Chapter 29

REFERENCE MODELS FOR ADVANCED E-SERVICES

Chris A. Vissers, Marc M. Lankhorst, Robert J. Slagter

Abstract: Reference models (RMs) capitalize on the experience that key functions and relationships determine a system's main design structure which has to be established before other design details can be settled. As such RMs can play an important role in designing complex (distributed) systems, in allocating design tasks to cooperating design teams and in facilitating their communication. These roles are also eminent in standardization. This paper discusses the need for precisely defined basic architectural concepts to construct RMs, building on experience with designing the OSI-RM. We apply these concepts in the design of a number of RMs for networked applications that provide advanced e-services.

Keywords: Reference model, architecture, basic architectural concept, service, protocol, networked application.

1. INTRODUCTION

Suppose you contract an architect to design your house. You discuss the size of the house and how it is situated on the premises. How rooms, corridors, staircases, windows, bathrooms, balconies, doors, a roof, etc. will be measured and put together. You agree on a master plan on the basis of which the architect will produce detailed specifications.

How come that you communicate so efficiently about that master plan? We suggest it is because you share a common frame of reference: you both know the concepts of premises, situation, measure, room, balcony, staircase, etc. You know their functions and their possible relations. Mentally, you both use a *reference model* of a house. This reference model (RM) defines a structure of major functions and how they are related while applying

commonly known concepts. It provides an abstract design, ignoring many details that will be filled in later such as construction details, materials to be used and colors to be applied.

Likewise, the design of information systems will be more effective and efficient if a suitable RM as a blueprint for information systems is taken as a starting point. Suppose you want to design a system to support electronic transactions. Then you have to answer questions such as: What services may users expect from the system? What should be the building blocks of the system? How should they interact? It would then be quite helpful if you also have a RM that shows building blocks for managing transactions, for providing financial and logistic services, etc., and that shows what information these blocks exchange.

We introduce RMs for a broad range of networked applications: structures of functional entities, implemented by software, that offer *services* to distributed users while exchanging information via a network. Examples are applications that enable users to trade by exchanging orders, bills, and payments via an underlying network, or applications that allow users to simultaneously access design drawings despite the fact that they are miles apart, or applications that allow users to search through remote video libraries and download a video. The majority of such services fall in one, or are a combination of the following categories:

- collaborative services;
- transaction services;
- content services.

Networked applications can become quite complex if you consider all the details of functional definitions, message formats, programming schemas, programming code, operating system calls, etc. RMs help you to master this complexity by focusing first on the high level system design, suppressing detailed design issues until this high level design has been settled. In fact you follow the strategy of the architect of your house.

2. REFERENCE MODELS, THEIR NATURE AND PURPOSE

A reference model provides people with a common reference to an object, e.g. a distributed system, as a basis for their common understanding, discussion, and further action. They are about models in the sense that they describe essential aspects of systems while abstracting from details not considered essential for the pursued goal. Typically the goal is to focus on what systems should do, rather than how they can be constructed and operate at the implementation level. We define a RM as a structure, or organization,

of related functional entities that defines only globally the key functions and key relationships of these entities. This means that RMs are incomplete system designs in two respects:

1. The high abstraction level implies that functions and relationships are defined only functionally and as implementation independently as possible, leaving freedom for the individual manufacturer to choose his own implementation strategy.
2. Key functions and relationships are defined only globally, leaving freedom for design teams to complete the functional design by adding design details that at the RM level are not considered key.

The latter requires some extra explanation since the term "key" is usually intuitively, rather then explicitly applied. By a *key* we mean a characteristic that largely determines the function of an entity, system or relation and consequently has a large impact on the structure and further design of a system. It means that extending the RM to a complete design by refining the key functions can be done while preserving the structure and relationships (interfaces) of these functions.

> For example, when it is key that a service is connection oriented, it will appear that this key requirement largely determines the nature of the service and the protocol that implements it. The same applies when it is key that a service is reliable. At the protocol level it will appear that protocol data unit (PDU) numbering is a derived key functional element to achieve both connection orientation and reliability which completely dominates the structure of the protocol.

The development of RMs is a design activity that should follow qualitative design principles such as: do not link what is independent (orthogonality), introduce functions in their most general form and avoid slightly diverging alternatives (generality), do not introduce what is immaterial (parsimony), and do not restrict what is inherent (propriety).

A RM serves several purposes, such as to act as a basis for:

- understanding the essential user requirements and derived (key) properties of the real world system (a networked application in our case) that has to be developed;
- formulating these properties in an initial high level design and preserving the conformance between the service provided and the protocol implementing this service;
- communication between the users of the real world system and the RM architect to communicate requirements and document their agreements;
- communication between the RM architect and the designers of systems;
- communication between different design teams that may work together on a system design.

There exists an extensive literature about RMs, reference architectures, and design patterns, their purpose, their form, the language in which they should be expressed, etc. (see e.g. [2] , [7], [16], [17]). A well-known

example is the RM for Open Systems Interconnection (OSI-RM) [11] which globally defines the services and protocols of networks, and applications on top of these networks *(Figure 1).*

Other well-established examples are the RM for Open Distributed Processing (RM-ODP) [13], the Workflow RM of the Workflow Management Coalition (WfMC) [9], and the Object Management Group's Object Management Architecture RM (OMA-RM) [14].

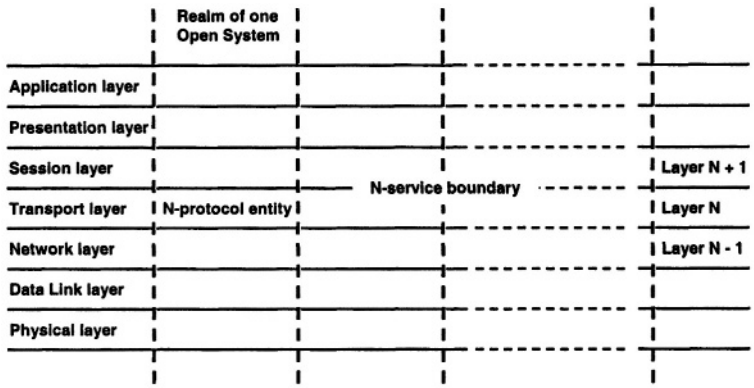

Figure 1. Original illustration of the RM for Open Systems Interconnection.

3. BASIC ARCHITECTURAL CONCEPTS AGAINST A HISTORICAL BACKGROUND

To serve its purpose as a common reference, a RM must be "constructed from" sound abstract and basic architectural modeling. This implies that such concepts should model properties considered essential for real world systems, that are precisely defined at the appropriate abstraction level, and that can effectively be applied as basic building blocks to construct a high level design. They should also be well understood and commonly supported. Here is the "Achilles heel" of many RMs. Experience with the definition of the OSI-RM, and its related services and protocols, has learned that whole crowds of experienced designers were perfectly willing to believe in the most bizarre and absurdly defined concepts as soon as some form of abstraction comes into play. Correspondingly it is astonishing to observe what confusion can be introduced in block diagrams while people are believing that the mere use of it would ensure clarity and precision. Since sound basic architectural concepts, apparently, are not so easily established, we define below some architectural concepts that we will use for our RMs

for networked applications. To underline that their definition is not so trivial we contrast them with some bizarre interpretations that have circulated in the OSI world.

