Protocol Reconfiguration using Component-based Design

Fotis Foukalas[1], Yiorgos Ntarladimas[2], Aristotelis Glentis[3], Zachos Boufidis[4]

[1] Communications Network Laboratory, Department of Informatics and Telecommunications, University of Athens, Ilisia Campus 157 84,
Athens, Greece
{foukalas, yiorgos, arisg, boufidis}@di.uoa.gr
http://cnl.di.uoa.gr

Abstract. Previous modular protocol design and implementation allow a flexible configuration and reconfiguration of protocol layers or full protocol stacks. However, in our days, software engineering technologies introduce new methods for designing and specifying modular software. Such a technology is the component-based software technology. Using those techniques, a software system could be modular. This paper proposed a reconfigurable protocol design and specification approach as well as a protocol reconfiguration management/runtime model based on protocol components that represent distinct protocol functions, which in previous works have been designed and specified as modules. The following content could be considered as a suggestion for a UML profile for protocol components and protocol reconfiguration.

1 Introduction

This document is an attempt to introduce a UML profile for protocol components and how protocol reconfiguration can be achieved using component-based software design methods. The objective of this paper is the definition of the protocol component data model and the protocol component management model for reconfiguration purposes. The protocol component data model is the preclusive step that defines the stereotypes for modeling, specifying and implementing protocols as components. It is obvious that a protocol reconfiguration presupposes a reconfigurable way to design and specify communication protocols. Considering components as modular unit versus modules, we avoid the probability that a component will not include its manifestation in order to be replaced within its environment. The new component definition and specification, defined in UML 2.0, permits the component concept to be used to describe component designs or implementations, without losing the ability to describe deployment information [4].

In addition, the component-based design is one of the key enabling technologies for designing reconfigurable software. This design method supports the compositional adaptation approach of software systems that can be modified dynamically in terms of its structure and behavior [7]. The compositional adaptation of a software system is not just enabled for tuning software variables, like the parameter adaptation approach, but in contrast, enables the dynamic recomposition of the software during execution, for example to switch software components in and out of memory, or to add new behavior to deployed systems. Component-based design supports two types of composition the static composition and the dynamic composition. The second composition method is relied on late binding programming technique, which enables coupling of compatible components at runtime. It may be remarked that the dynamic composition is the only way in which a software engineer can add, remove or reconfigure components at runtime, although this method's flexibility can be constrained in order to ensure the system's integrity across adaptation.

The intention of this paper is to introduce the concept of protocol components and their reconfiguration that depend on the components and composite structures packages as well as the state machines package from infrastructure and superstructure specification of UML 2.0 [4][5]. In this direction, some steps for the component-based protocol design and specification have been identified [8]. Moreover, components for the physical and data link layer have been introduced by OMG for radio communications [9]. Related works also have introduced modular approaches for the design and implementation of protocol functions [1][10]. Moreover, an architecture that allows dividing protocol function into components using component-based software engineering and particularly the EJB Component Model has been addressed [11]. Considering the above-mentioned works and also the UML 2.0 specification, we are introducing the UML extension for protocol design and specification as well as reconfiguration management using the component classifier. A component-based design approach is the way to achieve a dynamic reconfiguration of communication protocols.

The rest of this document is organized as follows: In Section 2, we introduce the protocol component concept and definition and also a case study for implementing a protocol function as protocol component. Section 3 describes the protocol reconfiguration management model introducing the protocol component reconfiguration concept and also a user model for a MOF layer 0 protocol reconfiguration management model. In this section also is introduced the protocol manager as the entity for managing the protocol reconfiguration process. Section 4 discusses the protocol stack issues and how a stack can be constructed using protocol components and what are the requirements for a protocol stack reconfiguration. In addition, some future perspectives of our work are addressed. The last section of this paper is devoted to summary and conclusions.

2 Protocol functions as components

2.1 Protocol Component definition

The idea for mapping protocol functions into modular software units has already introduced [1][3][10]. Protocol functions have already recognized as much as possible generic like the flow control, error control, segmentation/reassembly that represent well-known protocol functions. However, these generic protocol functions consist of more concrete protocol functions. For instance, error control protocol function consists of error correction, error detection and retransmission protocol functions as well as flow control consists of window-based and rate-based protocol functions [2]. Moreover, using different protocol mechanisms can specify protocol functions. Protocol mechanisms are like traditional protocol specifications: they have a unique name and they define the rules governing data transfer from one service user to its peer and handling of payload and control information [1]. On the other hand, in UML 2.0 the component concept has been clarified to be more definite and meaningful throughout the development life cycle. The UML 2.0 specification defines the component as a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment [4][5]. Considering all the above features of communication protocols as well as the capabilities of component to be modulated, we have designed the following diagram (Fig. 1) that defines the protocol component concept.

