Protocols versus Objects: Can Models for Telecommunications and Distributed Processing Coexist?

Marten van Sinderen, Luís Ferreira Pires
Centre for Informatics and Information Technology, University of Twente
sinderen@cs.utwente.nl, pires@cs.utwente.nl

Abstract

This paper identifies two paradigms that influence the design of telematics systems nowadays: the protocol-centred and the object-centred paradigm. Both paradigms have been introduced to cope with interoperability, each in their own way. The coexistence of these paradigms can have enormous impact on the design telematics systems. This paper identifies some combined uses of both paradigms and some fundamental research problems related to the coexistence of these paradigms.

1 Introduction

Telecommunications and distributed computing have become strongly related research areas [7, 9]. Where telecommunications used to concentrate on data transport (carrier) networks and distributed computing on end-user applications, telecommunications services are now associated with interactive, multimedia network facilities offered to end-users and integrated in distributed applications. Moreover, the terms telecom services and telecom/distributed applications are often used interchangeably. Telecommunications and distributed computing originally have different cultures, and consequently they use different terminology and different sets of concepts, which potentially lead to confusion whenever professionals from these two areas cooperate.

The differences between telecommunications and distributed computing can be reduced to two paradigms, which we call the *protocol-centred* and the *object-centred* paradigm. The former has traditionally been adopted by the telecommunications community and the latter originated in the computing community. An interesting observation with respect to the object-centred paradigm is that it is somehow dependent on the protocol paradigm: interactions between objects are supported by a distributed processing environment that 'transforms' the interactions into (implicit) protocols, provides generic services which are used to make the interactions distribution transparent, and internally uses a network infrastructure to accomplish data transfer.

Our interest in these two paradigms stems from the design of telematics systems. These systems support the interactions between people and/or processes over time and distance through the integrated application of telecommunications and information technology. Consequently, the design of telematics systems is based on the use of both paradigms. The thorough understanding of the paradigms is thus relevant here. The questions that inevitably arise are: when do we use which paradigm? for what purposes? how can we guarantee that system parts designed with different paradigms form a consistent system architecture?

The objective of this paper is to precisely characterise the protocol-centred and the object-centred paradigm, and to identify fundamental research problems that relate to the application of the paradigms in a single design instance.

The work presented in this paper is based on current work in the MESH (Multimedia-services for the Electronic Super Highway) project, a project of a Dutch consortium of research institutes, industry and end-users which is partially supported by a grant from the Dutch Ministry of Economic Affairs. This project adopted the overall concepts and principles of TINA as the basis for the design of a flexible platform for telematics. applications. It also intends to use CORBA and the T.130/120 series of recommendations, and therefore has to deal with the combined use of the two paradigms.

The remaining of this paper is organised as follows: section 2 presents the historical background of the different paradigms and the convergence of technologies; section 3 characterises the two paradigms; section 4 illustrates the combined use of the paradigms in existing developments; and section 5 discusses some fundamental problems related to the coexistence of these design paradigms.

2 Historical background

Nowadays we are experiencing the increasing convergence of telecommunications and computing technology [7,

1071-0485/97 $10.00 © 1997 IEEE

9], signifying the maturation of the area of telematics systems. This convergence is happening at least at two levels. First, at an organisational level, through the establishment of mixed consortia of telecommunication equipment manufacturers and computer manufacturers and the liberalisation of the telecommunication market. Second, at a technology level, through the establishment of architectures with a common network and terminal infrastructure that supports integrated multimedia services.

Notwithstanding this convergence of telecommunications and computing technology, the communities in which these technologies originated have different cultures, which may obstruct the development of comprehensive architectures and their application in the design of telematics systems. This section presents the historical background of these cultures.

2.1 Telecommunications

Three decades ago, the telecommunications community was involved with the development of communication protocols that would permit the interconnection of remote computer systems. The effectiveness of these standards was demonstrated in various early networks, such as the ARPAnet, Cyclades, SNA, DECnet, EPSS, Transpac, and Telenet. The need for interconnecting systems from different manufacturers ('heterogeneous systems') led to the establishment of the OSI reference model, a framework architecture used to interrelate and coordinate the development of *protocol standards*. Since most corporate network architectures define a mapping onto the OSI model, the OSI model can be considered as the archetype of a layered *protocol architecture*. It is therefore the best representative of the protocol-centred paradigm.

