```
From: 'Advanced Information Systems Engineering'
9th International Conference, CAiSE'97,
A. Olivé and J.A. Pastor (Eds),
Springer Verlag LNCS 1250, Pages 361--373, 1997
```

Design of Distributed Applications based on the OSI Model

Peter McBrien

Dept. of Computer Science, King's College London
Strand, London WC2R 2LS
mailto:pjm@dcs.kcl.ac.uk
http://www.dcs.kcl.ac.uk/staff/pjm

Abstract. This paper demonstrates how the fundamental concepts of a layered design found in the OSI model may be realised in the **dataflow diagram** (DFD) type approach to application development. The work describes a coupling between DFD and the OSI model, and describes a methodology using that coupling to design implementations of applications which conform to the OSI model. The work serves to complement existing approaches to the conceptual modelling of distributed information systems.
Keywords: Distributed information systems, implementation design, dataflow diagrams, OSI model.

1 Introduction

The **open systems interconnection** (OSI) model [8] has gained widespread acceptance as the reference model in which to describe, design and implement distributed applications. As yet, many of the concepts in the OSI model have not been fully exploited in development methodologies, whether of the 'traditional' DFD type (such as SSADM [1]) or newer OO type (such as Fusion [4] or Unified [2]). Although descriptions of development techniques, such as those presented at the recent IFIP WG8.1 [11] obey the general principles of the OSI layered structure, they do not concern themselves with the *detail* of how to design the implementation of one layer based upon the implementation of the layer below. Rather, they describe how the concepts of the distributed application relate to the different layers.

The aims of this paper are (1) to demonstrate how the fundamental concepts of a layered design found in the OSI model may be realised in the **dataflow diagram** (DFD) type approach [6, 5, 7] to application development, and (2) to describe a methodology built upon this coupling for designing implementations of distributed applications. As such, the work in this paper acts as a general approach to the implementation design of distributed applications, and complements existing work dealing with the conceptual modelling of distributed systems.

The remainder of this paper is structured as follows. We give a brief description of the OSI model in Sect. 2, before describing a coupling between OSI concepts and DFD concepts in Sect. 3. This coupling is then used as the basis for the description in Sect. 4 of a methodology for implementing information systems based on the OSI layered approach.

2 Background Information on the OSI Model

The OSI model defined in ISO7498 [8] adopts a layered approach for the design of distributed information systems. Each layer provides a set of **services** — the capability to make some form of communication over the distributed system. Each service is implemented by one or more processes, each of which is termed an **entity**. Entities can communicate with **peer-entities** (located within the same or other hosts) in the same layer by exchanging messages called **protocol data units** (PDUs). The set of valid exchanges of different types of PDU between a pair of peer-entities forms the protocol they obey. Since the layers are numbered, we may in general speak of (N)-entity, (N)-PDU, *etc*, when speaking of constructs associated with layer number N. The *implementation* of a message exchange in an (N)-layer is made by using the **services** of the (N-1)-layer below. The services are accessed via one or more **service access points** (SAPs), by invoking **service-elements**.