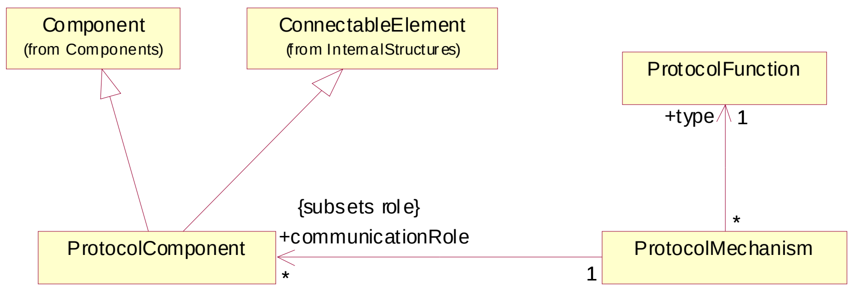

Fig. 1. Protocol Component definition

It has been recognized that a protocol mechanism consists of a sending and a receiving part, e.g., segmentation and reassembly, or the sending and receiving part of automatic repeat request (ARQ) [1]. *ProtocolMechanism* references protocol components, which can represent two definite roles the sender and/or receiver that instances play in this protocol mechanism. Therefore, the *ProtocolComponent* has a communication role in a protocol mechanism. Each *ProtocolComponent* represents the sender or the receiver role in a communication protocol mechanism. A protocol component can implement only one protocol mechanism and a protocol mechanism can be implemented by different protocol components. An example of protocol mechanism and protocol component association is given in the following paragraph.

Moreover, each protocol mechanism specifies only one protocol function and on the other hand each protocol function could be specified by several protocol mechanisms. The *type* association role defines the type of protocol function that the protocol mechanism specifies. Regarding the parents of *ProtocolComponent*, have been selected the Component from Components package and the ConnectableElement from InternalStructures package [4]. The parent Component indicates a protocol component as a kind of component since we have already envisioned a protocol component as a software unit that can be replaced within its environment. The parent ConnectableElement expresses a protocol component as a kind of connectable element that can be communicated with another instance through a network connection or something more simple as a pointer in case that the protocol component connects with another protocol component in the same node to form a protocol component stack. The following subsection gives an example of protocol mechanism specification using the extension in UML 2.0 for protocol components.

2.2 Protocol Component: A case study

In Fig.1 is illustrated the protocol components data model. The protocol components data model defines the basic stereotypes needed for the protocol design and specification using component-based software engineering methods and their associations with UML 2.0. In case that a protocol engineer needs to design and specify a protocol using protocol components, a question will be raised. What is the granularity of protocol functions and protocol mechanisms? It has been already recognized that the finer the granularity of protocol functions, the higher the number of different configurations and the higher the flexibility [1]. For instance, using a final granular approach could be specified an error detection protocol function that is a fine granular form of error control protocol function. In sequence, the error detection protocol function can be specified by a feedback error control mechanism, which finally can be implemented by a protocol component instance [2]. An example of the protocol component model for the fine granular protocol function error detection is depicted in the following figure (Fig. 2a). However, for a full specification of protocol mechanism a collaboration diagram is needed in order to depict the communication role of the protocol component. The Fig. 2b depicts the collaboration diagram of error detection protocol mechanism and the roles of the protocol component instances. It is useful to denote that a protocol component could be developed implementing both the sending and receiving part [1]. In our case the *fdback_EC_S* protocol component implements only the sending part.