OSI protocol standards define the minimum functionality required for the interconnection of different systems (PDU exchanges, PDU formats and codings), while leaving maximum implementation freedom to the manufacturers of these systems (abstract service primitives, no restrictions on PDU processing). The OSI model extends all the way up to distributed applications: the highest layer, the application layer, hosts application protocols that define rules for the interworking of application processes. Despite that the OSI protocol standards have not been very successful [4], the OSI model itself has been very influential as a conceptual model for reasoning about communication and interworking in distributed systems until now.

2.2 Distributed computing

In the distributed computing world, the *client-server architecture* became very popular in the 80's, since it enabled the distribution of computing and the sharing of services (including operating system functions, information resources and applications). A client-server architecture divides a distributed application into a *client* and one or more *server* processes operating on different systems connected via a network, thus forming a loosely coupled distributed system. A client presents a user interface to its user, and invokes operations on the server through interactions supported by the network; a server merely responds to requests from the client by performing the requested operation and returning the result. Ideally, a client should hide the interaction with the server and the supporting communication technology from the user, and the server should hide its platform and language-specific implementation from the client [11]. Designers of client-server applications divide the user-defined task into subtasks to be completed either by the client or by the server, and then concentrate on the implementation of these subtasks. Application designers are not so much concerned with an application protocol that defines the interworking of the client and the server. This is the case, since client-server interactions usually follows a simple request-response pattern which can be mapped onto a communication mechanism, such as e.g. RPC, supported by the network.

Object technology was introduced to design client-server applications in order to cope with the increasing complexity of client-server systems and to reduce development and maintenance costs [5]. An object in this context is an encapsulated entity with a unique fixed identity, whose services can be accessed only through well-defined interfaces. An object may play the role of a client or a server, depending on whether it uses or provides services. The use of object technology did not change the characteristics the client-server architecture, as described above, but rather facilitates the implementation of client-server applications by promoting modularity, flexibility and extensibility. The object-centred paradigm is best represented by the object-oriented client-server architecture, which is oriented towards the effective production of re-usable, interoperable and portable software for distributed application components.

Figure 1 illustrates the trends in distributed application development. It positions the technologies and base architectures associated with telecommunications and distributed computing that led to the protocol-centred and object-centred paradigm, respectively, and lists some specific standard or 'open' architectures that progressively merge technologies as the result of joint efforts of participants from both communities. Some of the specific architectures (notably CORBA and TINA) are discussed in section 4.2.

3 Characterisation of the paradigms

This section characterises the object-centred and protocol-centred paradigms based on a generic model of distrib-

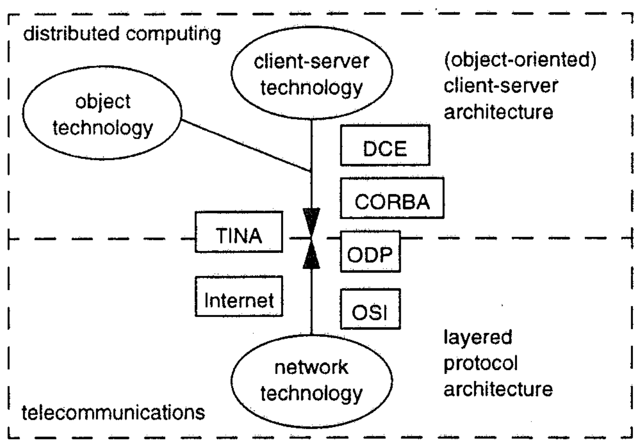

Figure 1. Trends in distributed application development

uted systems that is independent of these paradigms.

3.1 Distributed systems

The Webster's dictionary provides a definition of system particularly applicable to distributed systems:

A system is a regularly interacting or interdependent group of items forming a unified whole.