In this section we use an intuitive graphical notation; in the sections describing our reference models we use a more formal notation.

3.1 Service

We define a *service* as the (possible) behavior of a system as it can be observed and experienced by its users. The service concept is of prime importance since it defines precisely what benefit a system provides to its users. Actually it defines a system's "reason d'être".

> In the ISO-OSI standards meeting in November 1981 in Berlin there was a big fight between those who wanted to have separate service standards and those who rejected this idea and only wanted to consider a service definition as an informative addendum to a protocol standard. The argument was that OSI aimed only at systems interconnection and that only interconnection needed to be defined in a conformance testable way (by monitoring the PDU exchanges). Service standards were considered not to be conformance testable since they focus on user interaction (which apparently was considered irrelevant) and would require testing the common behavior of products of different manufacturers. The argument, of course, was nonsense, since one (i.e. ISO itself) can proof the conformance of the protocol entity specification against the service specification and then conformance test the protocol entity implementation against the protocol entity specification. Fortunately the advocates of service standards won the battle, however at a certain price: the service standard could not be published independently from the protocol standard.

A service defines a system as a black box, i.e. it provides the most simple but complete definition of the observable behavior while obscuring how the system is internally constructed. See *Figure 1.*

> In the early OSI documents a service was defined as "the functions of a layer, while using the functions of all layers below. See *Figure.* Since the functions in an N-layer are the N-protocol entities, this definition in fact confronts the users with the whole complexity of all protocol entities below an N-service boundary. It took until the end of 1982 before this definition was revised, following the work of the Formal Description Techniques (FDT) Group.

3.2 Interaction Point or Service Access Point

In order to make use of a service, users have to interact with it. The fact that a user can interact with a service can be formalized by the *interaction point* (IP) concept, in OSI called the *service access point* (SAP) concept. The formal semantics of an IP is basically the identification of the functional entities that can interact. For an IP these entities are the user and the service (provider). See *Figure 1.*

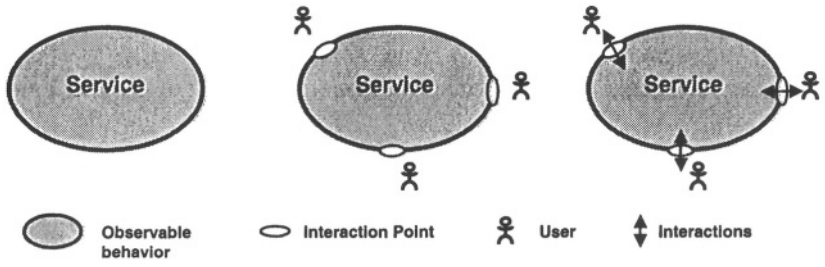

Figure 1. Service as a black box, service with IPs and users,service with IPs and user/service interactions.

Associated with IPs are the concepts of *addresses* and *names.* Generally an address is considered as a physical or logical location, while a name is considered as the identification of a specific user, the location of which is left undefined.

3.3 Interaction Primitive or Service Primitive

The interaction of a user with a service is always in terms of one or more "units of interaction", called interaction primitives, or simply *interactions* (Is). See *Figure 1.* In OSI they are called *service primitives* (SPs). The semantics of an I is basically a unit of common activity, which in principle is the same as a unit of common behavior, at an IP and resulting (with a certain probability) in some data at some moment. Interacting parties may provide different contributions to the I, and may have different use of and constraints on the results. At the appropriate abstraction level only the existence of the I and its attributes (the IP, the contributions, the results, the probability and the time moment) need to be defined.

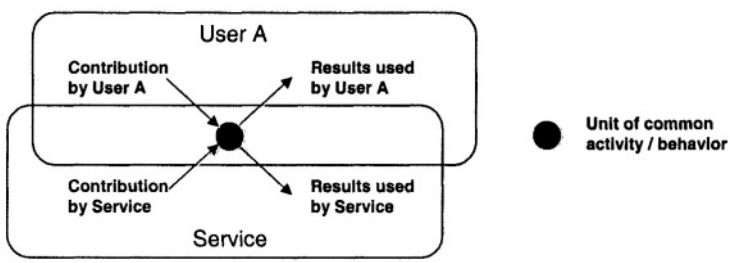

Figure 2. An interaction as an unit of common behavior of two entities providing different contributions and using different results.

Figure 2 shows a graphical way of illustrating a common activity, where a surface indicates activity and a surface overlap indicates common activity. The I can be defined by parameters and operations on them.

During the design and documentation of the OSI Transport Service, the design team (of which the first author was a member) did not want to view the abstract notion that a service primitive can be considered as a unit of common activity of two adjacent protocol entities. Instead, service primitives were only understood as one-directional message transfers, an interpretation many designers still have. Considering the establishment of a SP as a unit of common activity would have allowed to define the connection endpoint identifiers (CEIs) as a parameter of the SP whose establishment can be left to (protocol) implementation. By not appreciating this advanced design concept, CEIs were left out of the definition of SPs resulting in a politically agreed remark in the text of the standard [12].

3.4 Abstract Interface

An *abstract interface* (AI) defines the possible Is at one interaction point and their causal and parameter relationships. Whereas the IP is an abstract (logical) location, the AI defines the common behavior at that location.

There is quite some confusion between the notions of an AI and a *real interface* (RI) where many designers seem to only understand the latter. Where the AI is at the top architectural level, such as at the RM level, a RI is always at the level of a real implementation. In fact the same RI may be used in the implementation of a variety of AIs. The notion of AI never played a dominant role in OSI.

3.5 Service Design

A service can be designed by defining the IPs, the possible Is at the IPs, and all their causal and parameter relationships. Alternatively this can be designed by first defining the AIs at different IPs and then adding the causal and parameter relationships between the Is at the different AIs. The latter makes use of a constraint-oriented specification style [22], in which local and remote constraints on the service's behavior are separated. This helps in understanding complex systems, in deriving protocol entities from a service, and in proving their conformance.

In order to show the causal relationships of interactions, time sequence diagrams are often used. Although quite illustrative for simple situations, the two dimensional drawing scheme is not suitable for more complex relationships. For that purpose we better take resort to an architectural specification language such as AMBER [5].