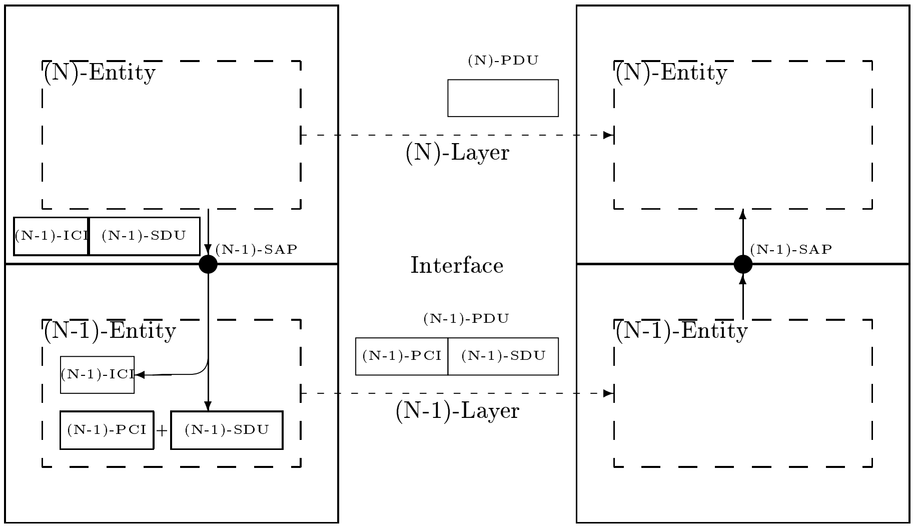

Fig. 1. Basic Arrangement for data being sent over an interface between layers

Figure 1 illustrates the basic arrangement for implementing an (N)-protocol, by showing how the virtual transfer of an (N)-PDU is implemented by the call via (N-1)-SAP to use an (N-1)-service, which the uses (N-1)-PDUs in executing the protocol for the service. The diagram shows how an (N)-PDU is put inside the

(N-1)-SDU field given to the (N-1)-SDU. The instruction telling the (N-1)-service what should be done with the (N-1)-SDU is contained within the (N-1)-ICI. The (N-1)-service is implemented by exchanging (N-1)-PDUs containing the (N-1)-SDU. (The diagram should not be interpreted as making it necessary that there is a one-to-one correspondence between the PDUs of different layers).

The above description of the OSI model only describes those features which we shall describe in the form of DFD models in the next section. The ISO standard should be consulted for details of the features of the OSI model not relevant to the discussion that follows.

3 Associated Between DFD Models and OSI Layers

In this section we will discuss how DFD may be used to model an information system structured according to the OSI model. We first make the (obvious) correlation between one layer of the OSI model and a DFD model, before discussing the association between DFD models for adjacent layers.

3.1 Modelling A Single Layer

An entity in the OSI model corresponds to the general notion of a process — in the sense of being a running program — which is involved in communication with other processes. This notion is modelled in DFD as a process, and thus we will model OSI entities as DFD processes. However not all DFD processes are OSI entities, since a DFD process may be decomposed into subprocesses. Thus we will need to indicate that a DFD process corresponds to an OSI entity by marking the process with an asterisk, and term an object so marked as an **entity-process** (or just **entity** where a distinction from an OSI entity is not required). We may say each entity-process, and all processes which are part of its decomposition, are **members** of the entity-process.

The communication between OSI entities is made by the exchange of PDUs. This notion is modelled in DFD by dataflows, and thus we will model OSI PDUs as DFD dataflows between processes which are members of distinct entities. Each type of PDU will be given a distinct DFD dataflow name. Access to OSI entities is made via a SAP which corresponds to the notion of an interface to the system. This is indirectly modelled in DFD by the use of external agents, and we will characterise an SAP by using a DFD external agent. The service elements of the SAP will be modelled as dataflows between the external agent and the processes which implement the services.

3.2 Association Between Layers

The modelling presented above of a single OSI layer using DFD models is an obvious one, but leaves unanswered the question of how one OSI layer is connected to other layer. It might seem appropriate to use process decomposition to model the implementation of an (N)-entity by several (N-1)-entities. However this would be an incorrect modelling of the OSI concept of an entity, since several (N)-entities may be served by an (N-1)-entity, as well as several (N-1)-entities serving a single (N)-entity.

We note that the interface to each layer (the OSI SAP) is modelled as an external agent, and it is the (N-1)-SAP that an (N)-entity uses when implementing the transfer of (N)-PDUs. Since flows represent classes of (N)-PDUs; we are in effect stating that external agents of the (N-1)-layer are associated with those flows of (N)-layer which pass between processes which are members of distinct (N)-entities.

3.3 Implementation Methods

When we wish to implement the application modelled in the (N)-layer, we have three alternative strategies.

