since actors are only stereotyped classes). This allows to establish further relationships between roles, i.e. subrole relationships. Use cases only allow the definition of subroles in the form of subtypes. This, however, may not always be practical and may be overcome by modeling them in the class diagram using explicit subrole associations. Use cases are good for capturing functional requirements but do not allow for the specification of non-functional requirements such as security. However, each role is associated with a set of permissions, i.e. access rights to use specific functionality of the platform. Following the Role-Based Access Control (RBAC) model as described in [14] it is possible to capture the access permissions of roles following one of the UML modeling approaches described by [15] and [16]. In addition, roles define qualifications that an agent must fulfill in order to be allowed to play that specific role. This is expressed as specific characteristics that the agent must possess.

Of primary importance for a complete model of the community view is the model of the logical space [2], that is, a model of syntax and semantics of the terms used by the communicating entities. This model provides the common language of the community describing the common domain of discourse as a foundation for common meaning in terms of syntax and semantics. This model may be as simple as a glossary or thesaurus but may also comprise full ontologies. The model of the logical space does not only describe the syntax and semantics of content provided on the platform but is also used to describe the platform itself, e.g. the agents, roles, and protocols. This helps agents to get information about how to act on the platform but may also aid the development team building up a common meaning of the concepts used in the model. Currently we model concepts and their definitions and terms and their syntactic representation in the form of class diagrams [13].

The community view defines the structural organization of a media platform. It captures all static entities that are part of the platform comprising agents, roles and the logical space. Any dynamic aspects are deliberately not considered in this view. This decoupling should provide more flexibility for changes in either the structural or the procedural organization. The linking element between the community view and the implementation view are the protocols which are associated to each role. Protocols define the rules of action in the medium which have to be obeyed by the roles. They therefore influence the dynamic behavior of the platform which is modeled in the implementation view.

4.2. Modeling the Implementation View

Agents interact with a media platform by playing a specific role. The functionality needed by the roles in order to implement their interfaces is available in the form of operations offered by the services of the transaction view. The implementation view describes how the behavior (i.e. the functionality) defined by the roles is to be implemented based on the functionality provided by the services of the transaction view [2]. The interaction between the roles and the service interfaces is dictated by the rules defined in the protocols, which are therefore the linking elements between the community view and the implementation view.

Protocols are defined over roles of agents [17]. Therefore, each role is associated with its protocol according to which it can perform on the platform. The protocol constrains the sequences of action (i.e. the allowable sequences of operation calls) that have to be obeyed by the specific role. These constraints eventually define the sequence of actions, i.e. the business processes, that are offered to a role. Therefore the protocols of all roles define the procedural organization of the media platform and define the number of possible business processes. For example, a business process for a customer role on the StreamCom platform would be the sequence of actions that leads to the consumption of a video stream. In order to achieve this outcome, the customer has to obey the protocols that define how to search for a stream, how to obtain a ticket, etc.

The rules defined in the protocols have the form of conditional clauses that are evaluated at run time. These conditions are based on either dynamic data, i.e. time
since the business process was initiated, or static data, i.e. information about the agent’s internal state. Therefore, the protocol can be used to provide effective control over the business process as well as for the formulation of personalized services. Furthermore, the actions defined in the protocol must not violate the permissions attached to a role, i.e. the protocol may not enforce the use of an operation for which the role does not have the execution permission. Contradictions must be detected during modeling.

Modeling the protocols and their dynamic behavior is the central issue in the implementation view. In order to achieve a clear separation of functional and interaction aspects we suggest to model protocols as first-class entities that encapsulate the interactions between the roles and service components. This is similar to the idea followed by the facade pattern as described in [18] where the protocols can be seen as the facades used by the roles in order to access the services. It is also similar to the concept of connectors as they are used in Catalysis [12]. In UML terms first-class entities are normally modeled as classes, hence, protocols are modeled as classes too. However, class model elements are poor to show the central characteristics of protocols – their dynamic behavior. In order to model the behavior of classes, i.e. the interaction coordination performed by the protocol, and the interactions, i.e. the operation calls, UML statechart diagrams can be used.

