International Conference on Software, Telecommunications and Computer Networks (SoftCOM)
October 07-10 2003, Split, Dubrovnik (Croatia), Ancona, Venice (Italy)

Using UML for the Design of Communication Protocols: The TCP Case Study

K. Thramboulidis, A. Mikroyannidis
Electrical & Computer Engineering
University of Patras, Greece
E-mail: thrambo@ee.upatras.gr, mikroyan@softeng.org

Abstract: **Object technology has been widely adopted in many application domains and the Unified Modeling Language (UML) has become the new industry standard for modeling software-intensive systems. UML is currently used for modeling of just about any type of applications, running on any type and combination of hardware, operating system, programming language and network. However, protocol design is still based on traditional methodologies. In the context of this paper, we present an approach that utilizes object technology and the UML notation for the development of communication protocols. A methodology is presented and extensions to the UML notation are proposed to address the peculiarities of protocol design. The construction of a TCP protocol for RTLinux was selected as an example to demonstrate the methodology. Design and implementation issues are presented and the resulting system is evaluated.**

1. INTRODUCTION

We have under development an Object-Oriented (OO) framework to implement the IEC61499 proposed architecture for open, interoperable and re-configurable distributed control application [2]. For the implementation of the CORFU interworking unit, which is used to interconnect the different types of field buses to a Fast switched-Ethernet, we selected RTLinux and we defined a modular architecture to satisfy the real-time constrains imposed by this kind of applications. The Industrial Process Control Protocol (IPCP) has been defined to satisfy real-time and non real-time requirements [3]. For the IPCP protocol to support commissioning, configuration and on-line re-configuration of control applications, TCP functionality was required. RTNet [4] was the only protocol stack that was found for RTLinux. It provides direct access to IP-based networking from RTLinux real-time code. RTNet's implementation is based on the standard Linux networking source code, with the necessary changes to make it real-time. RTNet implements the IP, ARP, UDP, and ICMP protocols over Ethernet but there is no implementation for the TCP protocol.

In [5] we have reported the use of RTNet to implement a prototype for the interconnection of a Profibus fieldbus with a Lonworks fieldbus. To satisfy the requirement for TCP functionality a first draft implementation of TCP, based on TCP Lean [6] was given. This TCP implementation was embedded in the RTNet module. In this paper we present an OO development for the TCP protocol layer that was carried out in order to provide a more robust, modular, expandable and layered TCP protocol stack for RTLinux.

Having applied the OO approach and the UML notation successfully in many application domains, we decided to exploit the advantages of this approach in the communication protocol domain. A survey on previous work that was carried out, highlighted the absence of significant progress in this direction. This is why we decided to apply our OO methodology. However, applying this methodology in this application domain, we found that it has to be adapted to address the peculiarities of protocol design. A number of extensions were introduced to satisfy the requirements imposed by communication protocol software.

The remainder of this paper is organized as follows: Section 2 briefly presents the main directions in protocol design. Section 3 outlines our modified methodology that can be used for the design of communication protocols. The design of a TCP layer for RTLinux is presented and implementation details are discussed in section 4. Finally, the proposed approach is discussed with more emphasis on performance evaluation and the paper is concluded.

2. PROTOCOL DESIGN

The current industrial practices in communication protocol design and implementation are unsatisfactory. There is a wide gap between state-of-the-art in Software Engineering and state of practice in communication protocols design and implementation. The development of communication protocols is still mainly based on the traditional procedural paradigm. The methodologies used for the development of protocols can be grouped in three categories. According to the first category that follows the procedural approach, a structured design methodology and a procedural language that is mainly C, are used for the development of protocol software. Even though this approach results in efficient implementations, reusability and flexibility are rather poor. The second category uses a formal description technique to create the protocol's specification, which is then translated into program code. SDL [7], Estell [8] and Lotos [9] are examples of specifications used, with SDL being the most widely adopted. Reusability is absent in the design phase and very limited in the implementation phase. The third category, which is continuously gaining ground, includes methodologies that are based on the OO paradigm. The OO approach results in implementations that exhibit increased modularity, flexibility, extensibility and reusability. Traditional tools used in protocol development, such as Conduits, have been expanded to exploit the benefits of the OO approach [10]. Successful results have been reported by researchers applying methodologies of this category. UML has been used by Sekaran for the analysis and design of L2CAP (Logical Link Control and Adaptation Layer Protocol) of the Bluetooth architecture [11]. Jaragh and Saleh propose the use of UML for designing behavioral, structural and architectural models covering both the static and dynamic aspects of protocols [12]. However, UML lacks many of the semantics needed for protocol engineering. Towards this direction is the work of Pärssinen et al. [13], which defines the Graphical Protocol Description Language (GPDL) as an extension of UML. Pärssinen et al. have also developed a tool aiming to translate the GPDL models into SDL models.

