Mobile Networks and Applications 2 (1997) 285–301

Communication protocol design to facilitate re-use based on the object-oriented paradigm

Andrew A. Hanish and Tharam S. Dillon

School of Computer Science and Computer Engineering, La Trobe University, Victoria 3083, Australia

The main motivation for the present work stems from the wide gap which exists between the research efforts devoted to developing formal descriptions for communication protocols and the effective development methodologies used in industrial implementations. We apply Object-Oriented (OO) modelling principles to networking protocols, exploring the potential for producing re-useable software modules by discovering the underlying generic class structures and behaviour. Petri Nets (PNs) are used to derive re-useable model elements and a slightly modified TTCN notation is used for message sequence encoding. This demonstrates a formal, practical approach to the development of a protocol implementation through OO modelling. Our utilisation of PNs in the context of object based modelling allows for isolation of the behavioural characterisation of objects into a separate design plane, treated as a meta-level object control. This separation permits greater execution flexibility of the underlying object models. It is that very aspect of our modelling approach which can be utilised in software implementations where dynamically determined 're-programming' (i.e., change of procedures) is needed. For example, one of the requirements in wireless networking software is the ability to cope with ever-changing transmission/reception conditions and that, in turn, creates greatly varying error rates. Similarly, handoff procedures create situations where dynamically determined change of operational modes is required. To illustrate the modelling concepts, the paper addresses the problem of inter-layer communication among multiple protocol entities (PEs), assuming the standard ISO/OSI Reference Model. A generalised model called the Inter-Layer Communication (ILC) Model is proposed. An example of a PE based on the Alternating-Bit Protocol (ABP) is also discussed. The final example demonstrates how meta-level object control (PNs) allows for the dynamic selection of different ARQ based algorithms.

1. Introduction

1.1. Motivation

When we talk about the communication protocol developers, we mean at least one of the two following groups of professionals:

(i) ISO experts who, usually as members of an expert group, formulate standards and specify communication system services with the relevant protocols. They should be regarded as true protocol designers/developers (on a very high, abstract level). Ideally, Formal Description Techniques (FDTs) are the main supporting tool for the needs of this group.

(ii) Industry experts who specify networking products (based on ISO standards). They have to be able to understand already specified standards and interpret their meaning unambiguously, while making suitable choices for the target environment and product requirements. This second group represents designers and developers of implementations of communication protocols.

There is a noticeable developmental gap between ISO formulated standards and the final form of the communication software products. There is no smooth transition between these two extremes.

In addition to the above, rapid developments in networking software resulting from the introduction of new technologies such as wireless data communication networks, demand more efficient networking software design methods. An application of PNs to object based modelling gives a well-known mathematical formalism to behavioural aspects of object models (which they presently lack).

1.2. The challenges of mobile and wireless networks

The example on ARQ procedures in section 6 illustrates the current refinements on our earlier work on OO design [16]. Mobile and wireless communication systems pose specific demands on software designers. Some of the more important ones are:

- delivery of efficient, compact, replaceable(re-useable) and highly specialised software components,
- ability to cope with volatile radio transmission/reception parameters introducing a high level of uncertainty about the Quality of Service (QOS) provided by radio based networks in the delivery of data packets,
- adaptable (self-configuring and/or replaceable) network management functions (accounting, statistics gathering, real-time monitoring) which, in turn, directly influence the total cost of running the networking infrastructure,
- delivery of supporting software architectural components with configurations that require minimal operator run-time intervention.

In the first years of development in mobile communication systems, precedence was given to solving a number of formidable technical (radio transmission, hardware and call

switching) problems. The recent interest in wireless networks has created the need for a more systematic approach to the adherence to protocol architectures within the framework of ISDN and B-ISDN standards [5,19,23,26,28,35] In our opinion, software self-configurability (mobility) and QOS issues are the most demanding and difficult to tackle, particularly when high data transmission rates of 10 Mb/s or more and multimedia applications are required.

1.3. Present situation

The current industrial practice in network software design and modelling is unsatisfactory. Implementors very often practise a 'black magic art', with substantial duplication of efforts by starting new projects from scratch and re-inventing what others have done. There are a number of reasons justifying this situation. This may be due to commercial secrecy or deadline-driven project management not supporting the generalised, tool building approach to communications software development. Also, Vissers [37] points out that among the ISO experts there is no consensus on how to take the full advantage of using FDTs. The following points are worth emphasising.

- In an overview of the state of the development of Formal Description Techniques (FDTs), Vissers [2] indicated that wider acceptance to using FDTs to describe existing and new protocols and services was still needed in the OSI community. The gap between the high expectations of benefits derived from using FDTs and the everyday industrial practice may remain for a long while. However, in the future the entire process of protocol design, verification, testing and derivation of implementation descriptions is expected to be FDT based.
- While descriptions of protocols are very detailed and many attempts have been made to formalise their verification/validation, very little practical work has been directed towards developing a systematic design methodology which:

  (i) emphasises a detailed algorithmic description from the design specification taking into account the constraints imposed by the implementation environment,

  (ii) identifies underlying primitives and abstractions to form the basis for the re-use of structures and designs when developing a protocol implementation.

- Generally, when beginning a new implementation of a protocol the work is started from scratch, relying perhaps on the ASN.1 engine to aid the development process.
- Abbott and Peterson [1] emphasised the need to develop new approaches to protocol engineering, and pointed to OO modelling as an important basis for such approaches. However, they provided neither a comprehensive conceptual model nor a detailed software structure model as the foundation for an OO protocol engineering and implementation methodology. Bapat [3] and Box et al. [9] provided other interesting insights into the application of OO modelling concepts in the networking area. OO development of real-time distributed systems was presented by Selic et al. [34].
- We believe that the derivation of communication protocol implementations should be done through modelling on a more abstract level, as suggested by Reisig [30,31]. This conceptualisation should be in terms of the model elements at hand and their relationship with other model elements via well-defined interface specifications.
- Several previous approaches have used different types of Petri Nets such as Place-Transition Nets [2,8,27], Numerical Petri Nets [36] and Object-Oriented Petri Nets [24] to model communication protocols. Their main objective was to model a protocol for the purpose of verification. In marked contrast, our approach is to use Petri Nets as a mechanism for development of the infrastructure for implementation.
- We aim to expose the generic aspects of protocol implementation(s) as an OO conceptual model emphasising the dynamic interaction of the objects involved. This allows the early identification of re-useable model elements during the modelling and design stages.

The literature on protocol implementation design treats a PE in isolation, concentrating on protocol features and services. Halsall [12] discusses some of the issues addressed in this paper. From a practical point of view, we contend that it is more beneficial to model an ILC and, within its context, the individual PE's operation. This is why the problem domain is divided into two parts representing two different levels of abstraction which ought to be considered in any network software model, design and implementation. The top level deals with the inter-layer communication (ILC). The corresponding model of the ILC is described in section 3. We view the ILC model as a type of inter-process communication (IPC) mechanism, as detailed in Hanish [13]. The lower modelling level deals with the way an individual PE interacts with its environment. A suitable model for a PE based on ABP is described in section 4.

As suggested in Dillon and Tan [11], section 2 lists the modelling tasks according to the accepted industry practice of identifying at least two modelling phases for the derivation of static and dynamic features. Note that these phases and modelling tasks are limited to conceptual modelling issues only. In section 5, we describe the modelling method and show how a high-level PN is mapped into an OO model element, suggesting a suitable type of PN to utilise. We also derive a model element's behaviour description and encode it using TTCN notation.

2. Object modelling tasks

Consider the general environment in which a networking software sub-system operates within a network node, as depicted in figure 1. The aim here is twofold.

1. Create a general OO environmental model in which
   1.1. the networking software sub-system operates,
   1.2. the PEs and other correspondents interact,
2. Provide maximum flexibility in configuring and initialising all correspondents comprising the model elements.

A summary of the modelling tasks (MT) to be performed follows [11]:

Conceptual Modelling Phase: Static model.

MT1: Analyse the natural language problem statement and select the concepts used.

MT2: Select candidate class objects.

MT3: Determine essential relationships among candidate class objects.

Conceptual Modelling Phase: Dynamic model.

MT4: Identify the model's essential events and messages.

MT5: Petri Net high-level model representation.

MT6: Identify external events and messages.

MT7: Identify internal objects' static structures.

Conceptual Modelling Phase: Component cataloguing.

MT8: Build object class documentation in the form of a class dictionary.

It is worth emphasising at this point that modelling tasks MT1–MT3 may be tackled in a number of ways as described by various methodologies such as OMT, Booch, Yourdon/Coad and recently UML (Rational Corp.).

A question arises, how does our approach differ from other accepted OO methodological approaches? Our approach is complementary in the sense that it allows utilisation of other methodologies. In addition, we propose to put more emphasis in modelling activities on:

- analysis of concurrency,
- utilisation of computational semantics of a category of Coloured Petri Nets (CPNets),
- modelling dynamics of object interactions.

The use of CPNets gives immediate access to a number of well developed algorithms for simulation and formal verification of designs.