1. Implement only the (N)-entities, and use existing implementations of (N-1)-services. This is will usually be the preferred option, since it permits the application to be implemented with the minimum development costs in terms of building new network protocols at lower layers.
2. Implement one of the (N-1)-entities, and use an existing implementation of the other (N-1)-entity. This technique will typically be applied when the (N-1)-layer operates on a client-server basis, and a standard server implementation is to be used in conjunction with a user developed client implementation containing special features required at the client.
3. Implement both of the (N-1)-entities, possibly using a new protocol for their PDU exchange. This technique will typically be applied when there are no suitable (N-1)-services which meet the requirements of the (N)-layer.

Should either choice (2) or (3) be taken, then not only will the design methodology described below need to be executed for the (N)-layer, but the also on the (N-1)-layer.

4 Design Methodology

Having described in the previous section a coupling between DFDs and OSI models, we now describe a methodology by which we may use a DFD model of an (N)-layer to design the implementation of the model using the services of the (N-1)-layer.

When implementing an (N)-layer using the (N-1)-layer, it is normal that there is a choice of services made available in the (N-1)-layer. The choice of which services are used by an application depends on the requirements of the process model in the (N)-layer. The service in the (N-1)-layer can mismatch with the requirements of the (N)-layer in one of two ways:

- The service may fail to meet the requirements of the process model, leading to an incorrect implementation of the process model.

– The service may support more than the requirements of the process model, leading to waste of resources in the distributed system.

Since we would need to avoid such consequences, it is important that a close match be made between the requirements of the (N)-layer and the services provided by the (N-1)-layer. This involves us first carefully identifying properties of the (N)-layer, and then matching those properties to one or more services in the (N-1)-layer. Our methodology will be based on the principle that the (N)-layer DFD should be decomposed to a **well formed DFD** which clearly identifies which (N-1)-services are used, and what requirements are being put upon them. The well formed DFD has the following properties:

- Each (N)-process connected via a dataflow to another (N)-process in a distinct (N)-entity is termed a **service-user process** and marked by a plus sign.
- Each (N)-service-user process is associated with exactly one (N-1)-service, so that the use of (N-1)-services is clearly identified and localised in the (N)-layer.
- An (N)-flow between a pair of (N)-processes is only allowed if either:
  1. The (N)-processes are part of the same (N)-entity, or
  2. The (N)-processes are (N)-service-user processes implemented by the *same* (N-1)-service.

  The first option is allowed on the basis that the communication will be made members of the same (N)-entity using normal intra-application communication, the second is allowed on the basis that two OSI (N)-entities can communicate using a particular (N-1)-service.

It should be noted that the above notion of well formed DFD will force a decomposition based on the (N-1)-services that each (N)-process uses. This means that we must identify the requirements each (N)-process has for (N-1)-services. The types of requirements an (N)-process might have will very between applications, but the common core aspects listed in the next subsection — all taken from the OSI standard — will normally need to be addressed. Before using the design method for an application is important that the property list be amended to include those particular to the application. The possible choices here is very large, and include such things a replication, fault-tolerance, financial cost of use, *etc.* The surveys presented by [10, 3], and in particular the taxonomy presented by [9] provide useful guidance in this activity. Once chosen, the properties should be divided into those for processes, and those for flows.

4.1 Properties of Processes

In identifying process properties, in the context of this work, we are only interested in those aspects of a processes' behaviour which impact on the manner in which it interacts with other processes via flows. When discussing the properties, we also identify which values of the property make it a more restrictive property, in terms of placing greater demands on the service implementing the property. The following **process properties** may be identified:

1. An (N)-process is involved in **synchronous communication** will generate certain (N)-flows which lead to it being suspended until a response via other (N)-flows reaches the (N)-process. Alternatively, an (N)-process may generate more flows before waiting for a reply to the first flow in which case it is involved in **asynchronous communication**.
   A process that can handle asynchronous communication can always work on a synchronous communications system, but the reverse is not true, and therefore the synchronous communication can be regarded as the more restrictive property.
2. Communication between peer-processes may be **peer-to-peer**, in that each entity may generate any of the permitted (N)-PDUs, and hence all flows are bidirectional. Alternatively, communication may be **client-server** where one (N)-process will only generate flows in response to receiving flows from another (N)-process.
   Handling peer-to-peer communication is more restrictive property since it entails that each process can handle all dataflow inputs that may be generated between two (N)-processes (*i.e.* all dataflows are bidirectional), whilst the client-server model limits the number of flows a particular process must handle.