3. APPLIED METHODOLOGY

For the development of the TCP protocol stack we decided to utilize our methodology that has been already successfully applied in other application domains. However, the methodology was tailored to satisfy the peculiarities of protocol design. Furthermore, some extensions to the UML notation are proposed to better handle the communication and synchronization requirements in the protocol development domain. In this section, we briefly refer to the outline of the applied methodology giving emphasis on the modifications imposed by the nature of protocol software.

For the development of the analysis model, the use case driven approach was adopted to delimit the system and define its functionality. Two types of models mainly constitute the analysis model of the system. The use case model and the problem domain logical view. In order to construct the use case model, the construct of actor is used to represent the roles that software entities play interacting with the protocol. Each actor may perform a number of use cases. In order to increase reusability from this early phase, we develop a use case diagram that captures associations, such as adds and extends, between the use cases. For the description of each use case the format proposed by Rumbaugh was adopted. However, we consider each use case in at least two levels of abstraction. The first-level description considers the interaction of the system as a whole, with the entities of its environment. A more formal representation of this description is obtained using a corresponding first-level Object-Interaction Diagram (OID) [14]. In this first-level OID, the protocol under development is represented as an entity and its interactions with external systems are captured. Fig. 1 illustrates the first level OID for the "active open" use case of the TCP protocol, which was considered as case study in the context of this work.

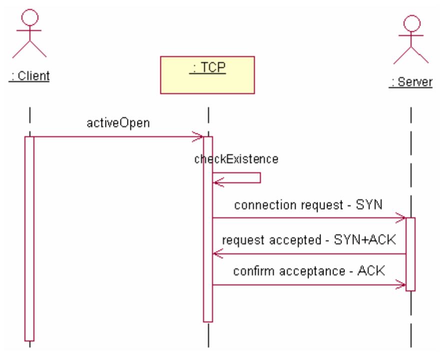

Figure 1 - First level OID of the "active open" use case.

This first-level OID is next expanded to a lower level abstraction OID, which we call detailed-OID. The objects that are required to compose the system for the described behavior to be provided should be identified and their collaboration defined. Coupling and cohesion are among the parameters that help the designer to identify these objects but unfortunately there are no well-defined rules to proceed. The designer's skills are at the moment the most important parameter for an effective design to be achieved. However, the proper definition of specific design patterns from the protocol domain should increase productivity in the design of communication protocols and speed up the development process. The work of Pärssinen and Turunen[15] address this issue. We are currently working to this direction too. Fig. 2 depicts the detailed OID of the "active open" use case. A thorough reference to this OID is given in the next section which deals with the TCP case study.

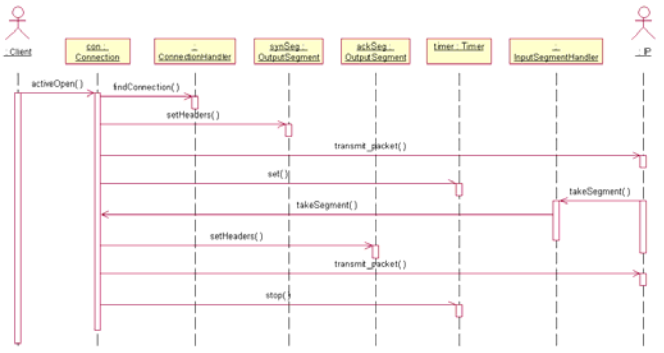

Figure 2 - Detailed OID of the "active open" use case.

The problem domain model is composed of class diagrams that capture the key abstractions of the problem domain and constitute the logical view of the system. It represents as objects, entities or concepts from the problem domain for which the system should handle information. These objects are later used in the construction of system's OIDs. A data driven approach results in the construction of the problem domain model before the use case model. However, a hybrid approach that involves the construction of the use case model in parallel with the construction of the problem domain model seems to be the best choice.

To proceed with the design of the system, we refine each OID to evolve to a more detailed OID that can be implemented with the selected implementation environment. During analysis, every object of the system is considered as active. However, this is not possible for the implementation. Concurrency and synchronization must be considered and the communication and synchronization mechanisms provided by the implementation environment should be properly utilized for an optimum implementation to be accomplished.