The statechart diagram is constructed of states which represent the situations during the protocol lifecycle. States are connected by transitions which may be triggered by events that cause a state change. Transitions from one state to another are only possible if the attached guard condition evaluates to true. The rules defined by the protocols are specified in terms of guard conditions. The allowed sequence of actions defined by the protocol is given by the sequence of states. During the transition an action may be performed or an event can be sent to another object, i.e. an operation call of a service component may be modeled. This can be used for simple forwarding of operation calls or any more complicated form of interaction. Examples for the latter would be actions that are implemented by the protocol itself, i.e. security and policy checks.

The notation of the statechart diagram is well suited to model the temporal/dynamic aspects of the behavior and it is also possible to make explicit the operation calls that occur during the protocol execution which can be used for detailed design in later stages of the development project. Though statechart diagrams are not very suitable for showing the assignment of actions to objects very clearly they do have the advantage to show parallel, iterative and selective behavior which is important for modeling protocol behavior. Since we consider protocols to be first-class objects a statechart shows the behavior of only one specific protocol object. A business process is defined by one or more protocols. In order to visualize the orchestration of several protocols into a whole business process activity diagrams may be used. For capturing a single run of such a process interaction diagrams such as sequence diagrams are useful.

![Figure 3. The protocol for acquiring a specified quality of service used by the stream provider role](image)

![Figure 4. The statechart diagram for the CP_BandwidthProtocol in figure 3](image)

Modeling only one protocol per role may possibly result in a very complex statechart diagram, though the use of sub-states may allow for structuring the model. A better approach is to allow protocols to have sub-protocols as proposed in [13]. Sub-protocols would then be coordinated by their super-protocols. In order to meaningfully decompose the protocol of a role we use the four phases of a typical market transaction as introduced by the MRM. Therefore, as a first step in structuring protocols each protocol assigned to a role would contain four sub-protocols that coordinate the interaction of the components within one specific phase of a market transaction. Each sub-protocol is modeled by a separate statechart diagram. This more modular approach allows to exchange parts of a protocol for a specific role, i.e. the protocol for the agreement phase could be changed from a Dutch auction to a reverse auction. Furthermore, patterns...
for protocols can be encapsulated in self-contained components and stored for later reuse. As functional requirements are passed through to the implementation view, a repository of previously developed protocols can be searched for in order to match the given requirements.

Referring to the StreamCom platform figure 3 illustrates the stream provider role which is associated to its protocol which has two levels of subprotocols. The subprotocol for the settlement phase (those of the other phases are not shown) has several subprotocols among which the protocol for acquiring quality of service is the one shown in the diagram (CP_BandwidthProtocol). This protocol offers a compatible set of operations to the role. The behavior of the protocol is shown by the statechart diagram in figure 4. This is very detailed as it also shows events and their signatures. Entering the protocol, the role needs to exchange keys with the service offering the quality of service support in order to authenticate and establish a secure connection. Then, the protocol allows the role to either make a new QoS request, modify an existing request or stop an already established flow. The events sent to the IQoSManager interface of the service providing component indicate the actual requests behind the operations offered by the protocol. In the given example these are simple message forwards. However, if a different service component for QoS support should be used any adaptations to a new interface may be accommodated here, transparent to the role. Furthermore, it would be possible that actions of the protocol are executed and guard conditions are specified which check for specific characteristics of the agent playing the role.

The model of the implementation view describes the procedural organization of the platform by linking the functional needs of the roles as described in the community view with the service providing interfaces of the transaction view in a specific sequence and governed by a set of rules that lead to meaningful processes. In order to establish this link, the interfaces of the services have to be known. These are modeled in the transaction view.

4.3. Modeling the Transaction View

Roles rely on services that enable their functions, i.e. paying for a product is normally supported by a payment service. We view services as being available in the form of software components that provide functionality via their exposed interfaces. These components can be distributed so that services can be consumed from any third party provider or may be obtained in the form of a software package that is deployed by the platform provider. For example, a payment service may be used by a variety of business media platforms and will usually be offered by an independent payment service provider.

This service perspective on components is not new and advocated by research [19], [20] and industry alike. Backed by the current hype of what is referred to as web services there is a chance that this form of software provision will dominate the future e-commerce landscape. An important part in modeling a service-based application involves the description of how a collection of services collaborate by making calls to each others’ interfaces. This dynamic behavior is independent from the single services and directed by the needs of the community. This issue is dealt with in the implementation view. What is left to be modeled in the transaction view are the interfaces that are provided by the services.