4.2 Properties of Flows

In the context of this work, we are only interested in studying those properties of a flow which impact on the method that the communications system handles instances of the flows. The following **flow properties** may be identified:

1. A **connection oriented** (CO) (N)-flow will ensure that flow instances generated by one (N)-process will reach the destination (N)-process in the order that they were sent. By contrast, a **connectionless** (CL) flow may deliver to the destination the flows in a different order from that in which they were sent.
   Any protocol which works correctly on a CL communications system will also work correctly on a CO system, and thus CO is the more restrictive property.
2. An (N)-flow may have **maximum transfer unit** (MTU), which places an upper limit in bytes on the message size contained within dataflows.
   Any protocol which works correctly on a system with a smaller MTU size will also work correctly on a system with a larger MTU size, and thus a larger MTU size is a more restrictive property.
3. An (N)-flow may require a certain **Quality of Service** (QoS) in order that the protocol that it implements is reliable. In the ISO7498 standard it is specified that the exact choice of which parameters that should be combined to form this notion of reliability is left to the designer of a system. Examples of the parameters to include the probabilities of corruption, wrong delivery (going to the wrong destination), or loss of flow instances.

In general a lower QoS indicates a worse quality of service — *i.e.* greatly likelihood of things going 'wrong' — and therefore a protocol which works with a lower QoS will also work with a communications system with a high QoS, whilst a protocol requiring a high QoS will not work well with a low QoS communications system. We thus identify that requiring a high QoS is the more restrictive property.

4. Most distributed systems are heterogeneous in nature, leading to the requirement that data be exchanged using a common **transport data syntax**. An (N)-flow may either use that syntax, or require that the (N-1)-layer can accept an **abstract data syntax** (ADS) and marshals the data for transport between hosts.
   A protocol that works without an ADS being provided will also work if an ADS is available. If the protocol requires an ADS, and an ADS is not available, then the protocol will not work. Thus requiring an ADS is the more restrictive property.
5. The (N)-flow may require a **secure channel** over which to operate if the flow contains confidential information, and the processes which use the flow do not employ any encryption mechanism. Although mentioned in ISO7498 as an example of one of the QoS parameters, the growing use of public networks such as the Internet mean that this property warrants direct attention as a particular property.
   Since an (N)-flow which is does not contain confidential information may still use a secure channel, but an (N)-flow containing confidential information should not pass over an insecure channel, we identify that requiring a secure channel is the more restrictive property.

4.3 Matching Services to Property Requirements

We may characterise any (N)-layer process model we are developing as requiring a certain level of support for each of the above service properties, and then draw up a **service-property** matrix, comparing the services available in the (N-1)-layer to what they provide for each property, and then select those services which meat the properties for the (N)-layer protocol.

For example, taking a few common services from the TCP/IP model [12], we may tabulate the properties of each in a **service-property matrix** as illustrated in Table 1. For the QoS figures, we have chosen a simple scale of 0 to 2, where 0 indicates a unreliable service, 1 indicates some error control, and 2 indicates a reliable service. It is not relevant to this discussion to describe what the features of each service are, but details may be found by referencing the **request for comment** (RFC) listed in the table for each service. However, to illustrate the process of how an entry is created in the table, we shall consider the **trivial file transfer protocol** (tftp) in some detail with respect to the seven properties in our matrix:

synchronous The tftp service expects the user to wait and handle a reply to each command issued, and therefore is synchronous.