Continuing the discussion under 3.1: In OSI one did not want to speak of Service Specifications in line with Protocol Specifications but only of Service Definitions. The reason to speak about Definitions was that they were not supposed to be implemented. So they were not specifications meant as prescriptions for implementation. This again was a

misunderstanding since a service is implemented by implementing the protocol. A Service is just one step in abstraction level (i.e. formulating the essential!) closer to the user. In fact also an OSI Protocol Specification is not implementable directly, since only the OSI Protocol Entity is, and its specification has to be derived first from the protocol standard by the designer (see also the discussion in Section 3.6).

OSI service standards were defined by English text and illustrated by block diagrams and time sequence diagrams. An example of the latter is shown in *Figure 3a.* To draw such diagrams, so called OSI Service Conventions were prescribed according to *Figure 3b.* Originally these conventions prescribed straight angled lines, as shown in *Figure 3c,* where even the magnitude of the angle had the meaning to indicate "transfer speed". This idea was probably inspired by the advantageous use of rulers to draw such lines. Its absurdness could only, after heavy debates, be convincingly demonstrated by showing the effect of Expedited Data which can overhaul Normal Data. While the conventions would prescribe a line with a larger positive angle for higher speed data transfer it actually would result in a line with even a negative angle.

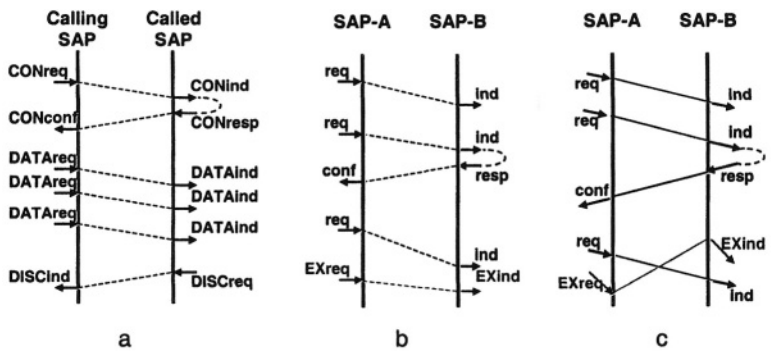

Figure 3. **The OSI connection oriented service (a), OSI service conventions (b) and its early predecessor (c).**

3.6 Service as a composition of Protocol Entities

In OSI it was said that a protocol renders a service. If so, then there should be a direct relation between a service, and the protocol that implements it. This can be easily obtained by defining that the composition of the protocol entities renders the service. This is illustrated in *Figure 4.*

From the above it follows that the IPs of the protocol that are accessible by the users are the same as the IPs (SAPs) of the service, which means that the protocol entities that have these IPs (SAPs) are also involved in the execution of Is at these IPs (SPs at these SAPs). Formally one can say that the behavior of the composition of protocol entities, as observable at the IPs, should conform to the service.

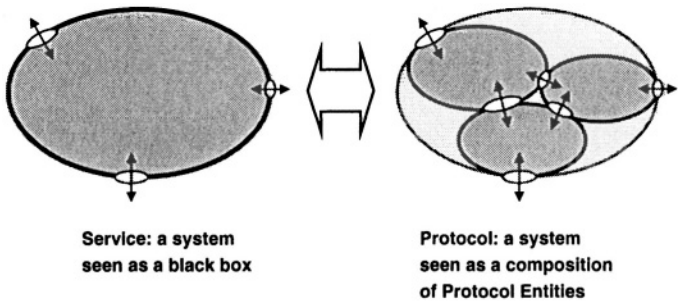

Figure 4. **A service as a composition of a(n arbitrary) structure of protocol entities.**

Consistent with this approach, an OSI (N)-protocol should be defined as the composition of the (N)-peer protocol entities in the (N)-layer and the underlying (N-1)-service of that layer, where the latter can be seen as a particular entity (protocol entity according to *Figure 4*), functionally quite different from the peer (N)-protocol entities and interconnected to them via (N-1)-SAPs. See also *Figure 5.* In OSI, however, the attitude was as if the (N)-peer protocol could be defined rather independent of the underlying (N-1)-service. This was also reflected in the task assignments to the various subcommittees: the Session Group did notdefine the Session Protocol and the Transfer Service, given the Session Service, but defined the Session Service and the Session Protocol given the OSI-RM.

Pre-occupied with the focus on the interconnection of open systems, OSI defined a protocol only as the relationship between peer-to-peer protocol entities (i.e. entities in the same layer). This led to the funny consequence that OSI protocol specifications never defined the explicit relation between SPs and PDUs. This in spite of the fact that concepts such as segmentation and reassembly, concatenation and separation, blocking and de-blocking, splitting and recombination, and multiplexing and de-multiplexing were defined in principle. Consequently, a conformance relationship between a service standard and a protocol standard rendering this service could never been proved. Certain OSI officials were even quite surprised that the FDT group wanted to formally specify both the (N)-service and the (N)-protocol and proof their conformance, questioning why one should make two different specifications of the same standard.

Saying that a service is a composition of protocol entities one could easily be tempted to also say that a protocol is a decomposition of a service. And frankly this use of words often occurs. However, the reader should be warned that a protocol *cannot* be derived by simply decomposing a service. The basic reason for this is that in a decomposition internal structure is revealed that requires additional design choices. These choices are also incurred by implementation concerns. Consequently a protocol is generally much more complex than a service. We will not further elaborate on this since it is beyond the scope of this paper.

3.7 Layered Protocols

The notion of protocol as a decomposition of a service, as introduced above, allows to define a protocol with an arbitrary structure. I.e. a structure with an arbitrary, problem dominated composition of protocol entities like we do hereafter in sections 5, 6 and 7. Frequently, like in the OSI-RM, we find layered protocol structures. As mentioned in our example above we achieve this by defining an (N)-protocol as the composition of the peer (N)-protocol entities and an underlying (N-1)-service, where the latter can be understood as an (N)-protocol entity with a specific nature. This allows to define the (N-1)-service again as a composition of peer (N-1)-protocol entities and an underlying (N-2)-service, as shown in *Figure 5.*

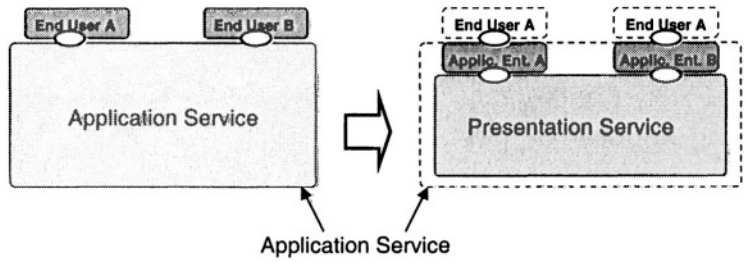

Figure 5. The OSI-RM as nested services interconnected by layers of protocol entities.

In so doing, a layered structure results that consists of a set of *nested* services interconnected by layers of protocol entities. In fact, the OSI-RM consists of 8 (not 7) nested services, from top to bottom starting with the Applications Service and ending with the Medium Service, where the latter is not further decomposed.