System parts of a distributed system have to interact in order to perform the system's task. Physical interconnection between parts is the minimal requirement for their interaction. An interaction means that copes with physical interconnection and data transfer between system parts is thus necessary for the proper functioning of a distributed system.

Figure 2 depicts a distributed system.

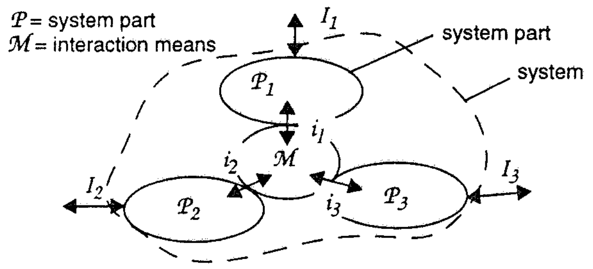

Figure 2. Distributed system and its parts

The following aspects of the system parts and the interaction means are used to characterise the design paradigms:

- local interactions between system parts and the interactions means (i_j in Figure 2);
- unambiguous understanding of information exchanged between system parts;
- coordinated behaviour of parts.

Although the interaction means provide for the physical interconnection and information exchange between system parts, each system part still needs to locally interact with the interaction means in order to indirectly (remotely) interact with other system parts. Furthermore, information exchanged between system parts has to be unambiguously understood by these parts. The unambiguous understanding of exchanged information enables inter-operability between parts, and is provided by a syntax definition with proper semantics (meaning). The coordinated behaviour of the system parts is necessary to assure that the parts together perform the system's task.

3.2 Protocol-centred design

In the protocol-centred paradigm, system parts are protocol entities, and the system as a whole provides a *service*. The interaction means between protocol entities is a *lower level service*. Therefore the model of the system to be built consists of a collection of protocol entities and a lower level service, for each protocol layer. The definition of protocol layers can be recursive each time a (lower level) service is identified, generating a so called *protocol stack*.

The lower level service provides physical interconnection and (reliable or unreliable) data transfer between protocol entities. Lower level services can have arbitrarily complex interaction patterns with the protocol entities, varying from connectionless data transfer (e.g., 'send and pray') to complex control facilities (e.g., handshaking with three-party negotiation). Therefore a lower level service may not be limited to support data transfer, since it may also support complex interactions between protocol entities.

Protocol entities communicate with each other by exchanging PDUs through the lower level service. PDUs define the syntax and semantics for unambiguous understanding of the information exchanged between protocol entities. The behaviour of a protocol entity defines the service primitives between this entity and the service users, the service primitives between the protocol entity and the lower level service, and the relationships between these primitives. The protocol entities cooperate in order to provide the requested service [10].

Operational characterisation. A method based on the protocol-centred paradigm consists of defining the service to be supported by the system in terms of service primitives and their relationships, and decomposing this service in terms of a structure of protocol entities and a lower level service. This structure, which we call a protocol, formally has to be a correct implementation of the service.

3.3 Object-centred paradigm

In the object-centred paradigm, system parts are *objects*, such that the model of a distributed system to be built consists of a collection of interacting objects. The interaction means between objects in this paradigm normally supports a limited set of communication patterns, related to so called *interface types*. The most common interface type is the *operation interface*, in which a (client) object requests some operation from a target object. The interaction means forwards this request to the target object, which generates a response. This response is forwarded back to the client object by the interaction means.

Unambiguous understanding of information exchange is achieved since the objects are capable of knowing each others interfaces. An operation interface, for example, is defined in terms of a collection of operations (methods) in terms of the input and output parameters (signature) of these operations. Interface definitions make it possible for objects to meaningfully invoke each other. Therefore the knowledge of interface definitions by objects makes it possible for objects to understand each other.

Operational characterisation. The most popular design methods based on the object-centred paradigm consist of performing domain analysis to identify the relevant classes for the application domain, refining these classes to create a model of the system supporting the application, and distributing instantiations of these classes (objects) in a way that suits other requirements of the system being developed. Examples of these methods can be found in [1] and [8].