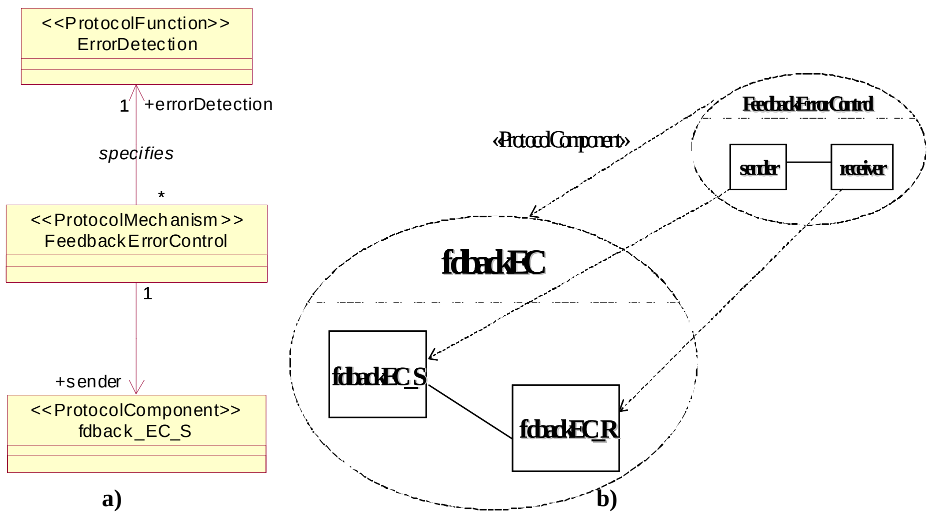

Fig. 2. a) Protocol Component Model for the error detection protocol function, - b) Protocol Mechanism Collaboration Diagram

3 Protocol Components Management Model

3.1 Protocol Reconfiguration Management/Runtime Model

Protocol reconfiguration has been defined as the replacement of a component by another component that implements the same protocol mechanism [1]. According to 3GPP definition, reconfiguration is the rearrangement of the parts, hardware and/or software that make up the 3G network [15]. From the software engineering side, dynamic reconfiguration means the ability of code to be introduced at runtime [7]. Considering all the above definitions we define the dynamic reconfiguration as the process of the replacement/rearrangement of system parts which indeed must be performed at runtime. Taking into account this definition we are introducing the following diagram (Fig. 3) that represents the protocol reconfiguration management/runtime model. The management model is an active runtime entity that is dealing with (managing) data. The base data has been introduced in the previous section.
The context of the protocol component has been defined as the runtime entity that is going to be forwarded to the new component. The *ProtocolComponentContext* constitutes the reconfiguration of the *ProtocolComponent*. The association between the protocol component context and the component itself is 1:n since a protocol compo-

nent context can be forwarded in numerous protocol components that implement the same protocol mechanism. The protocol component context is composed by protocol control information, actually, the header of the particular protocol data unit and the current state of the protocol component that is running. The *ProtocolComponentState* is a kind of *ProtocolStateMachine* of StateMachines package [4]. Furthermore, in order to provide the ability for a protocol reconfiguration in a protocol software environment, we have introduced the interface *ReconfigurableProtocolComponent*, which provides all these operation for reconfiguration management. Through this interface a *ProtocolManager* can manage the reconfiguration process.

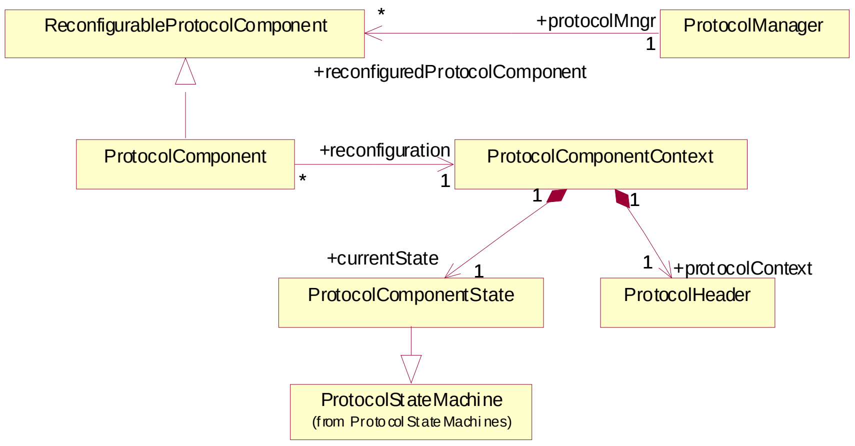

Fig. 3. Protocol Reconfiguration Management/Runtime Model

An example of protocol reconfiguration management user model, according to MOF layers, using the protocol components stereotypes and the protocol reconfiguration management model could be the following diagram (Fig. 4). This is based on dynamic protocol configuration and reconfiguration concept that has been introduced in document [1]. The *Ifdback_EC_S* is the unified interface of the protocol component provided for reconfiguration purposes. The operations of this interface enable to get and set the protocol context in order to reconfigure the protocol. There is also an *init* operation for the initialization of the protocol component with the appropriate protocol component context. Moreover, the interface for reconfiguration provides an operation for retrieving protocol component statistics for example the number of detected errors in a receiving CRC module. This last operation is provided for monitoring purposes and not for the realization of protocol component performance. At the moment, we are not going into details for protocol manager capabilities to select protocol components. Nevertheless, the following text encompasses a brief list of protocol manager capabilities in order to support the reconfiguration process.