3.8 A Protocol Entity Considered as a Service

Considering a service as a composition of protocol entities poses the question how such entities should be specified. The answer is simply by specifying them in the same way as a service is specified. This view allows one to consider a protocol entity as a service in its own right, however at a lower abstraction level, implying that a protocol entity can again be considered as a composition of lower level (protocol) entities. This view is consistent with the view to consider an (N-1)-service as a composition of (N-1)-protocol entities and a (N-2)-service.

This in fact gives us the basis of a top-down design methodology. It proves the importance of the service concept as a black box that can also be seen as a composition of lower level (protocol, or if you wish service) entities.

4. MODELING E-SERVICES

Networked applications are top-level protocol entities that directly interact with end users, corresponding to the OSI-RM application layer entities. They provide application services, often called *e-services,* while using network services. Inside application layer entities we do not necessarily have again a layered (sub)structure, but generally have a specific structure determined by the nature of the service to be provided. Some networked applications, for example, exhibit a centralized control structure as shown in the collaborative services of Section 5. Others may be structured according to phases in a process as is the case for the transaction services described in Section 6, or have a pipeline structure like the content services of Section 7. We illustrate the quality of our decompositions with examples that show interface-preserving refinements of some of these key functions.

Table 1 summarizes some characteristics of collaboration, transaction, and content services, the basic service categories we discuss. It shows that these categories are quite distinct.

Table 1. E-service characteristics.

	Examples	Data volumes	Procedures	Media
Collaborative services	Video conferencing Shared whiteboard	medium – high	Some control	Multi
Transaction services	Auction Workflow Management	low	Rigid	Single
Content services	Digital video library	high	Loose	Multi

The modeling approach we take evolved from the architectural concepts we described in the previous sections. The language AMBER [5] was designed as a specification language for business processes, with strong roots in design and description techniques for telematics services and protocols. The RSD modeling language [19] augments AMBER with concepts for describing networked enterprises. In the following sections, we use (self evident parts of) RSD to describe our models. These languages themselves cannot be discussed as they are beyond the scope of this paper.

In the design of networked applications the people and organizations as users of these applications play important roles. In the modeling process we therefore start with identifying the necessary roles and next derive the associated key functions from these roles. Our RMs, therefore, consist of two types of sub-models.

The first sub-model is the *role model (Figure 6)* that defines which roles are involved in delivering and using the service defined by the RM. A role can be considered an abstract carrier of behavior (like end users, or service provider in the OSI model) and is denoted by an octagon. Arrows between

roles represent the service delivery or the flow of information, goods or money.

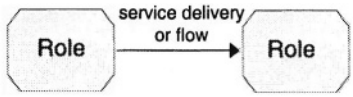

Figure 6. Role model.

The second sub-model is the *function model (Figure 7)* that defines the functions that together realize the behavior of a service. The responsibility for performing a function is associated with one or more roles. An actor that fulfils a role must carry out the associated functions or delegate this to another actor.

A function can be further detailed as a composition of sub-functions. Arrows between functions denote again the flow of information, goods or money. Flows may split and join, depicted by diamonds and squares, respectively; they enter and exit functions via triangular input and output interfaces. The thickness of the arrows is used to distinguish between different types of flows. Primary flows, e.g. the content delivered by a Service, may be depicted by a thick arrow; secondary flows, e.g. control flows, may be denoted by a thin arrow.

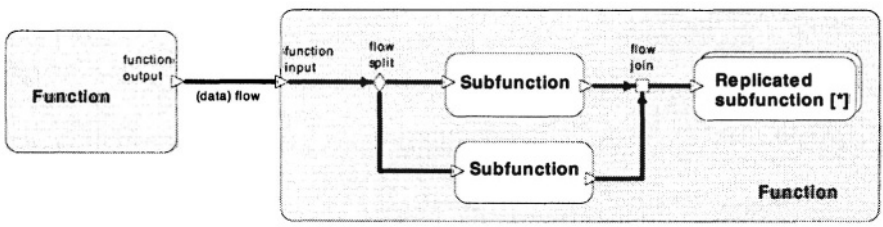

Figure 7. Function model.

Figure 8 shows how functions and flows correspond to protocol entities and interactions discussed in Section 3. Interactions may be refined to one or more flows that indicate the direction of the exchange of information, goods or money. The interaction between PE1 and PE2 in *Figure 8,* for example, is split into two flows between Function 1 and Function 2. Flows may be augmented with text.

The RMs described in the sections 5, 6 and 7 are described in much more detail in [3]. Here, we zoom in on parts of the RMs that are illustrative for our approach.

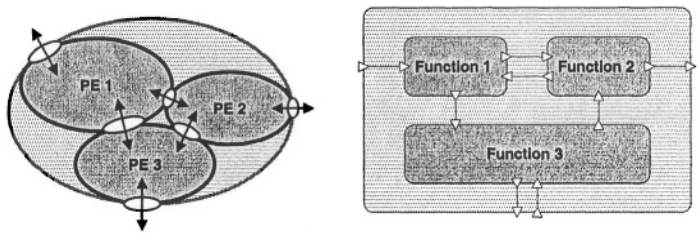

Figure 8. **Protocol Entities and Interactions versus Functions and Flows,**

5. COLLABORATIVE SERVICES

Collaborative services are designed to support groups of interacting people in their cooperative tasks. Examples of such cooperative settings are teachers tele-educating students, physicians tele-consulting each other, tele-conferencing board members, and engineers collaboratively working on designs. Collaborative services typically allow people to invite others to a virtual meeting, to communicate, share documents, share agendas, divide and together carry out work, etc. In this paper we use the term *conference* as a unit of abstraction, denoting a group of cooperating people that complete a common task while supported by collaborative services.

The prime use of collaborative services is to bridge distance and time between geographically dispersed collaborating people by allowing them to communicate and access data at different moments in time. Ensuring that users have consistent views of their collaboration and shared information is an important aspect of this service. A shared whiteboard service, for example, must provide mechanisms to ensure that all users can view the same information, can see all updates, and can know who may change the information. Typically, collaborative services exert a modest control over the users in order to organize their cooperation. One user may play the role of chairman and employ procedures to give the floor to others, interrupt them, etc.

5.1 A Reference Model for Collaborative Services

In a collaborative setting each user (participant in a conference) has access to collaboration support functions. These functions use a network for interconnection. See *Figure 9.*

The clustering of collaborative support functions into functional building blocks in our RM is based on the separation of concerns principle. Literature on how people cooperate (e.g., [1, 6, 8]) indicates that cooperative settings

differ (in abstract terms) regarding four aspects: 1) the set of people that cooperate, 2) the set of tools they use to communicate and access shared information objects, 3) the set of rules they apply, and 4) the mechanism they apply to start cooperation (denoted as conference enabling). On the other hand, functions to start and end cooperation and to manage the set of cooperating people are always present.