To put it briefly, in our approach, we view every design step as visually driven with tabular documentation support assuming that it can also be simulated and formally verified. We also try to provide support for a two level designs:

(1) derivation of re-useable model elements (this activity is akin to analysis),

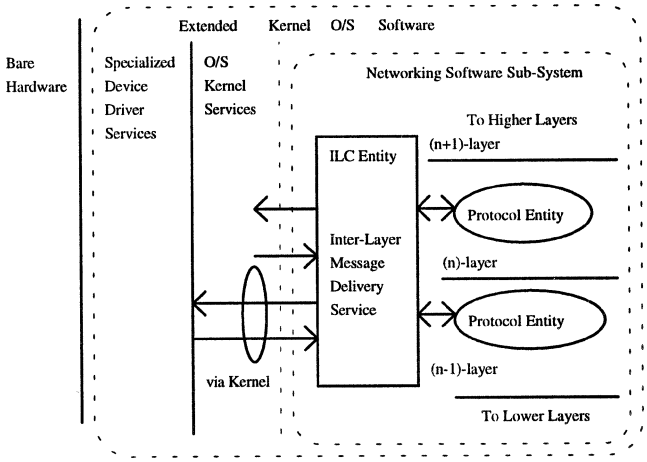

Figure 1. A general view of the networking software operational environment.

(2) composition of new designs by re-using existing elements (this activity is akin to synthesis) from a catalogue of model elements.

3. An overview of the ILC

OSI/ISO layering suggests a certain division of functions and services associated with each layer. However, it is insufficient from the software designer's point of view. Finer granularity of functionality and service selection is required. We start from these general layering principles and associated assumptions and consider a protocol entity as an object existing within an environment defined in the context of the local operating system's facilities and services. In this section, we present a problem statement in a condensed format. We often refer to various parts of the system under consideration as objects without explanations on how they have been derived. Examples of object derivation are given in section 5.1.

From figures 2 and 12, a network node (local system) is viewed as a message sink/generator in relation to other nodes. Therefore it must have message buffering capabilities to adjust the uneven message flow resulting from different message processing rates existing in the external network environment and among the local system's internal objects. In our model this responsibility for flexible buffering of messages rests with the ILC object (called in figure 12 the Mailer object). The flexibility is achieved through associating a pair of queues with each identified correspondent. In consequence, the ILC object controls the operation of InQ and OutQ of each of the correspondents as well as the correspondent's execution state. In figure 13 we distinguish two states, suspended or active with the processing of one message at a time.

In figure 12, it is implied that each PE is an object which executes concurrently with other correspondents, synchronising only through message exchanges. A PE executes in a non-preemptable regime driven by the contents of its input queue InQ. It communicates with other correspondents

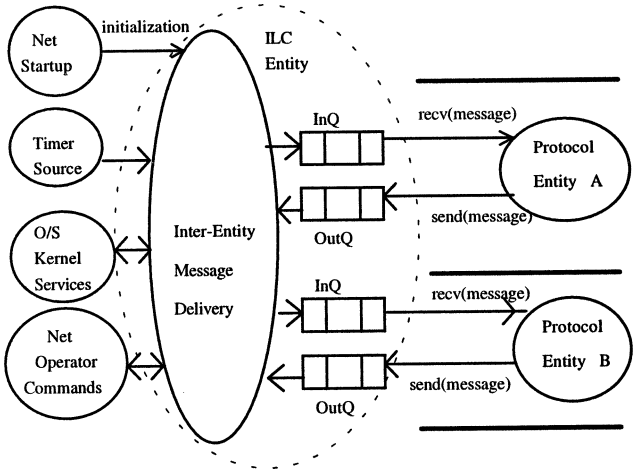

Figure 2. An initial view of the ILC model.

via generic 'send' and 'recv' messages. Suitable access to the timer messages is provided via the TimerEvtSrc object and to the other specialised kernel services such as access to the physical device drivers.

The message format exchanged between any two correspondents within a local system is called a Service Request Block (SRB). An SRB is an object class whose instances become a realisation of the OSI's abstract service primitives. It contains references to all the necessary service details such as service type, parameters and data. Halsall [12] calls a similar data/message functionality an Event Control Block (ECB).

All SRBs have the same format but the repertoire of messages (SRB types) available at individual interfaces to various correspondents differs. Each PE 'knows' the adjacent layer PEs and other correspondents (such as TimerEvtSrc) in terms of their local identities (addressing information). So, each SRB carries the addresses of the source and destination objects. This ensures proper functioning of the ILC object as an SRB delivery service provider (Mailer). As in the ISO/OSI model [15], each PE adheres to the upper and lower interface specifications allowing communication with PEs in adjacent layers.

Each PE follows the specific protocol rules for exchanging messages (called Protocol Data Units – PDUs) with the remote system's peer entity. It also has to 'know' how to address remote peer PEs. The treatment of addressing issues (Service Access Points (SAPs) and NSAP global addressability) is embedded within each PE's implementation and does not concern the operation of the ILC object.

The layer membership of each PE is regarded as a system administrator's configuration problem resolved dynamically during the system's startup time. The Startup object is responsible for the initialisation of the ILC entity and indirectly of each PE within a local system.

Since all PEs are message driven, a timer source generated message becomes just another message type received and processed asynchronously by each PE (if required). Communication with the TimerEvtSrc (figure 12) object instance allows all correspondents to schedule their own time events optionally.

The network operator (the Operator object in figure 12) generates unsolicited messages at randomly determined times, allowing local node network management issues to be incorporated within the ILC model.

The main advantage of developing an ILC model as we suggest lies in the fact that it separates local operating system concerns from a PE's operation, thus allowing the PE's implementation to be independent of the available communications hardware and operating system features.

From the discussion above, it should be clear that the ILC is represented as an object which operates as an operating system's kernel recognised preemptable process acting as a real-time traffic controller providing the following:

(i) buffering/queuing of messages for inter-layer and external networking environment communication,

(ii) detachment of PEs from the local operating system's environment,

(iii) a form of inter-task (inter-process) communication facility among all PEs,

(iv) control of each PE's execution state.

4. An overview of a PE

Again, in what follows in this section, should be treated as a condensed description of the problem statement.

By comparison with the description given in section 3, we require that each PE is represented as an object which operates as a non-preemptable process which:

(i) executes concurrently with other PEs,

(ii) synchronises with other PEs only through exchange of messages,

(iii) processes messages one at a time, in the atomic fashion,

(iv) has access to a timer and buffer management functions,

(v) communicates with its environment via generic send(srb)/recv(srb) messages,

(vi) does not concern itself with the local operating system's dependencies,

(vii) coordinates its execution only with ILC.

We will only briefly mention that designing a PE involves some specific protocol analysis and that requires a number of additional steps to be taken such as:

(i) the analysis of the English language protocol's procedural description,

(ii) operational description of the protocol in the form of:

- timing diagrams illustrating all/special cases of message exchanges,

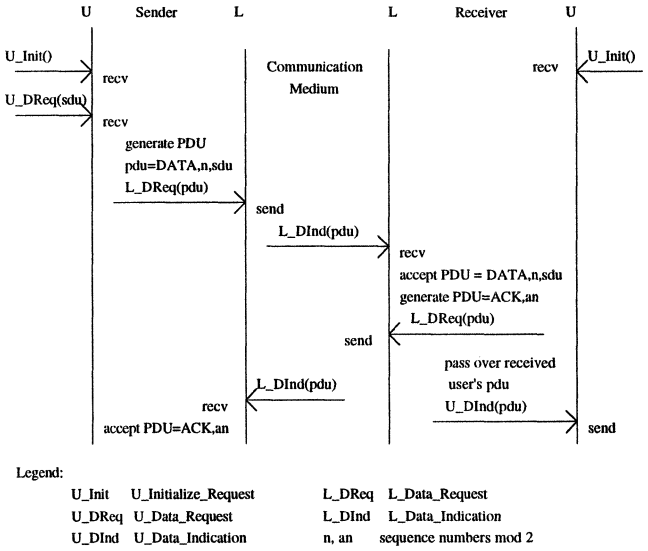

Figure 3. ABP's description of service boundaries.

- pseudo-code or SDL (or some other FDT based) protocol description,
- specification of PDU formats (usually given in ASN.1 notation).

To illustrate the generic structure of a PE, we consider the ABP protocol [4]. Figure 3 shows only the service boundaries and assumed PDUs.

We map the PE/ABP object into the ILC model structure (figures 12–14) and determine its internal composition so it allows the derivation of re-useable classes. The basic set of re-useable model elements suitable for any type of PE is given in figure 4. Note that FSM control aspects are not shown because the structure for the entire PE has been broken down into a number of state machines embedded within individual objects. We view a PE as essentially consisting of a number of generic objects selected through the modelling process to suit the particular protocol specification. Since we do not have a ready-made catalogue of re-useable parts, we have to build a set of generalised classes using analysis and decomposition. For example, an object called ServiceAcceptor is an abstract class. Its specialisation derived specifically for ABP is called AbpServiceAcceptor. So, the derived class structure could be as follows:

ServiceAcceptor,
  ABPServiceAcceptor,
  (and possibly other service acceptors).

There are other re-useable parts defined as abstract classes such as the Initialiser, Transporter, Distributor and ProtocolUtility. Each of these classes will have a number of specialisations tailored to the specific requirements of various protocols.