4 Combined uses of paradigms

This section illustrates the combined the use of the protocol-centred and object-centred paradigm in two existing developments, viz. the Abstract Service Definition Conventions (ASDC) of ITU and the Telecommunications Information Networking Architecture (TINA) of the TINA Consortium (TINA-C).

4.1 Abstract service definition conventions

The ASDC [2] have been used by ITU to define its recommendations (standards) for message handling and directory systems. With ASDC, a complex distributed information processing task can be represented by a collection of interacting objects, based on a division of the overall task into subtasks. This representation can be done at different levels of granularity. The philosophy of the ASDC is to start at a coarse level, e.g., by only distinguishing a system object and one or more user objects, and then to work towards progressively more detailed levels. This process is supported by a refinement technique, which allows the replacement of an object by a composite object that consists of a collection of interacting component objects. The technique of refinement is recursively applied until objects that can be considered 'atomic' are obtained. An atomic object represents an application process that operates on a single machine; its implementation can therefore be left to independent manufacturers. Until this point, design basically follows the object-centred paradigm. The primary objective of ITU recommendations is, however, to specify how distributed information processing tasks can be achieved through the interworking of application processes according to standardized protocols. Therefore, a transition to the protocol-centred paradigm is necessary. The ASDC recommendation describes how this transition takes place. In particular, it describes how an OSI realisation of the (abstract) interactions between two associated objects can be accomplished in an often trivial way.

Figure 3 illustrates the mapping of the ASDC model onto the OSI model. In order to explain this mapping, we have to mention some of the concepts as defined by ASDC and OSI, respectively.

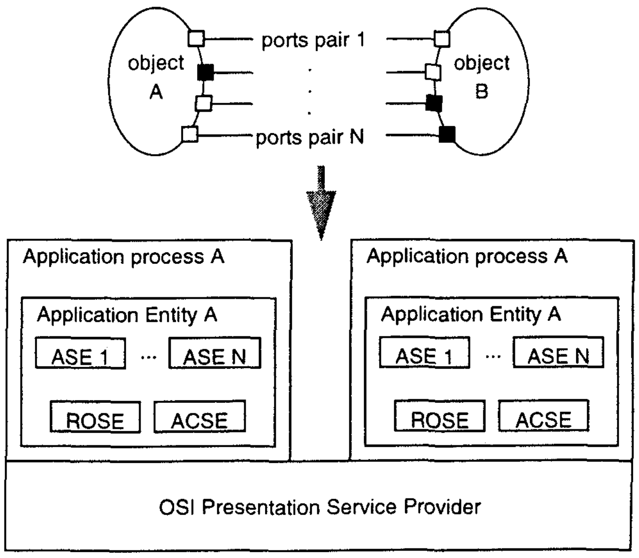

Figure 3. OSI realisation of interactions between two associated objects

Object interfaces are called ports. Two objects can interact with one another when a port of one is bound to a port of the other. Only matching ports can be bound, which means that the ports have the same set of associated operations and either both objects can invoke these operations on each other (symmetrical case) or they can invoke different subsets of the operations on each other (asymmetrical case).

An interaction between two objects corresponds to an operation that can be invoked in the context of two bound ports. Each operation is specified by an optional argument (to be supplied by the invoker of the operation), an optional result (to be supplied by the performer of the operation), and a set of distinct errors (one of which is supplied by the performer of the operation, if the operation could not be completed). There are two special operations which are always supported by an object, either as invoker or as performer, or in both roles: the bind and unbind operation, with which two objects can be associated and disconnected by binding and unbinding one or more of their ports, respectively.