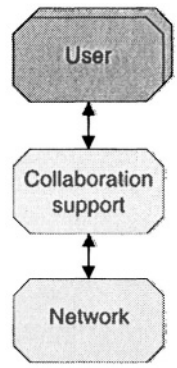

Figure 9. **Roles in a collaborative setting.**

Based on this we define the following key functions, as indicated in *Figure 10*: conference tools, to communicate and access shared information; coordination, to define and enact rules; conference enabling, to bring people together for cooperation; and conference management, to provide start and stop conferences and to manage the set of cooperating people. Conference management also keeps track of the conference tools that are in use and specifies what coordination policy applies.

Conference tools allow participants in a conference to communicate or collaborate using shared information objects. An audio conferencing tool and a shared whiteboard are examples of conference tools. Coordination defines and enacts the rules that may apply during a conference: the access rights for using collaborative services. The main function of Conference Enabling is to bring people together for cooperation, by providing awareness about other users who can be invited to a conference, or about other ongoing conferences that can be joined.

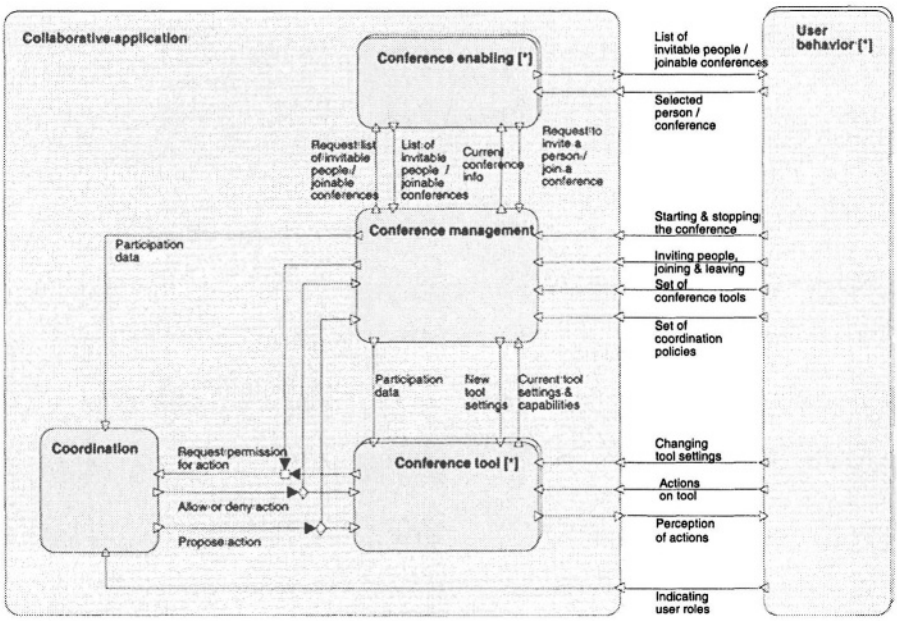

Figure 10. **Key functions realizing a collaborative service.**

5.2 Conference Management

As an example of interface preserving decomposition, we decompose the Conference Management function *(Figure 10)* that is responsible for providing services to users to manage an online conference. For a complete description we refer to [3] and [18]. These services allow users to:

- start and end conferences;
- join and leave conferences;
- invite people to a conference;
- select the tools to use in a conference;
- select the coordination policy that applies during the conference.

The specification of conference management sub-functions in *Figure 11* adds detail (i.e. lower level functional entities) to the high-level description in *Figure 10.* It shows how the specified interactions are provided by sub-functions. For instance, users can interact with conference management to indicate which tools should be active in a conference. Subsequently, conference management can adjust the conference tools settings, taking into account the tool capabilities and current tool settings.

The conference management function is furthermore responsible for providing information regarding the conference and its participants to other collaborative functions.

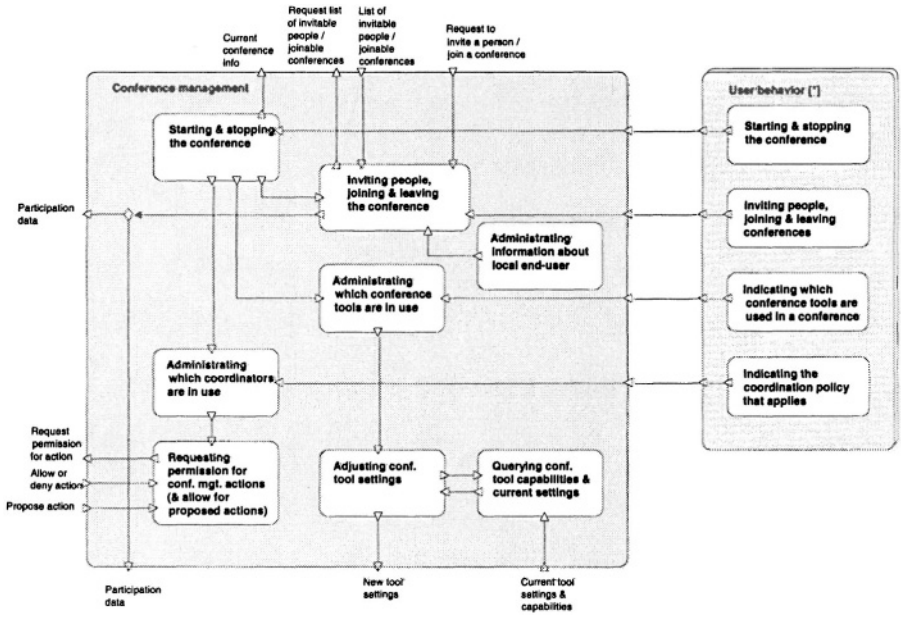

Figure 11. Conference Management sub-functions.

6. TRANSACTION SERVICES

Transaction services support formal, and often legally traceable, transactions between parties by organizing the exchange of pre-defined messages in pre-defined orders. Their prime use is to exert legal, commercial and financial commitments, and parties may be held liable when they fail to follow them. A procurement may be organized, for example, as a request for a product, leading to the indication of a price and followed by the acceptance or rejection of the offer. Companies trade according to predefined procedures such as: “you pay first, then I deliver”, or “I deliver, you pay in two installments”, or “you order, I ship unless you cancel”, etc. The procedures may be dictated by law or may have been agreed before in negotiations, etc.

Typically, transaction services do not convey massive volumes of data per transaction; rather they exchange simple messages like orders, reservations, bills, payments, etc.

Commercial transactions occur over and over in the course of a business day, and many different types of, so called, e-commerce services have emerged recently including online marketing, searching, ordering, payment, and after-sales support [20, 21]. Here, we will focus on e-commerce that takes place around *electronic marketplaces* in a business-to-business setting.