In the transaction view we are able to view the defined functionality of the platform in terms of a set of interfaces. These interfaces have dependencies based on the previously modeled interactions. Therefore, modeling the service interfaces is often an on-going parallel activity to interaction modeling. Furthermore, service interfaces often impose a specific invocation sequence on their interface which has to be reflected in the protocols. It is at this stage, where continuous top-down modeling is not possible anymore since existing components have to be integrated following a bottom-up approach. Modeling the services involves tasks like searching for existing services and evaluating them well before they are incorporated into the model. The search for services requires some form of directory service enabling design-time service discovery. Component catalogs or service registries, i.e. UDDI business registries [21], may be used. The evaluation involves the investigation of functional suitability as well as the evaluation of non-functional requirements such as guaranteed quality of service delivery, in case the service is provided by an external service provider.

This service perspective on components is not new and advocated by research [19], [20] and industry alike. Backed by the current hype of what is referred to as web services there is a chance that this form of software provision will dominate the future e-commerce landscape. An important part in modeling a service-based application involves the description of how a collection of services collaborate by making calls to each others’ interfaces. This dynamic behavior is independent from the single services and directed by the needs of the community. This issue is dealt with in the implementation view. What is left to be modeled in the transaction view are the interfaces that are provided by the services.

In the transaction view we are able to view the defined functionality of the platform in terms of a set of interfaces. These interfaces have dependencies based on the previously modeled interactions. Therefore, modeling the service interfaces is often an on-going parallel activity to interaction modeling. Furthermore, service interfaces often impose a specific invocation sequence on their interface which has to be reflected in the protocols. It is at this stage, where continuous top-down modeling is not possible anymore since existing components have to be integrated following a bottom-up approach. Modeling the services involves tasks like searching for existing services and evaluating them well before they are incorporated into the model. The search for services requires some form of directory service enabling design-time service discovery. Component catalogs or service registries, i.e. UDDI business registries [21], may be used. The evaluation involves the investigation of functional suitability as well as the evaluation of non-functional requirements such as guaranteed quality of service delivery, in case the service is provided by an external service provider.

Since existing services are already implemented and possibly already deployed we suggest to use UML.
component diagrams to model the services. This is in accordance with the semantics defined by the UML specification which recommends to use the component model element for modeling implemented, physical modules. Some authors prefer to use class diagrams to model components [22], [23] which may also be a valid option for the transaction view model. The interfaces of components are modeled in the “lollipop” notation. In order to model the connection between the protocols of the implementation view and the service interfaces of the transaction view the notation of dependencies are used. The high level view provided by the dependency arrows can be detailed to a level, where each action in the protocols can be traced to one specific operation in the interface it depends on. Figure 5 illustrates the component providing the quality of service support for the StreamCom platform. It consists of two interfaces of which one is used by the protocol for the bandwidth reservation as was discussed in the previous subsection. The IQoSManager interface offers all operations that are invoked by the protocol and have been modeled as sent events. The dependency between the class and the interface is shown by the dashed arrow. Although the UML 1.3 specification defines the semantics of dependencies as “indicating a semantic relationships between two model elements” [6] some modeling tools do not allow to mix class and component model elements in one diagram. In these cases representing components as classes would be a pragmatic solution.

The model of the transaction view provides the understanding of what functionality can be accessed from the protocols and is eventually provided to the roles. The service providing components can be grouped according to the phase of the market transaction in which their service is needed. However, some functionality may be needed across all phases, i.e. personalization services.

In the transaction view only the technical aspects that are directly related to the realization of the business architecture have been dealt with. Though this should be the primary focus in the early phases of a project it is also necessary to consider services that are not directly related to the business itself but do provide support concerning the technical infrastructure of the platform. These services are modeled in the infrastructure view.

### 4.4. Modeling the Infrastructure View

The infrastructure view describes the infrastructure services that are needed by the business oriented services of the transaction view. They can be distinguished from services of the transaction layer as they are not specific to any form of business architecture and they can usually not be contracted out but have to be available local to the platform. A number of basic technologies are part of almost every platform including database access middleware for persisting business objects, some form of transaction service for the secure execution of business transactions and some form of user interface library. To support a component-based approach, it is common to use some form of component infrastructure to handle all of the complex details of component coordination. Essentially, the component infrastructure provides a common set of component management services. For object-oriented distributed component-based systems these services include object persistence, object live cycle management, communication middleware, notification services, location, naming and security services, load balancing and transaction services [24]. A complete list of required infrastructure services can only be made after the technology for implementation has been selected and the infrastructure requirements of the services of the transaction view are fulfilled. Therefore, modeling the infrastructure view is omitted in the early phases and begins with the technology selection. The model of the infrastructure view can be omitted when communicating the platform model to business specialists.