The set of interactions between two objects can now be mapped onto an OSI protocol stack in the following way (see also Figure 3):

- ports pairs and their associated operations are implemented by separate Application Service Elements (ASE), typically named after the ports. Each ASE defines the data structures of the arguments, results and errors of its operations, and uses the Remote Operations Service Element (ROSE) to exchange this information according to a RPC-like communication pattern.
- the general framework for remote invocation of operations is provided by ROSE. ROSE uses the data transfer service facility of the Presentation Service to exchange invocations, results and errors of 'normal' operations, and it uses the Association Control Service Element (ACSE) to support the bind and unbind operation.
- the establishment and release of an association between two application processes is accomplished by the ACSE, which uses in turn connect and release service facilities of the Presentation Service. ACSE ensures that the application processes have a common understanding (known as the application context) of the ASEs that are to be used on an association.

4.2 Telecommunications Information Networking Architecture

TINA [3, 12] is an open architecture for telecommunications systems, currently being developed by a consortium of telecommunications equipment manufacturers and computer manufacturers. The TINA architecture provides a set of concepts and principles to be applied in all phases (specification, design, implementation, deployment, execution, and operation) of the telecommunications software lifecycle. Typical services to be provided by TINA systems are voice-based services, interactive multimedia services, information services, and management services. TINA-C uses object-oriented analysis and design to improve interoperability, re-use of software and specifications, and flexible placement of software on computing platforms or nodes.

TINA is concerned with two layers of software: the telecommunications applications, that is the software actually implementing the telecommunications services, and a Distributed Processing Environment (DPE), that is the software supporting the distributed execution of telecommunications applications. TINA experiments normally use CORBA [14] as a DPE.

CORBA is an open distributed object computing infrastructure, standardized by OMG (an organization of software vendor and object technology user companies). It defines a client-server interaction model according to which a client object can invoke an operation on a target object (the server) with a high level of transparency. Transparency relieves the application designer from some common tasks in networked applications, such as object registration, location and activation, and the handling of requests and responses.

TINA also identifies different areas of concern for telecommunications systems: services, networks, management, and computing. Each of these areas is covered by a separate architecture, with its own set of concepts and components. For example, the TINA service architecture [13] defines service components that can be mapped onto CORBA objects for implementation. TINA applications are developed as objects that interact with the TINA service components. These objects can in principle also be mapped onto CORBA objects.

A DPE may consist of objects of generic functionality (e.g., naming and trading objects) and use protocol stacks that are either available in the Native Computing and Communication Environment (NCCE) or implemented in the DPE itself. This implies that the design of a DPE should also follow the protocol-centred paradigm. In CORBA, for example, the ORB possesses characteristics of the protocol-centred paradigm, whereas the objects that provide transparency are developed according to the object-centred paradigm, and are invoked through the ORB.

Figure 4 depicts a schematic view of a DPE.

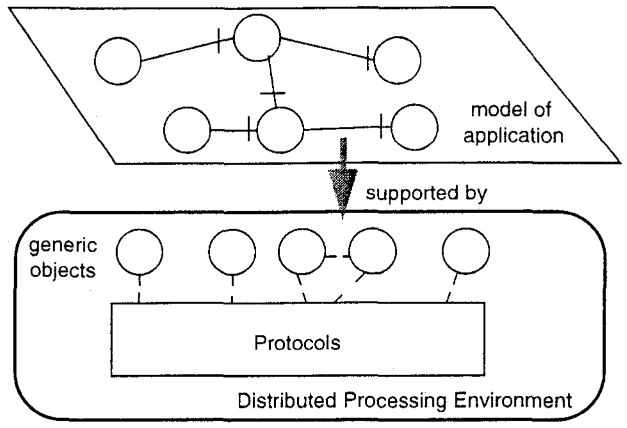

Figure 4. DPE and its applications

In Figure 4 the boundaries of the DPE symbolise the functionality incorporated in the DPE, as opposed to functionality incorporated in the application objects.

5 Discussion

The following observations can be made on the use of the two paradigms by considering the ASDC and TINA/CORBA:

- an object model can be roughly transformed to a protocol model by mapping interface definitions onto PDU exchanges. In the protocol-centred paradigm some additional protocol functions to manipulate the lower level service and to code and decode PDUs have to be explicitly defined. This is exemplified by the mapping of the object interactions in the ASDC model onto service elements in the OSI application entity;
- notations can facilitate the mapping of object interactions onto PDU exchanges. In the ASDC, the ASN.1 notation is used to accomplish this mapping, and in CORBA this role is played by OMG-IDL;
- a distributed processing environment should support the applications, in terms of a composition of application objects. Therefore the capabilities of the distributed processing environment have to match the nature of the interactions between objects in the object model.