6.1 A Reference Model for E-Marketplaces

Figure 12 shows the basic roles involved in an e-marketplace and their relations [10]. The primary user roles involved in any electronic trading system are those of buyer and seller. Another essential role is the one of marketplace operator as a front-end service provider. The latter is responsible for offering a large spectrum of services to the buyer and seller. All of these services are meant to support the fulfillment of the distinct phases of a business transaction process.

Almost always, the provisioning of some parts of these services is delegated or subcontracted to back-end service providers (e.g. providers of logistics services, financial services, foreign trade services, communication services, trust services). The services can be delivered either directly to the users (like in the case of logistic services), or through the front-end service provider, e.g. the marketplace operator (as in the case of trust services). Of course each marketplace has its own particular role, flow structure, and degree of complexity, and therefore various scenarios can be imagined to describe a commerce system.

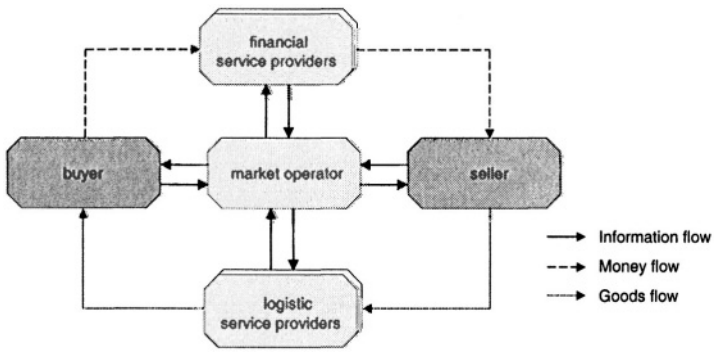

Figure 12. **Roles involved in an electronic marketplace.**

Let us now zoom in on the services provided by the market operator. The central organizing principle of our RM (see *Figure 13*) is to group those functions that occur within the same time frame and need the same information. Any commercial transaction has three main phases: information, negotiation and settlement. We therefore define three key functions of the marketplace: Information Management, Negotiation and Agreement Management, and Settlement and Fulfillment Management. Note that we distinguish between the functionality of the two user roles: buyers and sellers.

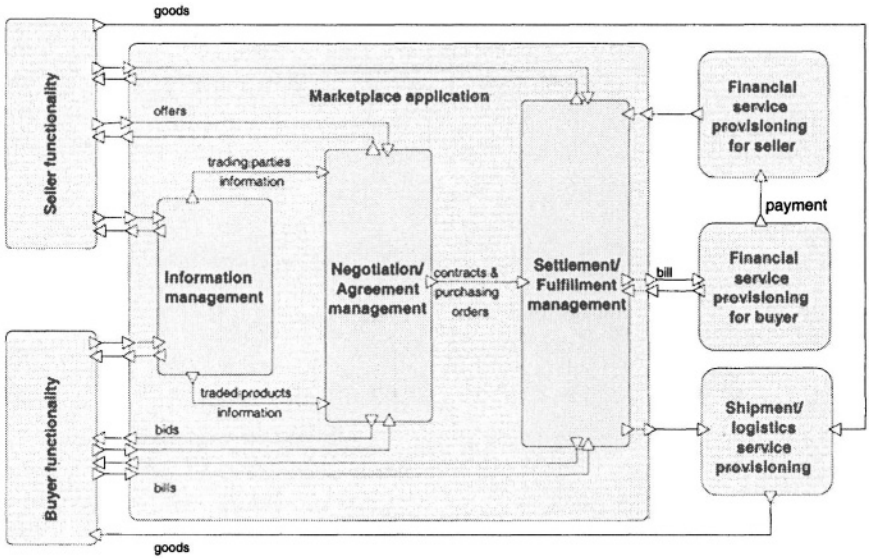

Figure 13. Key functions in an e-marketplace.

Information Management deals with the acquisition, storage, maintenance, provision, and presentation of information to buyers and sellers. This function covers information areas such as general user information and profiling, user offerings and user demands information, administration and maintenance of electronic catalogues, search engines in catalogues, marketing & advertising. This function is also responsible for the provision of information regarding the selected products and the trading parties to the function Negotiation and Agreement Management.

Once a user has decided to perform a transaction, the Negotiation and Agreement Management function takes care of the negotiation process, provides support for pricing mechanisms (auctioning, bidding, bartering, exchange, etc.), and for the issuing and distribution of electronic contracts and purchasing orders to the Settlement and Fulfillment Management function.

The latter operates whenever an agreement over a commercial transaction between two (or several) parties has been reached and has resulted in an electronic contract or order. This function provides support for the transfer of goods and money between the trading parties, and thus triggers the financial and logistic completion of the transaction. Some of the tasks that are covered by this function are invoicing, billing, and payment, tracing, and tracking orders, coupling to the back-office (including legacy systems), shipment facilitation or logistic services, etc.

6.2 Negotiation and Agreement Management

As an example of interface preserving decomposition, we now zoom in on the Negotiation and Agreement Management function. Again, we use functional decomposition according to the phases in the transaction as the guiding principle in the RM. First, the seller and the buyer engage in a negotiation process in the marketplace. This negotiation follows certain procedures through which the price and other conditions are settled. Second, once the price has been set, the negotiation ends, resulting in an agreement. This also entails issuing a contract (or the purchasing order) describing the terms of the transaction (products, amounts, prices, trading parties, guaranties, penalties, etc.). In our RM, this commercial functionality is embedded in the function Negotiation and Agreement Management (see *Figure 14*). This consists of a number of sub-functions.

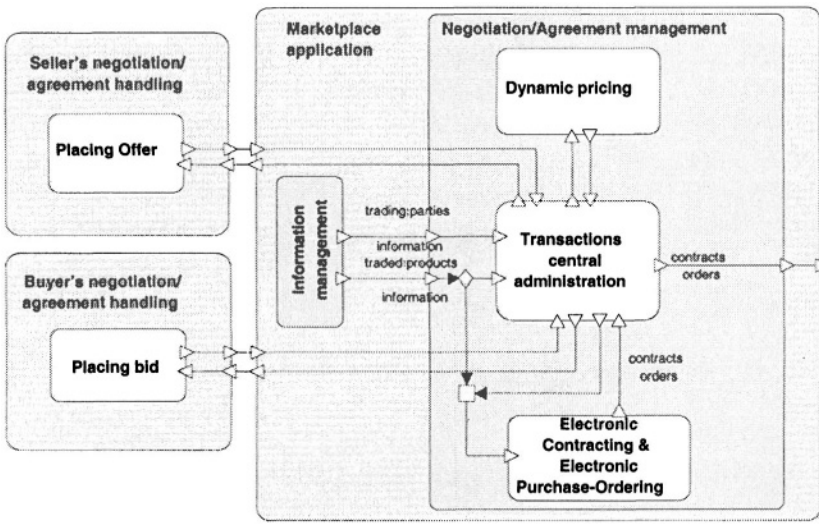

Figure 14. **Negotiation and Agreement Management sub-functions.**

The Transactions Central Administration is responsible for the information regarding the negotiation procedure, for controlling this procedure, and for commanding, in case of agreement, the issuing of an electronic contract (or purchase order). The inputs of this function are the information about the trading parties and traded products received from the information management functions, and, in case of dynamic pricing (such as an auction) the bids and offers received from the Placing Bid function and Placing Offer function of the buyer(s), and of the seller(s) respectively. In

the case of dynamic pricing, these inputs are forwarded to the Dynamic Pricing function where they are processed according to the rules of the pricing algorithm. The results are sent back to the trading parties, through the Transactions Central Administration function, and, eventually, a new iteration can start.