Usually, almost all of the functionality needed in the infrastructure view can be obtained from existing packages accessible via a clearly defined API. In order to obtain a detailed model of these modules it is possible to use reengineering tools that automatically generate the UML class diagrams from the source code or even the binaries. The class diagrams produced by these tools can then be grouped into packages. The model of the infrastructure is therefore described by a number of packages. Import and access relationships can be visualized among the packages. Usage of infrastructure services by model elements of other layers can be shown by dependencies.

Infrastructure services can be used from model elements of any layer. For example, the business objects of the community view may need to access some form of persistence service, the processes in the implementation view may require the functions of some kind of transaction service and the services of the transaction view may require the services provided by the run-time environment of the component infrastructure.

### 5. Conclusion and Further Work

The guidelines for modeling business media platforms discussed in this paper are still subject to improvements as they are evaluated in larger scale projects such as the StreamCom project. The sound foundation provided by the MRM allows to effectively map the business architecture into a supporting platform. The views of the MRM reflect the three key elements of business architecture according to Fingar [4] - organization,
processes and technology. The latter is divided into a service view providing value added services and an infrastructure view. The modularity, which is reflected in the models of the platform, leads to a component based architecture of business media platforms that allows to flexibly incorporate new services and add functionality. The clear separation between the functional aspects of the transaction view and the interaction aspects of the implementation view provides the necessary decoupling of the business architecture from the technical service components. This is a prerequisite for enterprises that need to be able to change their service spectrum.

While the MRM provides the information of what is to be modeled in each view, UML is an expressive notation for capturing the relevant aspects. The paper presented the most suitable diagrams for the single views as part of the guidelines in order to guarantee consistent results in distributed project teams. Furthermore, the use of UML offers the advantage to be used for abstract models in the analysis phase through to the detailed models of the implementation phase using the same set of notational elements.

The presented work is related to work in the field of software architecture and design. As has been shown it borrows from Catalysis and among other similarities uses collaborations as first-class units (in the form of protocols) in order to achieve good decoupled design. However, unlike more general methods like the Rational Unified Process (RUP) or Catalysis the presented approach is focused on a specific type of system, i.e. business media platforms. Therefore, we are more specific on the required aspects to be modeled such as the phases of a market transaction, the requirement for the definition of the logical space or the need for specific entities such as contracts.

Similar to Catalysis or RUP the presented approach is architecture driven. However, different views are used which are provided by the MRM. In addition to the views phases are used to partition the generic business process supported by business media platforms – the exchange of goods and services.

Future work will concentrate on the evaluation of the applicability of the suggested guidelines in larger projects and fine tuning. Modeling several platforms will not only help to refine the modeling guidelines but hopefully also lead to the identification of reusable patterns.

In order to increase acceptability the applicability of a standard in architecture description may need to be evaluated for usability. As business media platforms are software intensive systems according to the IEEE P1471 standard this standard is a candidate to be reviewed. Since IEEE P1471 is notation neutral and does not prescribe the use of specific views or models [25] an incorporation of this standard may be feasibly without major changes. However, some changes in terminology may be necessary.

Further research will also concentrate on the definition of a repository of those patterns that identify best practice in modeling for each of the four views. Though, the concentration will be put on the community and implementation layer since the infrastructure layer is mostly given and for the services in the transaction view it can be expected that over time, more and more specialized interface standards will be developed for domains such as financial services, logistic services and so on [26]. In the long term such shared repositories of the artifacts produced by business and systems modeling can serve as a reference architecture [4]. With such a reference architecture it would then be possible to plug in new services that are available on a service market and which only need to be defined by their interfaces and their usage protocol.

Acknowledgements

The StreamCom project is funded by the Swiss National Science Foundation under project number 5003-057755. We would like to thank the anonymous reviewers for their valuable comments.

References