In Figure 4 we have identified a DPE boundary. This boundary may be shifted in case advances in technology can influence the type of interactions between objects to be supported by the DPE. An example that proposes to introduce the concept of object group in CORBA, in order to explore the multicast facilities of native networks, can be found in [6]. Proposals for supporting streams in CORBA are also becoming popular. These developments have to be orchestrated with care, otherwise different object models will arise, endangering interoperability. Since IDL is the notation used to represent the potential interactions between objects, modifying the interactions in the object model lead to dialects of IDL.

Designers should have some guidelines to determine when to use the protocol-centred or the object-centred paradigm. The protocol-centred paradigm has been successfully applied to develop systems that cope with (end-to-end) connectivity and reliable communication. Some traditional applications (electronic mail, file transfer, etc.) have been developed according to this paradigm. The protocol-centred paradigm addresses interoperability problems in an implementation-independent way. The object-centred paradigm has been successfully applied to accelerate the design, implementation and modification of distributed applications. The object-centred paradigm supports interoperability of application objects written in different programming languages, e.g., by defining mappings from IDL to these programming languages.

A role of thumb could be to use protocols whenever complex interaction patterns are required and data structures (PDUs) are simple, and to use objects whenever the required interaction pattern is simple and can be supported by an available DPE, and the data structures (operations) are complex.

A number of research questions are yet to be answered, such as (amongst others):

- which application *requirements* are better addressed and supported by one paradigm or the other?
- what are the consequence for *interoperability* of systems built according to different paradigms?
- in case both paradigms are used for building a single system, where to place the *boundary* between the two (i.e., where to place functionality, either based on one or the other paradigm)?
- how to keep *consistency* between models of functionality at the opposite sides of this boundary?
- what is the impact of technological *evolution* at both sides of a DPE boundary?

References

[1] G. Booch. *Object-oriented analysis and design.* The Benjamin-Cunnings Publishing Company, Inc, California, USA, second edition, 1994.

[2] CCITT. *Recommendation X.407. Message handling systems: Abstract service definition conventions.* CTIT Blue Book Volume VIII, pages 200-227. International Telecommunication Union, Nov. 1988.

[3] F. Dupuy, G. Nilsson, and Y. Inoue. The TINA consortium: toward networking telecommunications information services. *IEEE Communications Magazine*, pages 78–83, Nov. 1995.

[4] T. Kalin and D. Barber. Has the OSI opportunity been fully realised? *Computer Networks and ISDN Systems 25*, pages 227–239, 1992.

[5] T. G. Lewis. Where is client/server software headed? *IEEE Computer*, pages 49–55, Apr. 1995.

[6] S. Maffeis and D. C. Schmidt. Constructing reliable distributed communication systems with CORBA. *IEEE Communications Magazine*, 35(2):56–60, Feb. 1997.

[7] D. Messerschmitt. The convergence of telecommunications and computing: what are the implications today? *Proceedings of the IEEE*, Aug. 1996.

[8] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design.* Prentice-Hall, Englewood Cliffs, 1991.

[9] D. C. Schmidt. Distributed object computing. *IEEE Communications Magazine*, 35(2):42–44, Feb. 1997.

[10] R. Sharp. Principles of protocol design. Prentice-Hall International Series in Computer Science. Prentice-Hall, Great Britain, 1994.

[11] A. Sinha. Client-server computing. *Communications of the ACM*, 35(7):77–98, July 1992.

[12] Telecommunications Information Networking Architecture Consortium. Overall concepts and principles of TINA. Version 1.0, Feb. 1995.

[13] Telecommunications Information Networking Architecture Consortium. Service architecture. Version 4.0, Dec. 1996.

[14] S. Vinoski. CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55, Feb. 1997.