When the price has been set, the Transaction Central Administration provides all the necessary information to the Electronic Contracting & Electronic Purchase-Ordering function. This initiates the issuing of an electronic contract (or purchase order). This function is also responsible for the storage of these documents, which is in fact the most important output of the transaction central administration, and of the negotiation/agreement management itself. They will be forwarded to the next main function of the marketplace, the Settlement and Fulfillment Management.

7. CONTENT SERVICES

Content services allow people to access and manipulate electronic content such as: accessing digital video libraries, viewing video programs, mixing parts of different video programs to compose a new program. The primary use of content services is to give users access to large amounts of data. Here, we will mostly discuss digital video disclosure because video is in some respects the most demanding form of content manipulation. To disclose video, we need a set of services for content production, a web enabled distribution channel and a method to consume it. The particular focus will be on video handling, data-interoperability, storage of high volume content, semi-structured metadata and structured data for services like metering, accounting, billing and payment ([15, pp. VI and 9]).

The main engineering problem of a multimedia system is to deal with the fundamentally analogue nature of real sounds and images, and to extract the meaningful information from the theoretically infinite amount of information they contain. An important practical consideration is that for this reason multimedia tends to be expensive. One has to process large amounts of data while keeping track of financially important information like usage. In content engineering, properties of data and the way data is represented and stored are important architectural issues.

7.1 A Reference Model for Content Services

In the content supply chain from media producer to digital content consumer, we identify four different roles to be played as shown in *Figure 15.*

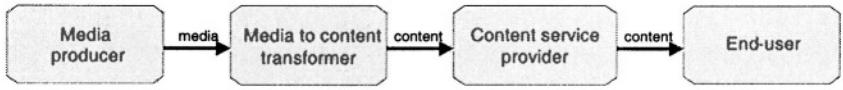

Figure 15. **Roles in the content supply chain.**

These roles perform consecutive steps in the 'content pipeline', from the production to the consumption of the content. This, therefore, determines the main structure of our RM. We decompose the content supply chain along the same boundaries as in *Figure 15,* resulting in the key functions of *Figure 16.*

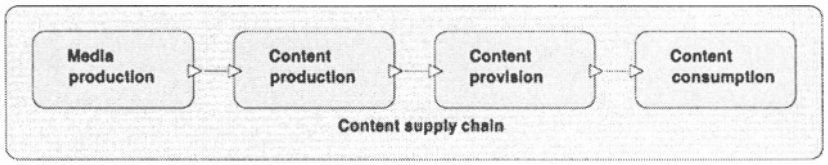

Figure 16. **Key functions of the content supply chain.**

The function Media Production is responsible for producing media assets (movies, music, pictures, texts, etc.), and metadata for these assets (e.g. bibliographical data, background data).

Content Production transforms media assets and associated metadata into digital content, by digitizing the media (if not digitally produced), and/or transcoding it into various digital formats. Additionally, it adds metadata for easy searching, data management, and business purposes, metadata and watermarks for digital rights management, and web pages, links to other pages and assets, advertisements etc.

Content Provision makes content available to end-users. It enforces digital rights, obtains payment from end-users, and pays media producers for the use of their content. Finally, Content Consumption provides functionality for searching and retrieving of content, paying for content, authentication for digital rights management, and possibly enjoying the content.

7.2 Content Production

As an example of interface preserving decomposition, we further refine the Content Production function. The full RM can be found in [15]. Content production is typically done by service industries for both the media industry and for content providers. The organizations involved are the organizations that provide the technical infrastructure for media production (which may or may not be owned by that media producer). The boundaries between media

production and content production are not entirely clear cut, and the boundaries may blur even further if digital media become the default.

In decomposing the Content Production function, we use another important organizing principle in RMs: the distinction between primary functions, i.e., dealing with the content itself, and secondary functions, which control and monitor these primary functions. This leads to the decomposition shown in *Figure 17.*

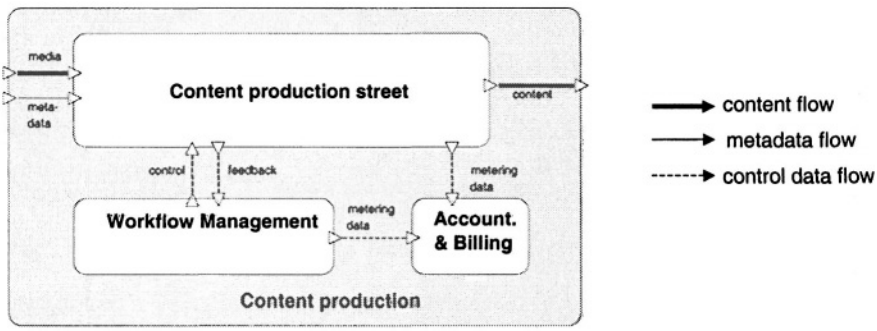

Figure 17. Content production sub-functions.

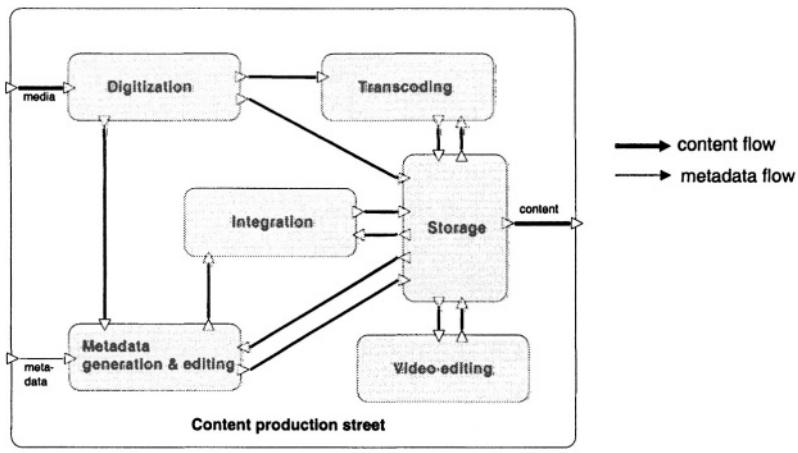

Figure 18. An example content production street.