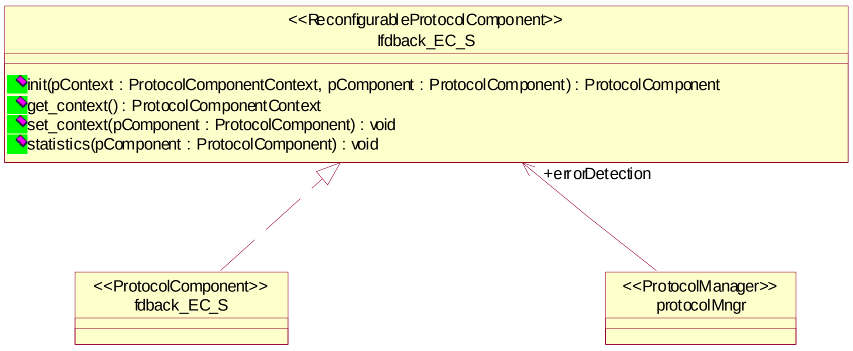
<<ReconfigurableProtocolComponent>>
Ifdback_EC_S
init(pContext : ProtocolComponentContext, pComponent : ProtocolComponent) : ProtocolComponent
get_context() : ProtocolComponentContext
set_context(pComponent : ProtocolComponent) : void
statistics(pComponent : ProtocolComponent) : void
+errorDetection
<<ProtocolComponent>>
fdback_EC_S
<<ProtocolManager>>
protocolMngr

Fig. 4. Reconfigurable Protocol Component

3.2 The Protocol Manager

The need for introducing the protocol manager has been raised since an entity for the following management procedures is necessary:

- A protocol manager must provide an efficient runtime environment for protocol configuration and reconfiguration.
- A protocol manager must link, initialize, and release protocol components, synchronize parallel components, and forward packets within stack of protocol components.
- A protocol manager must monitor the properties of protocol components
- A protocol manager must be enabled to realize protocol performance in order to decide the substitution between protocol components implemented the same protocol mechanism.
- A protocol manager must be enabled to map the requirements for protocol stack reconfiguration in the appropriate protocol selection and composition

However, the above-mentioned functionality is not an exhaustive list of protocol manager functionality. In addition, the protocol manager is envisioned as the entity that provides the time and storage management of the needs of a protocol component instance. The protocol manager will be also the entity that can parse the protocol component graph of a particular protocol stack. In the following section, we are discussing the requirements for a protocol component composition and selection and what they represent for a protocol reconfiguration.

4 Discussion

4.1 Requirements for Protocol Stack Reconfiguration

In most cases, the need for a reconfiguration process, in a modular software system, is being satisfied according to application requirements. In our case, a modular software system is the protocol stack of reconfigurable equipment [13]. A protocol stack reconfiguration has been considered as a reconfiguration process, which reconfigures a protocol or the stack as a whole, according to application requirements. The protocol stack is considered as protocol component graph that is composed by protocol components. Each protocol component implements a specific protocol mechanism required for the satisfaction of application requirements [1]. Application requirements represent the *required QoS* needed for running the appropriate application. The QoS requirements have to be mapped into protocol components properties for the selection of protocol components. The result of this mapping should be a reconfigured protocol stack providing the satisfaction of the application requirements.

However, a protocol stack reconfiguration requirements can be also surfaced from the properties offered by lower layers. For instance, a mobile device's protocol stack must be reconfigured in order to operate in a different radio technology. In such a case, the terminal can change its radio interface that includes properties. These properties represent the *offered QoS* and express the radio interface requirements. Some of the required radio-specific characteristics has already identified and indeed specified [12].

Therefore, considering the above mentioned reconfiguration scenarios we have recognized two dimension requirements; requirements that represent the required and offered QoS respectively (Fig.5).

