Chapter 9

Summary of Conclusions

This chapter summarizes the conclusions of this thesis. We categorize the conclusions by their relation to design quality criteria, state-of-the-art description and evaluation, the design framework, the design model, and the reference architecture.

Design quality criteria

The design of application protocols should be guided by design quality criteria. Although effectiveness and efficiency (cost/performance ratio) are important desirable qualities of application protocol realizations, no criteria related to these qualities can be formulated at an architectural level. Criteria related to ‘ease of use’, however, can be formulated at this level. Application of these criteria leads to ‘clean’ designs, which are found to be the right basis for deriving effective and efficient realizations. The criteria have been characterized by the particular aspects of ease of use they pursue: consistency, orthogonality, propriety, and generality.

State-of-the-art description and evaluation: OSI upper layer architecture and model

The OSI reference model identifies three layers of application protocol functionality. The two lower layers, viz. the Session and the Presentation Layer, are involved in the support of all distributed applications. The highest layer, the Application Layer, consists of a collection of Application Service Elements (ASEs) and Control Functions (CFs). Depending on the distributed application that must be supported, different compositions of ASEs and CFs can be selected with the Association Control Service Element (ACSE). The ACSE is a special ASE that itself is always involved in the support of distributed applications.

This structure and the assignment of functions to entities in this structure have been criticized on basis of design quality criteria. The most important criticisms are:

- the OSI upper layer architecture (OSI-ULA) is a static architecture which is not suitable for all distributed applications. The flexibility of the Application Layer should
apply to the OSI-ULA as a whole, i.e. no static protocol layer hierarchy should be enforced;

• transfer syntax constraints (enforcing a common encoding for PDUs) should not be assigned to a ‘central’ entity, such as the Presentation Layer in the OSI-ULA. Each distinct application protocol should define abstract syntax constraints (enforcing valid PDU formats) and the corresponding transfer syntax constraints.

• segmentation/reassembly and concatenation/separation should be performed by the Transport Layer and not, as is currently the case, by the Session Layer;

• the (re)synchronization functions of the Session Layer are fine examples of political compromises that lead to poor technical design solutions.

Some of these deficiencies are due to the poor definition of service concepts and of Application Layer concepts. As a consequence, service decomposition methods are not defined, and protocols are not designed on basis of required services, with proper consideration of quality design criteria. Adequate design methods and design concepts can help to reduce the, currently long, development times of application protocol standards.

Despite the quality deficiencies, derived application protocol realizations are not necessarily inefficient. If an implementation is made of the combined functionality of the Session, Presentation, and Application Layer protocol entities, the implementor has considerable freedom to restructure functions, and to optimize local protocol processing. On the other hand, the process of restructuring adds complexity to the implementation phase.

Design framework

A design can be structured with different intentions or for different reasons. Two important structuring domains have been distinguished:

• behaviour domain: in this domain, behaviour and behaviour structure is defined. Relevant primitive concepts in this domain are action, interaction, and causality relation; and

• entity domain: in this domain, entities are defined with associated behaviour. Relevant primitive concepts in this domain are action point and interaction point.

Furthermore, three related abstraction levels have been identified that support distributed system design:

• interaction system perspective: abstracts form the individual responsibility of interacting entities;

• integrated system perspective: abstracts from the internal behaviour (implementation) of an entity;
• distributed system perspective: defines the internal behaviour of an entity in terms of interacting sub-entities.

Based on these abstraction levels, an application protocol design trajectory has been defined. This design trajectory is a simplified model of the application protocol design process, with design milestones that run from an enterprise service to an application protocol architecture.

**Design model**

Compositions of the primitive concepts of action, interaction and causality relation can be used to represent arbitrarily complex behaviours. Two behaviour structuring techniques have been presented based on composition rules with respect to these concepts:

• causality-oriented composition: behaviour components are related through the coupling of entry and exit points, forming causality relations between the components;

• constraint-oriented composition: behaviour components are related through contributions to shared interactions.

Different types of behaviour decomposition and refinement have been identified that support the design steps in the application protocol design trajectory:

• causality-oriented decomposition: decomposes an integrated behaviour such that a causality-oriented composition of that behaviour results;

• constraint-oriented decomposition: decomposes an integrated behaviour such that a constraint-oriented composition of that behaviour results;

• causality refinement: introduces new actions, but preserves the actions of the given behaviour; and

• action refinement: replaces (some) actions of the given behaviour by activities, where activities are compositions of actions;

These manipulations can in general not be automated, because of the design freedom involved. However, it is possible to verify whether a resulting behaviour is a correct decomposition/refinement of the given behaviour.

**Reference architecture**

The main purpose of a reference architecture is to help the designer in choosing a suitable architecture for his specific design, and to incorporate pre-defined building blocks. An application protocol reference architecture must be flexible enough to cope with the diverse requirements of distributed applications. It should therefore not define a single, static architecture, but it should allow the designer to select a suitable architecture from a set of possible architectures.
A flexible application protocol reference architecture has been derived by considering the design decisions concerning the structure of a design in the application protocol design trajectory. The top level structure of a distributed application consists of a layer of local processing functions (information processing entities), a layer of user-defined application protocols, and a layer of standardized application protocols. User-defined application protocols can use standardized application protocols as pre-defined building blocks.

Application protocols can be further structured in terms of protocol layers, and protocol layers in terms of protocol sections. The structuring techniques used here are constraint-oriented composition and causality-oriented composition. The number of protocol layers, and the choice and composition of protocol sections depends on the class of distributed applications that must be supported, and should therefore be decided by the designer. The concept of protocol sections also allows to integrate the layer of user-defined application protocols and the layer of standardized application protocols: a single application protocol layer can in principle be composed from user-defined protocol sections and standardized protocols sections.

Two design methods have been presented that support the development of protocol layers and protocol sections.

Some application protocol sections that can be used as very general application protocol building blocks have been identified and characterized: association establishment, inter-stream synchronization, intra-stream synchronization, activity synchronization, commitment control, and dialogue control.