We view the basic PE functionality essentially as initialisation and data transfer. All other functions are selectable during the modelling stages. For instance, the Connector handles establishment and maintenance of connections.

We regard connection handling as part of a higher-level abstraction compared with connectionless communication.

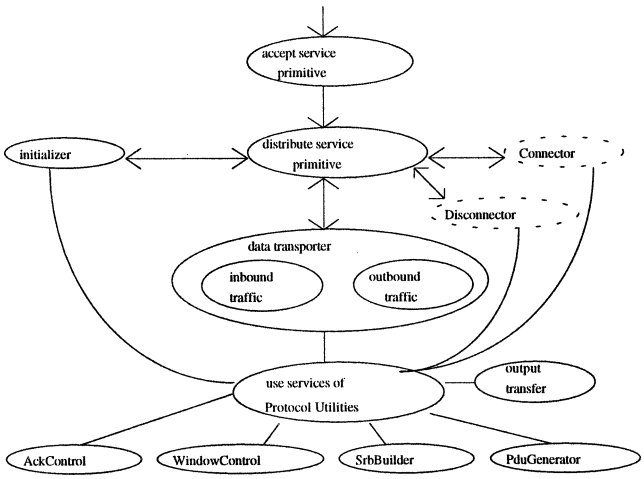

Figure 4. PE's internal structure.

Hence, the basic functionality is implemented via a connectionless service (or, in other cases, a single connection). AckControl is an optional element treated as one of the utilities a protocol might use. Other utilities used by every protocol are SrbBuilder and PduGenerator. The output transfer's functionality can be structured in a number of ways. In our implementation example, the OutQ handling is the individual PE's responsibility (figures 13 and 14).

5. Modelling method

The OO modelling suggested by the authors such as Rumbaugh [32], Booch [7] and Coad/Yourdon [10] does not emphasise enough the dynamic aspect of modelling, treating it merely as an extension of static (data) modelling and utilising a similar methodology. The latest UML version of Rational Corp. [29] tries to alleviate this problem The approach taken by Jacobson [20] with case modelling gained some acceptance because it emphasises the dynamics of message exchanges. We suggest a further expansion of OO modelling and analysis by showing a top-down, compositional development of Petri Nets (PNs) with message sequence encoding and applying it to the networking protocol domain. We use PNs as suggested by Reisig [30,31] not for traditional verification purposes, but as a generalised modelling and design tool for the derivation of the generic class structure which may be used as the basis for implementation. The use of a PN exposes in a very natural manner possible parallelism and inter-dependencies. This is particularly appealing within the OO paradigm. Our PN approach to modelling service primitive handling within a PE (figure 15), is based on the work of Bourget [8].

Message sequence encoding is another subject which we explore in the context of OO protocol modelling [11]. We have taken the existing ISO TTCN standard [17], modifying it slightly to serve our needs not for testing but for the general purpose description of message sequencing. It can be used effectively within the OO paradigm for design, implementation, automatic test case generation, simulation

and testing of OO models. In addition, an object cataloguing practice can be enhanced by the tabular TTCN based format.

5.1. Multiple views of objects

It is useful to be able to clearly distinguish in the modelling process how objects are viewed and manipulated. We use the following four planes of viewing objects.

5.1.1. User/application plane

Objects are perceived by their functionality/data manipulation capabilities and a modeller/designer develops a judgement on how well they fit together. The latter is a designer's view on object interface compatibility. For the time being, we will use a pragmatic approach in defining 'the fit' between two objects very much in the style of the work of electronic engineers. When trying to match two or more ICs, they evaluate the 'closeness of the fit' by comparing the description of two interfaces (usually pin description) from a catalogue of parts called a Data Book, and then work around those interfaces possibly adding some components 'in-between' to make the interfaces work properly. This plane is the top and most abstract design level for object-oriented implementations where a designer takes pre-fabricated objects and matches their interfaces through analysis of problem domain requirements and a catalogue of software parts. Incompatibilities may then be removed by, for instance, introducing new, more specialised subclasses or, intermediate objects might be added. The user level views may be complemented by the use-case style of analysis [20].

5.1.2. Model plane

Entire systems are modelled and object classes are derived in this plane. We use CPNets (Jensen [22]) as a tool for modelling, debugging, simulation and verification. CPNets represent an abstract view of behaviour of the modelled system and its individual parts. A translation of model elements from the User Plane view (a catalogue of parts) onto CPNet view is provided usually in the form of appropriate part labelling system. Dynamics of model element interactions are encoded by CPNs with support of the TTCN notation. CPNets represent a dynamic, behavioural model of object classes (on a meta-level), controlling the execution of the underlying object model.

5.1.3. Formal plane

This represents a translation of CPNets into a mathematical formalism with a suitably chosen set of rules for manipulation of CPNet representations.

5.1.4. Implementation plane

It deals with low-level implementation details of a formally defined model. Usually a number of components are involved such as implementation of CPNets simulator, graphical elements manipulator, verification tool for CP-Nets.

Table 1
A fragment of word analysis table.

Word/Name/Phrase	Related functionality
Startup	One of the correspondents, PE configuration process, initialises ILC and all PEs
Inter-layer communication entity ILC	Provides an SRB delivery service among all correspondents
Protocol entity PE	Absorbs SRBs from input queue performs protocol specific process sends SRBs to output queue
Service request block SRB	A uniform message format exchanged among ILC and all

Table 2
A fragment of the candidate object classes table.

A candidate object class	Characterised by
Startup	PE configuration process, initialises ILC and all PEs
Messenger	Manages InQ and OutQ, controls the execution of each PE
PEntity	Has its own identity(address), executes a protocol specific process,
ServiceRBlk	SRB is a uniform message format exchanged among all correspondents

5.2. Static modelling

The derivation process of static/class structure of a model is well developed with many available methodologies (e.g., [7,10,20,29,32]). Using the graphical and methodological suggestions of Dillon and Tan [11], the derived diagrams for the ILC model indicating the class structure are given in figures 5 and 6.

We found that modelling tasks are easier to perform when they are enhanced by following a tabular format.

For example, in MT1, a modeller not only studies the natural language description or some other form of problem statement but also greatly enhances his/her own understanding when presenting selection of relevant words or names in the tabular form (see table 1).

Note that in table 1 we would normally list all relevant words or phrases without suggestion as whether they represent a good name for a class or not.

Another example (see table 2) of the use of tabular representation refers to MT2 when after the problem domain analysis a designer has to choose an initial (and still subject to many revisions) set of classes.

Table 2 prepares for MT7 where the detailed object class's internal data structures are defined as shown in table 3.

Finally in MT3, during the static phase modelling, a designer must arrive at a representation of essential relationships among candidate object classes. We follow sugges-

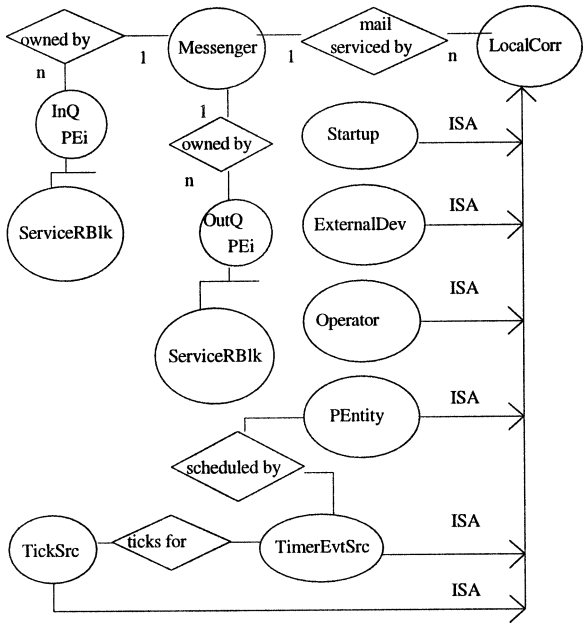

Figure 5. ILC – non hierarchical relationships.

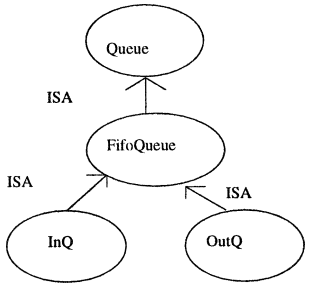

Figure 6. ILC – Queue class hierarchy.

Table 3
A table fragment showing internal class object's static structure.

Class name	Class data	Instance data	Comments
Messenger		operState, corrPriorityList, queueList	
PEntity		peAddr, operationalState	layer: peaddr active/suspend
ServiceRBlk		serviceType, peSrcAddr, peDstAddr, dataRef, serviceParams	ref to PDU

tions made in [11] but representations derived from other methodologies may as well be easily included.

It is often desirable at a certain level of abstraction to make a distinction between a component and a part treated as a single entity. To avoid confusion, we use the phrase "a model element" for both cases. Model elements identified during the modelling process become abstract building blocks that are used later in constructing software applications. Modelling tasks presented in subsequent sections

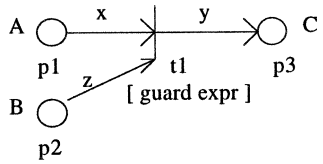

Figure 7. An example of CPNet.

assume as a starting point an understanding of the structure of the model which was developed in the static modelling phase. Some intuitive understanding of the model's functionality should also be developed during the static modelling phase, as it is difficult to justify an object's existence without appreciating to some degree its functionality.