Table 1. Service-property matrix

service		property							layer
name	RFC	synchronous	peer-to-peer	CO	MTU	QoS	ADS	SC	
SMTP	821	$\times$	$\surd$	$\surd$	∞	2	$\times$	$\times$	application
tftp	1350	$\surd$	$\times$	$\times$	2^{25}	1	$\times$	$\times$	
ftp	959	$\surd$	$\times$	$\surd$	∞	2	$\times$	$\times$	
SUN-RPC	1057	$\surd$	$\times$	(1)	(1)	(1)	$\surd$	$\times$	
NFS	1094	$\surd$	$\times$	$\times$	∞	2	$\times$	$\times$	
TCP	793	(2)	(2)	$\surd$	∞	2	$\times$	$\times$	transport
UDP	768	(2)	(2)	$\times$	$2^{16}-1$	0	$\times$	$\times$	

(1) Depends on transport layer protocol used
(2) Concepts of synchronous and peer-to-peer are application layer specific

peer-to-peer There is the notion of tftp servers to which clients connect, and therefore it is not peer-to-peer.

CO Each command is handled separately, without the notion of logging in to the service, and therefore it is not CO.

MTU Up to 2^{16} blocks of 512 bytes may be sent.

QoS Some error checking is performed on data transfer, but other commands may simply be lost or ignored.

ADS The data is a raw stream of bytes, with no attempt at encoding data types into a common syntax.

SC No security is used either for connection or data transfer.

The services in Table 1 have been divided into those at the application layer and those at the transport layer. In general, when building a distributed application, the application layer services should be used, since this avoids the necessity to implement many network protocols. However sometimes it is necessary to build a new application layer protocol based on the services of the layer below, or to rewrite the protocol based on the transport layer services. Thus we include the transport layer services in the matrix.

Matching (N)-processes to (N-1)-services involves the following steps:

1. Listing the requirements on the properties of all *inter-entity* (N)-dataflows, and associated (N)-processes. This may be done using a **process-property matrix** and a **dataflow-property matrix**, similar in structure to the service-property matrix.
2. Matching the requirements in the process-property and dataflow-property matrices with the service-property matrix, such that a list is obtained for each process of all services which satisfy *both* the process *and* the dataflows attached to the process.
3. If no close match is obtained for a process, then that process should be decomposed into subprocesses which do satisfy the conditions in (2).

A simple example application requirement is presented in Example 1, which we shall use to demonstrate how this method works in practice. The first attempt at drawing up process-property and dataflow-property matrices is shown in Table 2.

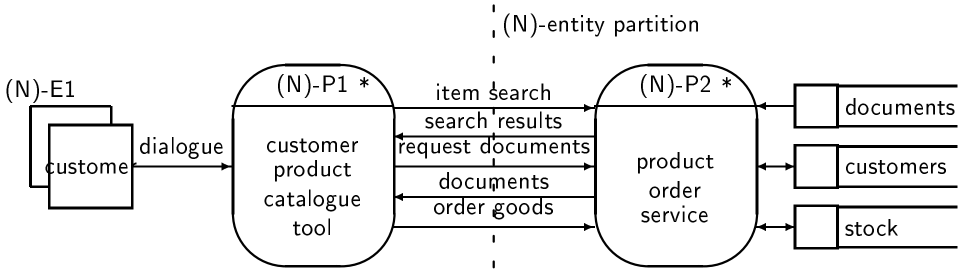

Fig. 2. Product electronic catalogue service: 1st level DFD

Example 1. Product electronic catalogue

Figure 2 contains the 1st level DFD for a proposed Internet application to allow customers to search for product information and order items. The client product catalogue tool (N)-P1 *will allow (1) the customer to obtain documents describing products and (2) the customer to order products. All information will come from a* product data server (N-P2) *held at the company.*

We do not detail the dialogue *between the customer and* (N)-P1 *since it does not impact on the discussion in this paper. The tool made issue a request* item search *giving some keywords, the reply to which is* search results *containing a list of document references. The user may give a list of such references in* request documents*, the request being fulfilled by* documents. *We can verify that documents are no larger than 100KB in size. It is expected that search requests and document requests will be answered in the order in which the customer makes them.*

Customers may supply their credit card details and address together with a list of goods in order goods*, the goods then being dispatched by post. Since the application will be required to work over the Internet, some form of secure channel is required for the credit card details to be sent between the two sites.*

Analysis of Table 2 reveals that overall most requirements are similar for the two process/flow combinations in the model. However one flow, order goods, does not require a CO channel, but does require a secure channel. Secure channels represent a considerable overhead due to the relative complexity of using encryption techniques. The system would be most efficient if a secure channel service were used only for the order goods flow, and the others used a service not providing a secure channel. The system also has a mismatch between requirements and services in that the processes make use of both asynchronous and synchronous communication.