The Content Production Street is the "factory" in which analog or digital media assets and the corresponding metadata are transformed into content. The Workflow Management function does job scheduling, job monitoring and resource allocation inside the production street. The Accounting and Billing function keeps track of resource consumption. The core functions of the content production process are contained in the example decomposition

of the Content Production Street, shown in *Figure 18*. The sub-structure of this function is very specific to the content production domain, but its general organizing principle is that of a data-centric architecture, which is very well suited to a data-intensive application such as content production.

8. CONCLUSIONS

We have taken collaborative, transaction, and content services as categories of basic e-services from which, in principle, more complex e-services can be composed. By focusing on the most important functional aspects of a service we have shown that it is possible to define a high level, main design structure for the networked application as a composition of key functions and their relationships that supports this service and that can act as a RM. The structure of this composition is very much determined by the nature of the service to be provided. This main design structure has to be established before other design details can be added. The architectural quality of the compositions has been illustrated by showing interface preserving decompositions of some key functions as examples of adding design details.

The approach shows that RMs can play a significant role in designing complex systems, in allocating design tasks to cooperating design teams and in facilitating their communication. As such, RMs can also play a vital role in standardization activities.

To serve its purpose as a common reference, we think it is of great importance that a RM and its derived functions is built on sound architectural modeling concepts that are not only generic, complete, and consistent but also realistic abstractions of real world system properties, are well understood and commonly supported. In that respect we strongly support Brooks' statement, "conceptual integrity is the most important consideration in system design" [4]. Therefore we have outlined some concepts we consider essential for architecting RMs and contrasted them against some historical misjudgements.

Since e-services that are distributed across the Internet are relatively advanced developments, the body of knowledge regarding their architecture and design is still very much under development. This paper aims to contribute to these developments.

ACKNOWLEDGMENTS

Much of the work on basic architectural concepts, to which this paper refers, has been done at the University of Twente. We would like to acknowledge Marten van Sinderen, Luis Ferreira Pires, and Dick Quartel for their important contributions to this work. The material on the example reference models originates from the GigaPort program (http://www.gigaport.nl), a large Dutch initiative on future Internet applications and networks, funded by the Dutch Ministry of Economic Affairs. We would like to acknowledge Frank Biemans, Rogier Brussee, Maria-Eugenia Iacob, Paul Porskamp, and Henk Jonkers for their contribution to these reference models.

REFERENCES

1. Ackerman, M. (2000). The Intellectual Challenge of CSCW: The Gap Between Social Requirements and Technical Feasibility. *Human-Computer Interaction,* Vol. 15 (2-3), pp. 181-205.
2. Bass, L., P. Clements, and R. Kazman (1998). *Software Architecture in Practice.* SEI Series in Software Engineering, Addison Wesley Longman, Reading, Massachusetts.
3. Biemans, F.P.M., R. Brussee, M.E. Iacob, H. Jonkers, M.M. Lankhorst, P.A.P. Porskamp, and R.J. Slagter (2002). *Reference Models for Networked Applications: Blueprints for systems that support commerce, collaboration and content deployment via the Internet.* TI/RS/2001/069, Telematica Instituut, Enschede, The Netherlands.
4. Brooks, F.P. (1995). *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley, Reading, Massachusetts.
5. Eertink, H., W.P.M. Janssen, P.H.W.M. Oude Luttighuis, W.B. Teeuw, and C.A. Vissers (1999). A Business Process Design Language. In: Wing, J. M., Woodcock, J., & Davies, J. (Eds.), *FM'99 – Formal methods: World Congress on Formal Methods in the Development of Computer Systems,* Toulouse, France, September 1999. LNCS 1708. Berlin; Heidelberg: Springer, pp. 76-95.
6. Ellis, C.A., S.J. Gibbs and G. Rein (1991). Groupware: some issues and experiences. In: *Communications of the ACM,* 34, 1 (Jan. 1991), pp. 39 - 58.
7. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley.
8. Hofte, G.H. ter (1998). *Working Apart Together: Foundations for Component Groupware.* Telematica Instituut, Enschede, The Netherlands, ISBN 90-75176-14-7. https://doc.telin.nl/dscgi/ds.py/Get/File-7623/
9. Hollingsworth, D. (1995). *The Workflow Reference Model,* TC00-1003 Issue 1.1. Workflow Management Coalition.
10. Iacob, M.E., and A. Smit (2001). *Electronic Markets: classification and reference model,* TI/RS/2001/021, Telematica Instituut, Enschede, The Netherlands. https://doc.telin.nl/dscgi/ds.py/Get/File-15266/
11. ISO – International Organisation for Standardisation (1984). *Basic Reference Model for Open Systems Interconnection,* ISO 7498. ISO.

12.ISO – International Organisation for Standardisation (1986). *Transport Service Definition,* ISO 8072. ISO.
13.ISO – International Organisation for Standardisation (1995). *Reference Model for Open Distributed Processing,* ISO/IEC DIS 10746-1. ISO.
14. Object Management Group (1997). *A Discussion of the Object Management Architecture.* Object Management Group, Inc.
15.Porskamp, P.A.P., and A. Tokmakoff (2000). *Content Engineering Architectures,* TI/RS/2000/055, Telematica Instituut, Enschede, The Netherlands. https://extranet.telin.nl/docuserver/dscgi/ds.py/Get/File-9813.
16.Rechtin, E., and M.W. Maier (1997). *The Art of Systems Architecting,* CRC Press, Boca Raton, Florida.
17.Shaw, M., and D. Garlan (1995). Formulations and Formalisms in Software Architecture. J. van Leeuwen (ed.), *Computer Science Today: Recent Trends and Developments.* Springer-Verlag LNCS Vol 1000, pp. 307-323.
18.Slagter, R.J. (2004). *Cooperating People and Systems: Guiding the Design of Component Groupware.* Telematica Instituut, Enschede, the Netherlands, (forthcoming).
19.Steen, M.W.A., M.M. Lankhorst, and R.G. van de Wetering (2002). Modelling Networked Enterprises. In Duddy, K. & Aagedal, J.Ø. (Eds.), *6th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2002),* Lausanne, 17-20 September 2002., pp. 109-119.
20.Technology Strategy Team (1999). E-commerce: the Third Wave (2nd ed.). White paper. US: EC Cubed, Inc.
21.Timmers, P. (1998). Business Models for Electronic Markets. *EM - Electronic Markets,* 8 (2), 3-8.
22.Vissers, C.A., G. Scollo, and M. van Sinderen (1988). Architecture and Specification Style in Formal Descriptions of Distributed Systems. In: Aggarwal, S. and Sabnani, K. (eds.) *Proc. IFIP 6.1 on Protocol Specification,Ttesting and Verification,* Atlantic City, pp. 189-204.