No assumption is made that static modelling has finalised the model's structure in any way. On the contrary, the structure is open to further modifications and improvements during dynamic modelling. As dynamic modelling reveals more details to the model builder, often new elements are retro-fitted in the static structure or existing elements are removed.

Symbols used in figures 5–7 have the following meaning:

- a diamond shape denotes a non-hierarchical relationship,
- a circle/oval shape denotes an object class,
- a horizontal bar with one or more vertical branches denotes composition relationship (is-part-of, belongs-to),
- an arrow labelled with ISA denotes an ISA relationship (specialisation to generalisation).

5.3. Petri Net drawings

Dillon and Tan [11] advocate a particular class of high level nets, called State Controlled Petri Nets (SCPN) as a mechanism for modelling the dynamics of OO systems. The SCPNs are a special case of Coloured Petri Nets of Jensen [22]. By a simple example we only give the feel for the formalism used in Colour Petri Nets (see figure 7).

Transition's $t1$ activation (firing) is constrained by the guard expression. Each place is a depository of tokens which are arbitrary data types. The token type is determined by its colour set (here A, B, C are defined elsewhere). The arc expressions could involve conditional, inter-arc dependent token selection, e.g., z could be replaced by "if x is odd then 1`b else 3`b" where type(b) $= B$. Places will normally have multiple occurrences of individuals of the same type. Hence, 3`b means: take three 'b' individuals of type B. Variables x, y, z have to be bound to some tokens of the appropriate type. In general, arc expressions have to evaluate to a token(s) of the corresponding place's type/colour set. In most of our examples in this paper we omit many of the elements of CPNets to emphasise the modelling principles when using CPNets. The traditional control token (black dot) is regarded as a colourless token.

For the purposes of this discussion, tokens are arbitrary messages in the communication sense rather than object-oriented messages. These tokens are treated as objects, i.e.,

class structured data types. Firing of transitions triggers some procedure activation which is executed in an atomic fashion.

While deriving PN drawings, the compositional rule in the top-down approach is enforced. Both places and transitions may be further detailed through refinement or embedding. We initially develop a PN net drawing concentrating on the model itself (i.e., its functionality). Typically, places and transitions are replaced by more detailed nets up to a certain abstraction level. Throughout the process, immediate annotations and labels for every transition and every place must be made.

Sometimes the appropriate name for a place is not apparent, but transitions are usually well defined and can be labelled easily. When there is difficulty in labelling one or more transitions, one has to look at the entire model developed so far and assess informally its correctness. Very often a transition which we think 'must be' at some spot in the drawing and for which we cannot find a proper description indicates that either something is missing in the model, or our understanding of what is required of that model is insufficient.

5.4. Model element description

Model elements can be characterised as

model element = virtual wiring + behaviour + data

- **model element** could be a composite object, or a simple, atomic part object – both have their own methods and data.
- **virtual wiring** means a passive interface description, a set of all possible messages plus optional parameters at each designated point of interaction.
- **behaviour** is described by a set of all possible sequences of messages for every given point of interaction organised into a set of dialogue cases (encoded using a modified TTCN notation).
- **data** consist of a set of references to other objects describing the object's state.

Our treatment of the object's method interface needs some clarification. From figure 8, a subset of methods in Object X accesses Object Z's methods 'a' and 'b'. Similarly, a subset of methods in Object Y accesses three methods 'b', 'c' and 'd' in Object Z. Hence, while in theory the interface of Z (methods a, b, c, d) is visible and accessible to both Object X and Object Y, in practice only Object Z's methods 'a' and 'b' are visible to Object X and methods 'b', 'c' and 'd' are visible to Object Y. It is useful when carrying out dynamic modelling to capture this 'limited' visibility aspect of object interactions because of the need to map a subset of available methods associated with an object to a PN place.

Those different interactions of objects can be modelled through what we call points of interactions and an object could have more then one point of interaction with other

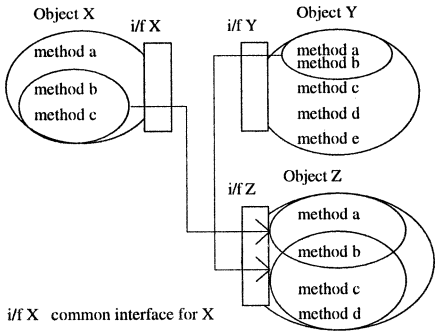

Figure 8. Interfaces and method visibility.

objects. Points of interaction are labels identifying correspondents. They are conceptually similar to Points of Control and Observation (PCOs) [17]. In terms of networking protocol requirements, a PCO may coincide with a Service Access Point (SAP) identifier. In our OO protocol modelling exercise, we treat SAP addressing as a totally different issue, pertinent only to the internal protocol operation (or protocol entity addressing). As illustrated in section 5.6, a point of interaction is mapped onto a Petri Net place of some description. The consequence of distinguishing multiple PCOs within an object is their limited external visibility. This feature already exists in OO environments where typically the object's methods are grouped into private/internal or externally visible methods. Furthermore, for reasons of convenience, the externally visible methods are also grouped into specialist categories. That last feature of categorisation does not usually impair the external accessibility of methods. The main purpose of these remarks is to identify the use of methods against individual PCOs/correspondents ensuring the compositional/additive effect of dialogue structure/behaviour associated with each model element.

We view model elements as communicating agents and often refer to them as correspondents. In fact, it is our conviction that the OO paradigm can meet the challenge of building an OO system as a simulation model of a network of communicating correspondents structured in some specific way (including both class and communication/message exchange structuring). The strong suggestion here is that models of communication protocols should be built in an OO fashion according to the communication principles which they themselves embody.

5.5. Specification of points of interaction

Once a PN has been refined to a sufficiently detailed level (it is the modeller's decision where to stop the refinement process based on the requirements specification of the model) then the virtual wiring for the model elements is designed. We propose a table format as in figure 9 adhering in appearance to the TTCN conventions. Note that we distinguish all individual points of interaction accessible to a model element. At least one such point exists for every element namely, the universal interface open to

Table 4
A fragment of class-activities table.

Class name	Related activity
Startup	initialise Messenger, configure a PEntity, initialise a PEntity
Messenger	activate a PE, suspend a PE, deliver an SRB/process send–recv
PEntity	send an SRB, receive an SRB, request a timer event, cancel a timer event
InQ	dequeue an SRB enqueue an SRB
TimerEvtSrc	schedule timer event, cancel timer event

any correspondent. Typically, there will be an 'internal' point of interaction as well that allows methods within an object to invoke other private object's methods. Points of interaction are defined directly from the PN drawing as specifically named places. The example in figure 9 indicates one or more method names associated with each point of interaction. The list of methods is subject to further improvements.

5.6. Model element's behaviour encoding

Based on a thorough understanding of class definitions and their static relationships within the problem domain, the designer, in the modelling step MT4, is required to develop a good grasp of each class's functionality. Again, as table 4 illustrates, this is done best in the tabular form. It should be clear that the MT4 step already indicates what methods are applicable for each of the classes. During the previous modelling steps each class's functionality was informally discussed but here we register the initial set of applicable methods in preparation for the step MT6.

After completing MT4, a designer develops a CPNet representation/drawings (see section 5.7) in succession of refinements and/or embeddings and then encodes message sequences. The basis for this modelling task is the final, unfolded PN drawing for a model element. A simple mapping takes place:

- each model element's labelled place is mapped into a TTCN PCO identifier,
- in general, a labelled transition is mapped into a method.

The mapping of transitions requires a note of caution. During the construction of method sequences, it is not always obvious whether we refer to another object's method/functionality or whether we need another method name within the currently activated object. To locate the object membership of a given method correctly, we need a good grasp of the static model's features (particularly class structuring) and their purpose.

We use the following amendments to the TTCN notation (figure 9):

1. Each table identifies a **Model Element** and the **Reference** at the top. For cataloguing purposes, one might add some form of a Petri Net Drawing identification associated with the element's behavioural or virtual wiring tables.
2. The **Verdict** part in TTCN tables is omitted here.
3. We have retained the **Constraints** part with the constraints stated explicitly for easy visual verification. The TTCN notation uses as a supporting tool the ASN.1 notation which in our case could be very convenient for the design and implementation phases.
4. We have added one pseudo-event called RETURN to verify objects returned by synchronous communication based on the Call-a-method/RETURN-an-object convention existing in most of the OO execution environments.
5. TTCN does not cater for concurrent processing. There is no 'execute in parallel' type operator. We contend that model elements executing concurrently and communicating asynchronously should be documented in such a way that their role in the model is not ambiguous. Usually some appropriate description of additional constraints pertaining to the model element's role as an independently operating agent is sufficient. We utilise TTCN notational convention by encoding concurrent interfaces as alternatives. The operational mode should be clear from the PN drawing and the additional documentation. Alternative solutions should be investigated in the future. Usually, independently operating agents will have the FOREVER label shown and referred to in the TTCN style behavioural table.
6. Labels denoting points of interaction are explicitly stated in front of "!" and "?" event designations and those are followed by the method names. There is only one generic 'recv' method. The internal 'send' method always converts the private 'send' into the destination object's 'recv'. Should the receiving object require additional checking of the sender's identity, such information is available through the current execution context of the receiving object.
7. At the end of each behavioural table we make a check list of the externally visible events/methods, as opposed to the ones used internally by the object's methods.
8. TTCN test cases are event trees with parameters passed to each tree as required. We relaxed that treatment since there is no need to use parameters in the same way. Typically the notation "! recv(srb)" describes an activation of the 'recv' method with the reference to a ServiceRBlk class instance passed in 'srb'. When required, this notation could be supplemented with additional detail in the constraints column. For example, "srb.type = U_Data_Req" indicates the compulsory value within the srb object instance reference.