Table 2. 1st level DFD requirements matrices

process	property	
	synchronous	peer-to-peer
(N)-P1	some	×
(N)-P2	some	×

(a) process-property matrix

dataflow	property				
	CO	MTU	QoS	ADS	SC
item search	√	1000	2	×	×
search results	√	10000	2	×	×
request documents	√	1000	2	×	×
documents	√	100000	2	×	×
order goods	×	1000	2	×	√

(b) dataflow-property matrix

Since a well formed distributed process model does not allow one process to use multiple services, we need to decompose the DFD of Fig. 2 to the second level, giving the model shown in Fig. 3. This has separated out the transmission of the order goods to be handled by one pair of processes (N)-P1.2 and (N)-P2.2; whilst the browsing of information supported by the other flows is implemented by (N)-P1.1 and (N)-P2.1. Note that the new flow make order between (N)-P1.1 and (N)-P1.2 does *not* require to be added to the dataflow-property matrix, since it links processes which are members of the same (N)-entity-process.

Table 3. 2nd level DFD process-property requirements matrix

process	property	
	synchronous	peer-to-peer
(N)-P1.1	√	×
(N)-P1.2	√	×
(N)-P2.1	×	×
(N)-P2.2	×	×

Drawing up in Table 3 a process-property matrix for the 2nd level DFD (the dataflow-property matrix is the same as in Table 2, since the only additional dataflow is between processes which are members of the same entity), we see that each second level process only uses only synchronous or only asyn-

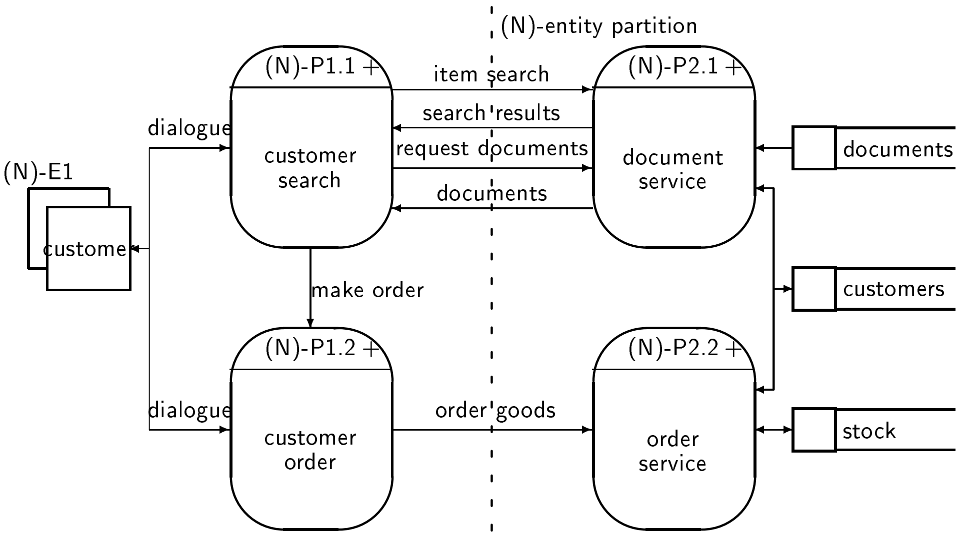

Fig. 3. Product electronic catalogue service: 2nd level DFD

chronous communication. All the dataflows used by (N)-P1.1 and (N)-P2.1 have the same properties, save the minor variation in MTU. Combining the property and dataflow-property matrices for this pair of processes shows that they closely match the ftp service in Table 1. For (N)-P1.2 and (N)-P2.2 and their single associated dataflow, there is a good match with SMTP (i.e. email), except that the SC feature is missing, and the peer-to-peer feature of SMTP is not used. In this circumstance we need to either (1) find a version of SMTP with secure channels, (2) modify our specification of the (N)-processes such they perform there own encryption and thus do not require a secure channel, or (3) implement the secure version of the (N-1) service we require.