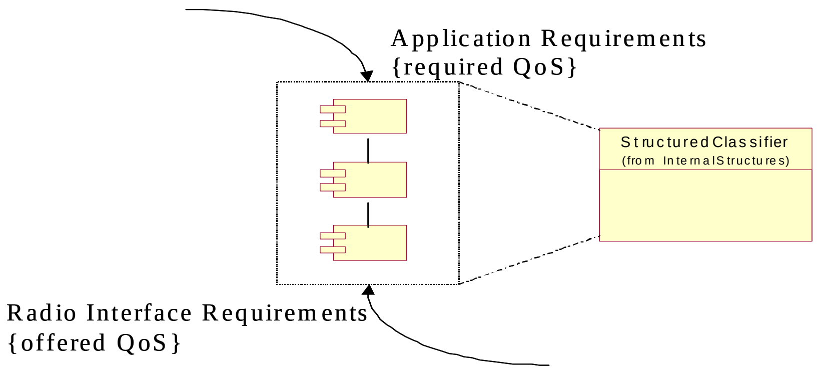

Fig. 3. Requirements for Protocol Stack reconfiguration

Using the definitions of several OMG's specification for components, we can define the requirement and property for protocol reconfiguration management. Requirement is the desired feature requested by component implementation [6]. Property represents a set of protocol component instances that are owned by a containing classifier instance [4]. In our case the containing classifier is the protocol stack that contains protocol components. The parent of protocol stack has been considered the structured classifier from the *InternalStructures* subpackage since the structured classifier provides the mechanisms for specifying structures of interconnected elements that are created within an instance of a containing classifier. Those interconnected elements are the protocol components that constitute the protocol stack as well as play the role of the protocol function that protocol components implement and a protocol mechanism specifies.

4.2 Future work

In the previous subsection, we have envisioned and defined the protocol stack concept composed by protocol components. In this direction, we are considering the protocol component selection and composition according to radio interface and/or application requirements. The selection of the appropriate components according to requirements is enrolled in reconfigurable computing area. The requirements have to be mapped to the corresponding properties that encompass the criteria for the selection of protocol components in order a reconfiguration procedure takes place. We are working on this area and our future work will encompass the selection of protocol components according to radio interface requirements.

5 Summary – Conclusions

Many of the modular approaches for protocol software systems have been introduced the concept of module for protocol design, specification and implementation. However, the concept of software module does not include the deployment information. Including deployment information a software system can be dynamically recomposed, installed and updated. All these procedures have been considered as the reconfiguration procedures of reconfigurable equipments [13]. We are envisioning a reconfigurable protocol stack that can be dynamically recomposed, installed and updated and for this reason we have introduced the protocol component concept.

References

1. Thomas Peter Plagemann “A Framework for Dynamic Protocol Configuration”, PhD Thesis, Swiss Federal Institute of Technology Zurich, 1994.

2. Gerald J. Holzmann, “Design and Validation of Computer Protocols”, Bell Laboratories, Prentice Hall, 1991.
3. L. Berlemann, A. Cassaigne, B. Walke, “Modular Link Layer Functions of a Generic Protocol Stack for Future Wireless Networks”, to appear in Proc. of *Software Defined Radio-Technical Conference*, Phoenix USA, November 2004.
4. Object Management Group. UML 2.0 Superstructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc /03-08-02.pdf (August 2003).
5. Object Management Group. UML 2.0 Infrastructure Specification: Final Adopted Specification. http://www.omg.org/docs/ptc /03-09-15.pdf (December 2003).
6. Object Management Group. “Deployment and Configuration of Component-based Distributed Applications”: Working Draft, http://www.omg.org/docs/ptc /04-05-15.pdf, 2002-2003.
7. Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, Betty H.C. Cheng , “Composing Adaptive Software”, Computer, v.37 n.7, p56-64, July 2004.
8. A. Alonistioti, F. Foukalas, N. Houssos, “Reconfigurability management issues for the support of flexible service provision and reconfigurable protocols”, Software Defined Radio Forum Technical Conference 2003 (SDR 2003), November 17-19, 2003, Orlando, Florida.
9. Object Management Group. “PIM and PSM for Software Radio Components”: Final Adopted Specification , dtc/04-05-04.
10. C. Tschudin, “Flexible Protocol Stacks”, Proc.ACM SIGCOMM ’91, Zurich, Switcherland, 1991, pp. 197-204.
11. Matthias Jung and Ernst W. Biersack, “A Component-Based Architecture for Software Communication Systems”, In Proceedings of IEEE ECBS, pages 36--44, Edinburgh, Scotland, April 2000.
12. Beyer, D.A.; Lewis, M.G., “A Packet Radio API”, Page(s): 1261-1265 vol.3.
13. “End-to-End Reconfigurability”, IP IST Project , http://e2r.motlabs.com/
14. Object Management Group. “CORBA Components ”: Working Draft, ptc/02-08-03, September 2002.
15. 3GPP TS 32600, “Telecommunication Management; Configuration Management (CM); Concept and high-level requirements;(Release 6)”.