Model Element: NetProcess

Reference : Local-Operating-System-Environment/NetProcess

Identifier : NetProcess-Wiring

Purpose : Operates within the local o/s environment, iterfaces directly via o/s kernel with device drivers, real-time clock and operator console. Initialized once from the Startup process.

Point of interaction	Correspondent	Direction	Message	Params	Comments
PCO Label	Object Name	in / out	Method Name	Object Refs	Text
dev-in	ExternalDev	in	devMsg	iorb	o/s specific
dev-out	ExternalDev	out	devMsg	iorb	i/o request block
tick-in	TickSrc	in	realTick		RT interrput driven
oper-in	Operator	in	operMsg	string	buffer contents
oper-out	Operator	out	operMsg	string	
init-in	Startup	in	initMailer	param	parameters to be
init-out	Startup	out	configPEntity	param	determined
			ackInitMailer	param	
			ackConfigPEntity	param	
			initError	reason	

Extended Comments All points of interaction are local operating system specific. There is a need to translate/transform externally generated messages into a uniform format (ServiceRBlk) used by this model element. Note that srb refers to ServiceRBlk and iorb is associated with the kernel's i/o operation.

Figure 9. Interfaces and method visibility.

5.7. An example of PN based modelling

Only some steps of the gradual, top-down approach using Petri Nets are given here. Dotted elliptic/rectangular shapes are place model elements and transition model elements respectively with their refinements shown within. Black-edged rectangles usually show the environmental boundaries. We start from some general notion of NetProcess interacting with a communications device (figure 10). Both are a part of the local operating system environment. In general, we treat places and transitions as objects in our model. Places are usually the passive acceptors of messages and transitions are active elements which transform messages and produce new ones. Note that we distinguish between messages flowing through places and methods (also messages in OO parlance).

The main rule applied in this process is that places are replaced by S-sets and transitions are replaced by T-sets, as described in Reisig [30,31]. After some initial refinements we arrive at the view presented in figure 11. Note that although the main model object is called NetProcess, the interaction points labelled as 'DeviceServ-in' and 'DeviceServ-out' are PCOs of both the NetProcess and ExternalDev objects, the latter being external to our model. Hence, for each model element we always include its externally defined points of interaction because they are a part of its virtual wiring. There are more elements interacting with the NetProcess object then shown in figure 11. From the problem domain description we know that they all are independent processes. After a few more refinements, we arrive at the ILC model view shown in figure 12 (the

Model Element : DeviceServ-in

Reference : NetProcess/DeviceServ-in

Identifier : DeviceServ-in-Behaviour

Purpose : Absorbs device specific messages from ExternalDevice and translates them into ServiceRBlk.

Behaviour Description	Label	Constraints	Comments
TTCN syntax		Explicity stated	Text
dev-in ? devmsg(iorb) internal ! translateToSrb(iorb) ? RETURN(srb) mail-in ! send(srb) -> FOREVER ? RETURN(nil) -> FOREVER	FOREVER	error free iorb	

Extended Comments :

External events	Internal events
devmsg(iorb)	translateToSrb(iorb) send(srb)

Figure 10. An example of behaviour encoding.

NetProcess object). In this dynamic model enhancement, we retain as much as possible of those model elements derived in the previous static phases [14]. Each individual element has the same refinement procedure applied to it recursively until a sufficient level of detail is achieved to warrant unambiguous design and implementation decisions.

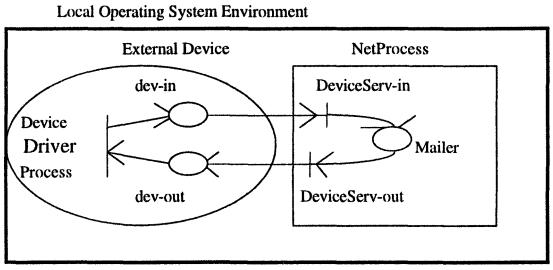

Figure 11. An initial top level PN.

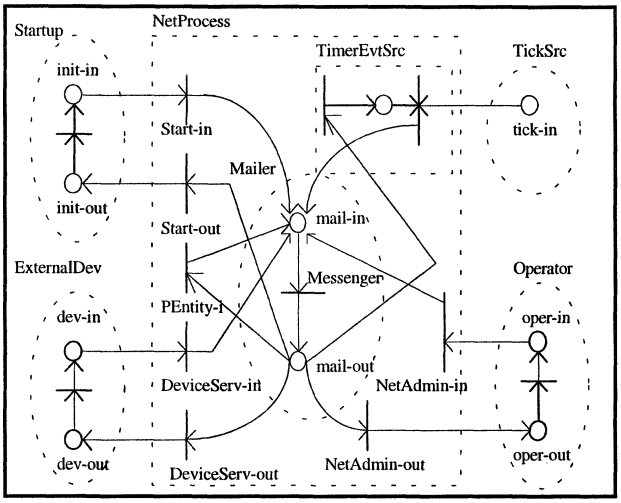

Figure 12. A Petri-Net representation (Channel-Agency variety) of the ILC model.

As the top-down PN structure allows for multiple levels of control abstraction, it also allows related abstractions for message passing. The latter are maintained through virtual wiring and interface specs.

Note that each of the detailed nets will have appropriate documentation created for it in the form of the virtual wiring (figure 9) and behavioural tables (figure 10). This forms the basis of a catalogue of parts created by the modelling process. In our implementation, we have introduced a hierarchical system of identifiers for each drawing and table (not shown here), reflecting the top-down PN structure.

Taking as a starting point figure 12 and then following through figures 13–15 it is easy to see how a composition of individual PNets has been achieved. The examples given represent only a selected subset of objects for illustrative purposes. Note also how interfaces (places) are labelled to achieve 'connectivity' with other objects.

The TTCN style message/method sequencing follows naturally from a PNet representation. For instance, compare figure 13 with figure 16.

6. Dynamically adjustable ARQ

6.1. Introduction

Various Automatic Repeat Request (ARQ) strategies are implemented to suit different protocol requirements in many

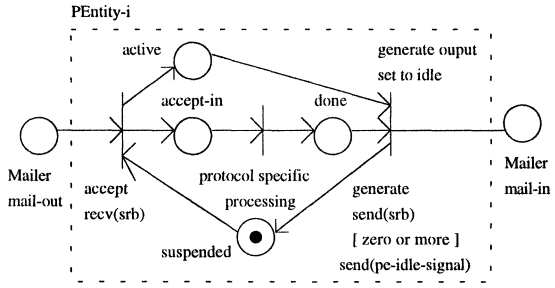

Figure 13. The top level control of a PE.

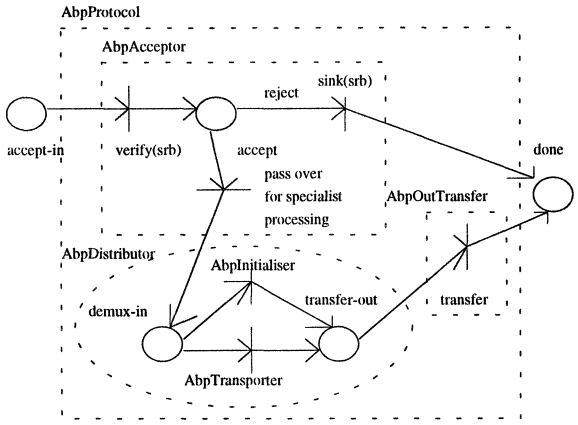

Figure 14. PE/ABP internal structure.

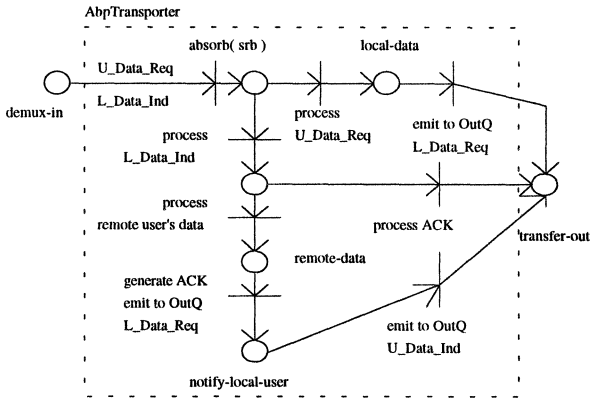

Figure 15. ABP service primitive processing.

layers. Typically, they exist in the Data Link layer. They may even be a part of an end-user written database query application program utilising IP interface system calls. Here we assume the following: a Data Link (DL) protocol entity (PE) provides services to its user (a Network Layer) protocol entity utilising services of a Physical Layer (PHL) protocol entity through well-defined interface points called Service Access Points (SAPs).