5 Summary and Conclusions

The paper has presented a coupling between DFDs and the OSI model, where OSI entities are modelled as collections of DFD processes, and dataflows between processes in distinct entities are implemented by a DFD for the layer below. The implementation of an (N)-layer process model by (N-1)-layer services involves the following steps:

1. Identify the properties of the (N-1)-layer services.
2. Design a conventional DFD for the system, and decide where (*i.e.* on which host) in the distributed system the various processes will be located. The processes at one particular location form one or more entity-processes. (Normally, the part of the application at one location would be implemented as a single entity, though for purposes of software reuse or fault tolerance, the application may be split into several entities at any one host).

3. Identify the property requirements of the those (N)-layer dataflows and associated (N)-processes for which the dataflows pass between (N)-processes which are members of distinct (N)-entities.
4. Aggregate the property requirements found in (3) by grouping them by (N)-process.
5. For each (N)-process, match the aggregated property requirements with those of the (N-1)-layer services. If the match is poor, decompose the (N)-process into subprocesses and repeat from step (4).
6. If no (N-1)-layer service exists for a particular requirement from the (N)-layer process, then either
   - the (N-1)-layer needs to be modelled by another DFD using the same techniques,
   - a new (N-1)-service must be found, or
   - the (N)-process specification be relaxed to have less strict requirements on the (N-1)-layer.

The methodology has been tested on a number of small case studies, and has been shown to be capable of determining the services to use in building distributed applications. It is believed that the methodology will scale well to large systems, since the analysis of the matrices is systematic in nature, and thus should be easy to implement in automated tools. However the methodology still needs to be tested on the development of a large system in order to determine its usefulness on large projects.

Future work will concentrate of developing the methodology to support OO modelling techniques. In principle this will be done by forming a coupling between OO and OSI models, and them using the same methodology for specifying requirements and decomposing the objects so that objects only require one type of service. However the details of this process have yet to be studied.

References

1. C. Ashworth and L. Slater. *An Introduction to SSADM version 4*. International series in software engineering. McGraw-Hill, 1993.
2. G. Booch and J. Rumbaugh. Unified method for object-oriented development. Technical report, Rational Software Corporation, http://www.relational.com, 1996.
3. T.L. Casavant and M. Singhal. *Readings in Distributed Computing Systems*. IEEE Computer Society Press, 1993.
4. D. Coleman *et al.* *Object Oriented Development: The Fusion Method*. Prentice-Hall, 1994.
5. T. deMarco. *Structured Analysis and System Specification*. Yourdon Press, 1978.
6. C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice-Hall, 1978.
7. J.A. Gulla, O.I. Lindland, and G. Willumsen. PPP an integrated CASE environment. In *Proceedings of the Third Nordic Conference on Advanced Information Systems Engineering*, volume 498 of *LNCS*. Springer-Verlag, 1991.

8. Information processing systems — OSI reference model — the basic model. Technical report, International Standards Organisation, 1994.
9. B.E. Martin, C.H. Pedersen, and J. Bedford-Roberts. An object-based taxonomy for distributed computing systems. In *Readings in Distributed Computing Systems* [3], pages 152–169.
10. S. Mullender, editor. *Distributed Systems.* Addison Wesley, 1993.
11. A. Sølvberg, J. Krogstie, and A.H. Seltveit, editors. *Proceedings of the IFIP WG8.1 Working Conference: Information System Development for Decentralised Organisations.* Chapman-Hall, Trondheim,Norway, 1995.
12. K. Washburn and J.T. Evans. *TCP/IP Building a Successful Network.* Addison-Wesley, 1993.