The ARQ error handling strategies utilise either stop-and-wait, selective repeat or continuous transmission regimes. There are a number of continuous transmission ARQ variants described in the technical literature, each with its own throughput efficiency characteristics making it suitable only for some types of data exchange [6,33,38]. Most of them assume that the transmission channel error

Model Element : PEntity-i

Reference : Local-Operating-System-Environment/NetProcess/PEntity-i

Identifier : PEntity-Behaviour

Purpose : This description applies to all protocol entities.
They execute concurrently but process only one message at a time.

Behaviour Description	Label	Constraints	Comments
TTCN syntax		Explicity stated	Text
mail-out ? recv(srb) [operational_state = SUSPENDED] (operational_state := ACTIVE) + ENTITY-CONTROL + ENTITY-IDLE (operational_state := SUSPENDED) -> FOREVER [operational_state <> SUSPENDED] internal ! sink(srb) -> FOREVER ENTITY-CONTROL accept-in ! abpProcess(srb) done ? generateSends(srbList) ENTITY-IDLE mail-in ! send(srb)	FOREVER	srb.type := pe-idle-signal	critical region critical region

Extended Comments :
Treat ENTITY-CONTROL and ENTITY-IDLE subtrees as internal methods.
abpProcess(srb) invokes a protocol specific FSM functionality within the AbpProtocol object.
srbList represents all messages generated by AbpProtocol during one activation

External events	Internal events
recv(srb)	send(srb)
abpProcess(srb)	entityControl(srb)
	generateSends(srbList)
	sink(srb)
	entityIdle

Figure 16. An example of PE's message sequencing.

rate does not change considerably over a period of time. However, in circumstances such as those existing in satellite or cellular radio communication systems, transmission of data is highly volatile with randomly changing error rates and long propagation delays. These conditions require the adoption of a flexible, dynamically adjustable ARQ strategy such as the one described in Yao [38].

Here, the major development phases of the ARQ handler prototype are documented. It assumes a continuous, dynamically modifiable ARQ strategy following the algorithm described in Yao [38]. Two particular algorithms are utilised, the basic Go-Back-N [33] and Birrell's Preemptive Retransmission [6] (also called n-Copy).

Due to space limitations, we will show only some aspects of the proposed methodological approach.

6.2. Object model of the ARQHandler

The ARQHandler may be treated as an object based reuseable component with well-defined interfaces. The general structure of a PE (see figure 4) requires a component element which we call the DataTransporter. It handles inbound and outbound traffic within a PE and calls upon services of more specialised object components (see figure 17).

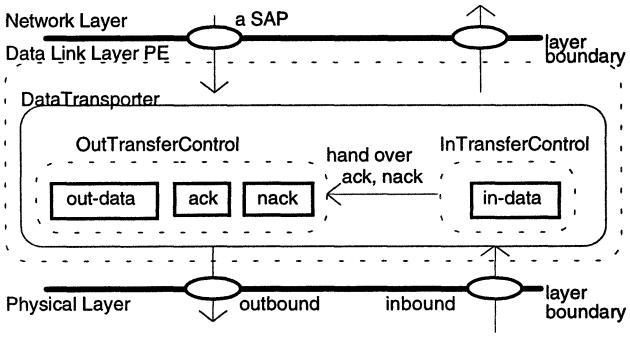

Figure 17. A partial view of a PE's major components.

For our purposes, it is sufficient to distinguish messages containing 'data', 'ack' or 'nack'. These messages are carried in objects called ServiceRequestBlocks (SRB). An SRB represents an implementation of an abstract service primitive. SRBs contain all the required information about the service which a PE provides to its user (another PE in the layer above). Normally, a PE would handle traffic in both directions (to and from the external network),

thus implementing sender and receiver elements. Initially, we concentrate on a sender handling the outbound traffic and responses received from the remote PE via the feedback channel. How physically the channel transmission is implemented (full-duplex, simplex channel(s)) is not important. Also, other parts of a PE's design such as connection management and the proper DL protocol operation (such as in LAPB, LAPD) are omitted.

6.3. Some assumptions

The main purpose here is the derivation of the ARQ handler component which is the major part of the OutTransferControl. It might be the only part with some interfacing code. The message type labelled as 'out-data' in figure 17 represents a number of octets which are received and have no special meaning on this level. This data is handed over by some other software components (not shown) for delivery to the remote receiver.

It is important to note that the various ARQ algorithms assume the same operational mode of the remote receiver. This is the basis for implementation of the adaptive ARQ schemes such as the one described in [38]. Also, when the ARQ strategy adjustment takes place (the ARQ algorithm switch is totally transparent) the acknowledgment queue (AckQ) contents remain intact. The newly chosen ARQ algorithm continues processing out-data, nacks and acks as before the switch.

All (re-)transmissions, error checking and acknowledgment handling are performed internally by the ARQ handler component according to the available various ARQ strategies. These strategies are chosen dynamically depending on the inferred channel error rate. In consequence, the objects implementing the low error rate or the high error rate handling algorithms may be executing at any time instance. Following suggestions in [38], these two objects implement respectively the Go-Back-N [33] and n-Copy [6] ARQ algorithms. Furthermore, it is assumed that the DL protocol data units (PDUs) are well-defined data structures recognised by both the receiver and sender. All sequence numbers are used modulo N, where N is some number agreed by the sender and receiver. The CRC or some other error checking algorithm is used. Moreover, for the sender's side, the round trip propagation delay, the window size, the transmission repeat factor, the maximum number of octets which can be sent in one transmission attempt over the chosen communication channel (max DL PDU size), the threshold values of Alpha (for counting Nacks) and Beta (for counting Acks), and a timeout value are all suitably chosen.

6.4. Dynamic ARQ selection algorithm

The ARQ strategy is selected based on constant monitoring of successful (Ack) and unsuccessful (Nack) transmission attempts and inferring on that basis the current channel error state [38]. There are two constants which are arbitrarily chosen, Alpha and Beta, and a counter AckCount.

Table 5
A list of candidate objects and their attributes.

Class name	Attributes	Comments
ArqHandler		Communicates with AckQ and AckControl, responsible for window control, (re-)transmissions, message sequence number generation, timeout handling and timer source handling
	MaxWindowSize	
	currentWindowSize	
	SeqNumModulus	
	assignNextId	Id for the next data message
	RepeatMsg	For n-Copy (re-)transmissions
	ackControlRef	A pointer
	ackQRef	A pointer
AckControl		Responsible for interpretation of Acks and Nacks, reconfig., i.e., switching between the two ARQ strategy's, execution of ARQ algorithms in respect to the AckQ contents
	Alpha	
	Beta	
	ackCount	
	nextAckId	Next expected sequence number
	arqHandlerRef	A pointer
	ackQRef	A pointer
AckQ		Communicates with ArqHandler and AckControl, responsible for generation of QItem objects and storing them in chronological order, queue maint. (additions, deletions of QItems), some ARQ strategy dependent specialised functions
	queueLength	The optimal queue organisatio is the implementation time decision
	queueRef	The actual queue pointer
QItem	id	The sequence number
	timoutCount	
	status	AckPending, Retransmit
	RepeatMsg	
	outDataRef	A pointer

The sender may be in either the low error rate (L) or high error rate (H) state. In the L state the Nacks are counted and compared with the Alpha value. In the H state, the Acks are counted and compared with Beta value. When the threshold value is reached, the counter is cleared and the transition to the other state takes place with the appropriate adjustment of the sender's ARQ strategy algorithm.

6.5. Conceptual analysis

The problem description analysis, supported by the study of representative research papers, leads to the list of candidate classes and their attributes given in table 5.

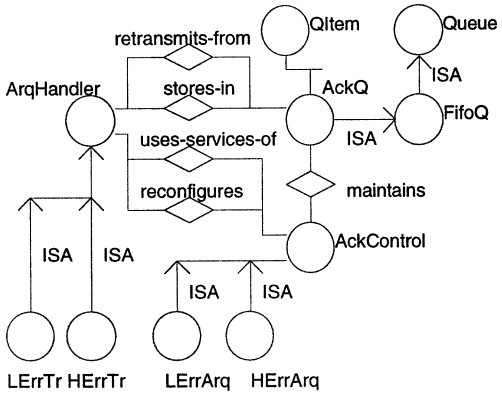

Figure 18. ArqHandler's static relationships.

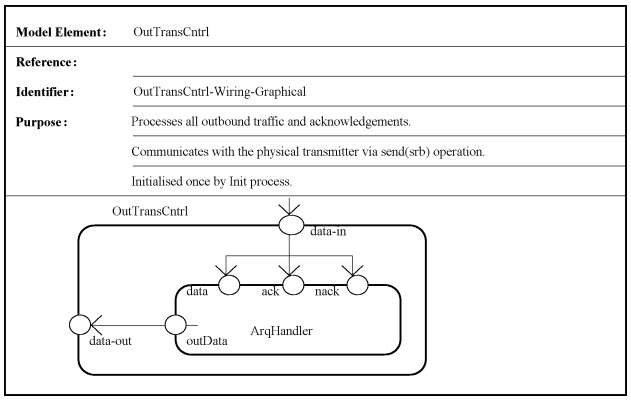

Figure 19. Top level component virtual wiring.

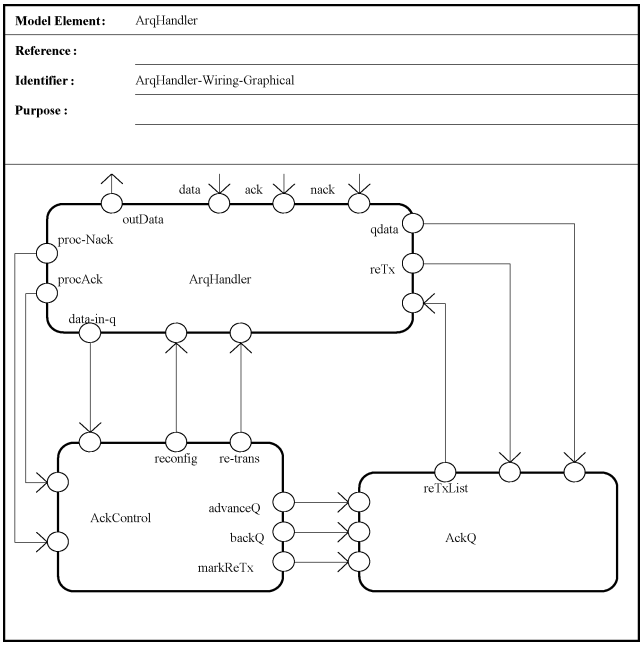

Figure 20. ArqHandler virtual wiring.

6.6. Essential relationships

Note that LErrTr and HErrTr facilitate the ARQ strategy specific data transfer. They differ only in the initialisation routines. Otherwise, all functionality is provided by the common super-class ArqHandler. Similarly, LerrArq and HerrArq differ in acknowledgment treatment for different ARQ strategies. A further requirement is a transparent switch between (LErrTr, LErrArq) and (HErrTr, HErrArq) instances of object class pairs.

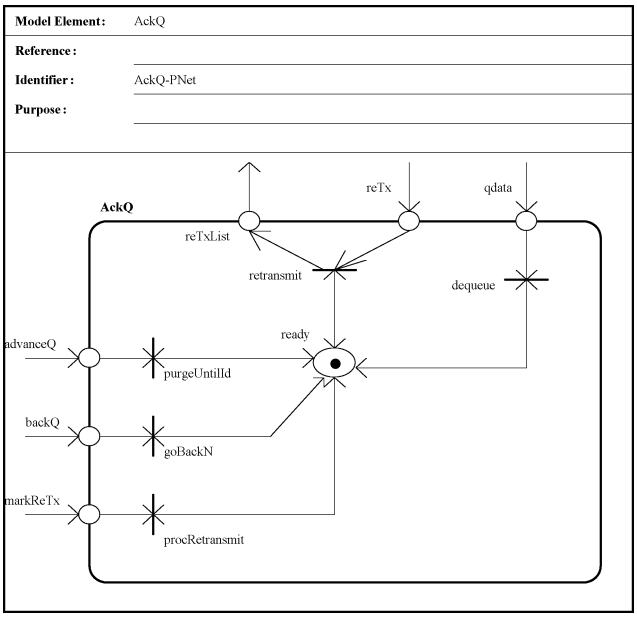

Figure 21. AckQ CPNet representation.

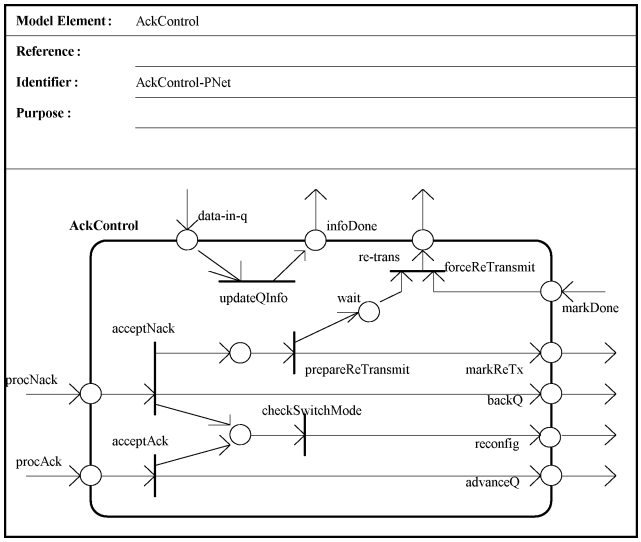

Figure 22. AckControl CPNet representation.

6.7. Inter-object virtual wiring

By creating virtual wiring documentation such as that shown in figure 9, it may be modelled as in figures 19 and 20. Note that the labelled points of interactions will become (in the next design stage) a part of Petri Net representation (see refinements in figures 21–23).

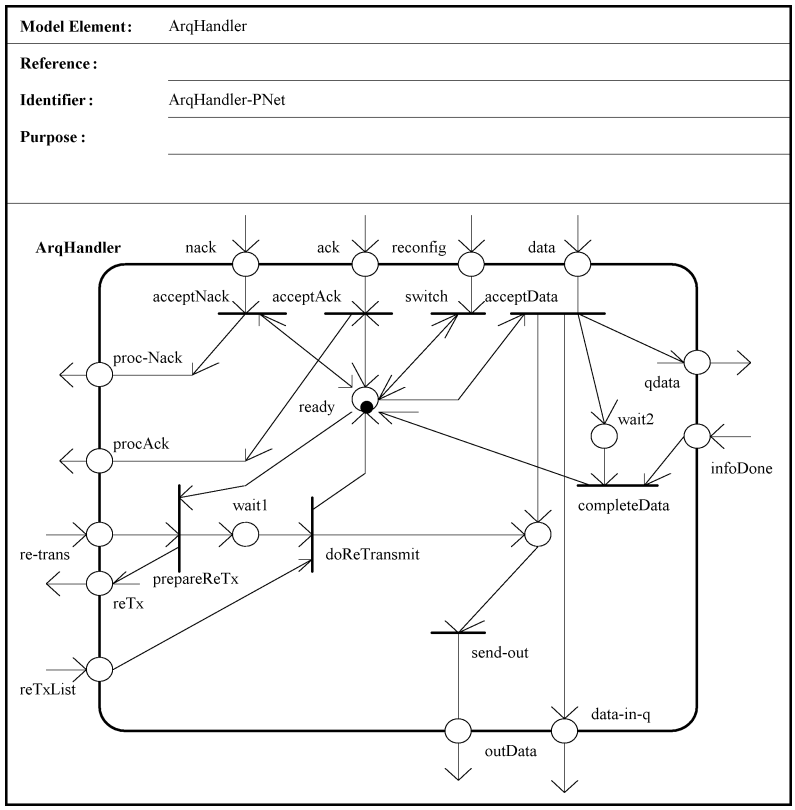

Figure 23. ArqHandler CPNet representation.

6.8. A CPNet model element representation

Each model element can now be treated on its own merits and a suitable Petri Net can be drawn, giving a new design perspective on each individual object's behaviour. Note that CPNets become quite simple in the context of individual object class definitions. Points of interactions become places and external virtual wiring can be interpreted as a fusion of places, connecting specified Petri (Sub-)Nets that correspond to various object classes. Alternatively, by convention we omit (to avoid cluttering) PN transition bars between any two connected POIs. Consider figure 20 as an example. Three objects ArqHandler, AckControl, and AckQ can be abstracted to PN places that are connected through appropriate transitions (omitted in figure 20 by convention) between corresponding places (POIs). Thus, the ArqHandler's virtual wiring of figure 20 represents a valid Petri net (as a meta-level control structure). Omitted transition bars represent generic message transfer (send/receive) between the POIs of the appropriate correspondents. Also, note that after the CPNet specification stage, the initial object virtual wiring might be augmented with some additional points of interactions (e.g., 'infoDone') due to new requirements forcing a designer to synchronise some of the events (method execution). In ArqHandler, we have enforced the rule of processing one message at a time. It is always instructive to analyse created CPNets from the point of view of concurrent executions. Note that some transitions might produce a null result (without generating an output token).

The full ArqHandler prototype documentation is lengthy (in excess of 80 pages). The analysis of virtual wiring of objects and the derivation of CPNet representations provides a new way of looking at any object's behavioural and other supporting functionality. In our prototype, the achieved economy of function use was striking (6–10 methods per object class). CPNets provide another plane of object behaviour specification, a meta-level control of the underlying object model.

The design of the ARQ protocol indicated above, illustrates the power of the methodology for design of networking software of the sort found in mobile networks.

7. Conclusions

Dynamic interactions in OO representations have always posed difficulties for system modellers and designers. These difficulties are even more strongly emphasised when modelling and designing networking protocols. It is therefore important to develop a good mechanism for modelling the object interactions. We focused on the way a protocol implementation can be derived through the OO modelling process involving both static and dynamic features.

In this paper, we have shown how the use of a very general form of Petri Nets can serve the purpose of dynamic object modelling and detailed message sequence derivation in a compositional fashion. The method lends itself easily to the specification of re-useable model elements.

By defining objects, their properties and interactions, one obtains a good understanding of the ramifications of protocol implementation within particular environments or within a particular software tool. During the dynamic modelling stage the introduction of new object classes may be required.

The advantages of the use of Petri Nets are well understood and do not need restating. The TTCN notation has proven to be an excellent tool for precise message sequence encoding mainly due to its simplicity, visual appeal and well understood semantics. It also allows for the incorporation of automated design tools. Both Petri Nets and TTCN encoding can be used for verification and testing as well as for modelling, design and implementation of networking software of the sort found in mobile networks.

Acknowledgements

The authors wish to thank reviewers for constructive criticism and Professor Kia Makki for his advice and encouragement in producing the extended and updated version of [16].

References

[1] M.B. Abbott and L.L. Peterson, A language-based approach to protocol implementation, IEEE/ACM Transactions on Networking 1(1) (February 1993).

[2] C.C. Ang, V. Jordan and T.S. Dillon, Application of Petri nets to specify and verify the ISO CASE protocols, in: *Proc. Second International Conference on Interoperable Systems*, Tokyo (November 1988).

[3] S. Bapat, *Object-Oriented Networks* (Prentice-Hall, Englewood Cliffs, NJ, 1994).

[4] K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson, A note on reliable full-duplex transmission over half duplex links, CACM 12(5) (May 1969) 260–261.

[5] E. Berutto et al., Radio protocol architecture of the CODIT UMTS system, in: *Mobile Communications – Advanced Systems and Components*, ed. C.G. Gunther, Lecture Notes in Computer Science 783 (Springer, Berlin, March 1994) pp. 417–427.

[6] N.D. Birrell, Pre-emptive retransmission for communication over noisy channels, IEE Proc. Part F 6 (November 1981) 393–400.

[7] G. Booch, *Object-Oriented Analysis and Design with Applications* (Benjamin/Cummings, 2nd ed., 1994).

[8] A. Bourget, A Petri net tool for service validation in protocol, PSTV-VI, IFIP (1987) 281–292.

[9] D. Box, D.C. Schmidt and T. Suda, ADAPTIVE – An object-oriented framework for flexible and adaptive communication protocols, in: *High Performance Networking*, Vol. IV, eds. A. Danthine and O. Spaniol (North-Holland, IFIP, 1992) pp. 367–382.

[10] P. Coad and E. Yourdon, *Object-Oriented Analysis* (Prentice-Hall, Yourdon Press, 2nd ed., 1991).

[11] T.S. Dillon and P.L. Tan, *Object-Oriented Conceptual Modelling* (Prentice-Hall, Englewood Clifs, NJ, 1993).

[12] F. Halsall, *Data Communications, Computer Networks and OSI* (Addison-Wesley, Reading, MA, 2nd ed., 1988).

[13] A. Hanish, Operating systems and communication protocols, in: *Proc. Int. Workshop on Object Orientation in Operating Systems (IWOOOS '95)*, IEEE Computer Society, August 14–15, Lund, Sweden (1995).

[14] A. Hanish and T. Dillon, Object-oriented modelling of communication protocols for re-use, Part I – Static modelling aspects, Dept. Comp. Sci. and Comp. Eng., TR 2/95, La Trobe University, Melbourne, Australia (1995).

[15] A. Hanish and T. Dillon, Object-oriented modelling of communication protocols for re-use, Part II – Dynamic modelling aspects, Dept. Comp. Sci. and Comp. Eng., TR 3/95, La Trobe University, Melbourne, Australia (1995).

[16] A. Hanish and T. Dillon, Object-oriented modelling of communication protocols for re-use, in: *Proc. Fourth Int. Conf. on Computer Communications and Networks*, Las Vegas (1995).

[17] ISO/IEC, TTCN, DIS 9646–3.

[18] ISO/IS, Basic Reference Model, ISO/IS 7498 (1984).

[19] B. Jabbari et al., Network issues for wireless communications, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 902–906.

[20] I. Jacobson, *Object-Oriented Software Engineering* (Addison-Wesley, Reading, MA, 1993).

[21] F.J. Jaimes-Romero et al., Modelling handoff and dynamic channel allocation using Petri nets, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 876–881.

[22] K. Jensen, *Coloured Petri Nets*, Vols. 1 and 2 (Springer, Berlin, 1991 and 1995).

[23] J.-P. Katoen, Functional integration of UMTS and B-ISDN, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 160–164.

[24] C.D. Keen and C.A. Lakos, A methodology for the construction of simulation models using object-oriented Petri nets, in: *Proc. European Simulation Multiconference* (1993).

[25] N.K. Liu, Formal description and verification of expert systems, Ph.D. Thesis, La Trobe University, Melbourne, Australia (1991).

[26] G.Y. Liu and G.Q. Maguire Jr, Efficient mobility management support for wireless data services, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 902–906.

[27] J. Munemori, T. Mizuno and S. Takeda, An extension of SDL for the executable specification of communication systems based upon Petri nets, INTAP, ACM (1988) 111–118.

[28] K. Pahlavan and A. Falsafi, Trends in local wireless data networks, in: *Proc. IEEE 46th Vehicular Technology Conf.*, Atlanta, GA (April 1996) pp. 21–25.

[29] Rational Software Corp., *UML-Summary, UML-Notation Guide, UML-For Real-Time Systems Design*.

[30] W. Reisig, *Petri Nets in Software Engineering*, Lecture Notes in Computer Science 255 (Springer, Berlin, 1986).

[31] W. Reisig, *A Primer in Petri Net Design* (Springer, Berlin, 1992).

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

[33] M. Schwartz, *Telecommunication Networks: Protocols, Modelling and Analysis* (Addison-Wesley, Reading, MA, 1987) Chapter 4 pp. 119–134.

[34] B. Selic, G. Gullekson and P.T. Ward, *Real-Time Object Oriented Modelling* (Wiley, New York, 1994).

[35] M.S.K. Sushko, Advanced data services for wireless communication networks, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 331–335.

[36] F.J. Symonds, Modelling and analysis of communication protocols using numerical Petri nets, Ph.D. Thesis, University of Essex (May 1978).

[37] C. Vissers, FDTs for open distributed systems, A retrospective and a prospective view, in: *PSTV-X*, eds. L. Logrippo, R. Probert and H. Ural (North-Holland, IFIP, 1990) pp. 341–362.

[38] Y.-D. Yao, A Go-Back-N ARQ scheme for mobile satellite communications under variable channel shadowing conditions, in: *Proc. IEEE 45th Vehicular Technology Conf.*, Chicago, IL (July 1995) pp. 341–345.

Andrew Hanish has a background in mathematics (M.Sc., Pedagogical Institute of Opole, Poland) and in computer science (Grad. Dipl. Comp. Sci., Monash University, Australia). He has worked in the computer industry for many years as a consultant and a software developer, specializing in design and implementation of communications software. In 1992, he joined the Department of Computer Science at La Trobe University in Melbourne, Australia, as a lecturer. Currently he is completing his doctoral thesis at La Trobe University dealing with applications of Petri nets and object-oriented design methodologies to network protocol modelling and design. His main areas of interest are object-oriented modelling, applications of Petri nets, computer network protocols and their specification, validation and testing, network performance evaluation, and simulation methods for network protocols and distributed systems. He is a member of ACM and IEEE.

Tharam Dillon received a Bachelor of Engineering (Honours) in 1967 and Ph.D. in 1974 from Monash University, Melbourne, Australia. He was appointed to the inaugural Chair of Computer Science at La Trobe University in Melbourne, Australia, in 1985. He is currently Professor of Computer Science and Computer Engineering at La Trobe University in Melbourne, Australia, and Director of the Applied Computing Research Institute. A major area of his research involves object-oriented systems and software engineering. One of the distinguishing features of his software engineering research has been its application to communication protocols. He has also carried out research in the automated knowledge acquisition area related to decision trees, theoretical issues, and neural network-based methods for extracting symbolic knowledge. His other research interests include database and knowledge-based systems, and hybrid systems that combine neural, fuzzy and symbolic paradigms. He has published over 320 papers in international and national journals and conferences and written three books and edited four other books. His authored books are Khosla, R., and Dillon, T.S., 'Engineering Intelligent Hybrid Multi-Agent Systems', Kluwer Academic Publishers, 1997; Sestito, S., and Dillon, T.S., 'Automated Knowledge Acquisition', Prentice-Hall, Sydney, 1994; and Dillon, T.S., and Tan, P.L., 'Object-Oriented Conceptual Modeling', Prentice-Hall, Sydney, 1993. He has also worked with industry and commerce in developing systems for banking and finance, power systems, telecommunications, and health care systems. He is editor-in-chief of the International Journal of Computer Systems Science and Engineering and the International Journal of Engineering Intelligent Systems, as well as co-editor of the Journal of Electric Power and Energy Systems and an associate editor of IEEE Transactions on Neural Networks. He has served on the program committee of many international conferences, at times as chairperson. He is a member of several international technical committees and working groups and is the convenor of the International CIGRE Task Force 38-06-02 on Expert Systems for Monitoring and Alarm Processing in Control Centres. He is also a member of the IFIP Working Group 2.6 on Knowledge and Data Semantics. He is a Fellow of the Institution of Engineers (Australia), Fellow of the Safety and Reliability Society (UK) and a Senior Member of IEEE (USA).