Nevertheless, the UML standard does not provide the required constructs to capture these design issues. The activity diagrams can not be used, since they describe the execution of a single thread, which can fork and join. In contrast, our aim is to show the synchronization between active objects that interact in the context of a use case. Moore and McLaughlin have proposed a concurrency or tasking diagram, which has been based on the collaboration diagram[16][17]. This diagram depicts the tasks and the way they interact through various mechanisms (i.e. fifos, mailboxes). This approach is not applicable in our case, so we proposed the required extensions to the UML notation to capture the following semantics:

a) *Concurrency*. In order to be able to represent in an OID more than one threads of control, we have introduced the dashed line notation in the body of an object. When a thread of control is suspended or blocked, its body lines are converted to dashed. Automatic conversion can be obtained based on the semantics of the posted and received messages. To simplify the diagram, we decided to allow a passive object to be shown in the OID more than once. The `con` object for example in fig. 3 is executed by both threads i.e. the Client instance thread and the InputSegmentHandler instance thread. This can not be shown with the today's UML CASE tools.

b) *Synchronization*. Operations with special semantics like set-timer(), wake-up(), wait() and notify() are defined and used to support synchronization between threads of execution.

In fig. 3, which shows a design level OID of the "active open" use case of TCP, the above extensions are used to represent concurrency and synchronization in the OID.

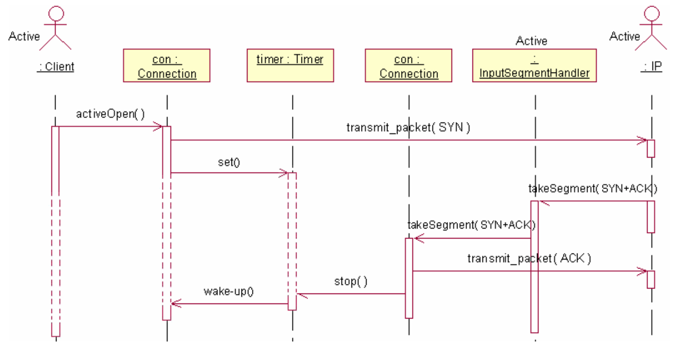

Figure 3 - Design level OID of the "active open" use case.

Three active objects have been shown: the Client instance, the IP instance and the InputSegmentHandler, which represents the TCP's input thread. The arrival of a segment is passed as an asynchronous message from IP to the InputSegmentHandler. Synchronization between the Client and the InputSegmentHandler threads is obtained using an instance of Timer. The client thread which sets the Timer instance, is suspended (dashed lines) until a stop signal arrives, or there is a timeout. The suspended thread is then woken-up and returns to normal execution.

4. THE TCP CASE STUDY

The Rational Rose general-purpose CASE tool [1] was used for the design of the TCP protocol for RTLinux. The use case model is presented in fig. 4. Each use case was described and a first-level OID as well as at least one detail-OID was constructed for it.

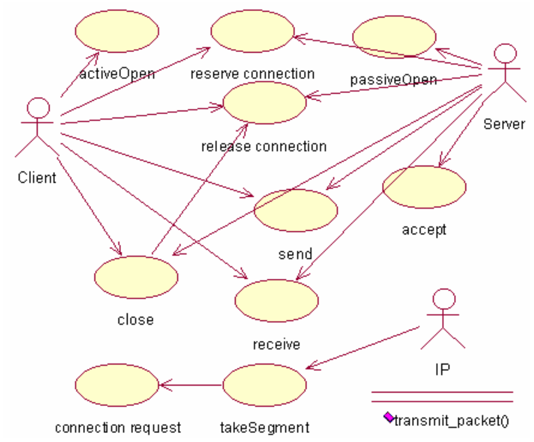

Figure 4 - The use case model of TCP.

Fig. 2 shows the detailed OID of the "active open" use case. The main responsibility of the ConnectionHandler object is to handle the connection objects. It is responsible for creating and destroying connection instances, as well as finding the connection instance that has a set of specified characteristics.

The problem domain model was constructed in parallel with the construction of detailed OIDs. Fig. 5 shows a part of this model of TCP. The `Connection` class, which is the heart of this model, has as data members all the information that in traditional TCP implementations is stored in a TCB. Method specifications are given according to the instance responsibilities. Method `takeSegment()` for example, processes the incoming from IP segments; it checks the segment's sequence number, acknowledgement number and flags, then copies any data to the connection's input buffer and, if necessary, it sends back an ACK segment.

The analysis class diagram was refined to produce the design class diagram. Two active objects were identified: a) the InputSegmentHandler for handling incoming from IP segments and b) the OutputSegmentHandler for managing the transmission of segments. Even more, each actor constitutes an active object.

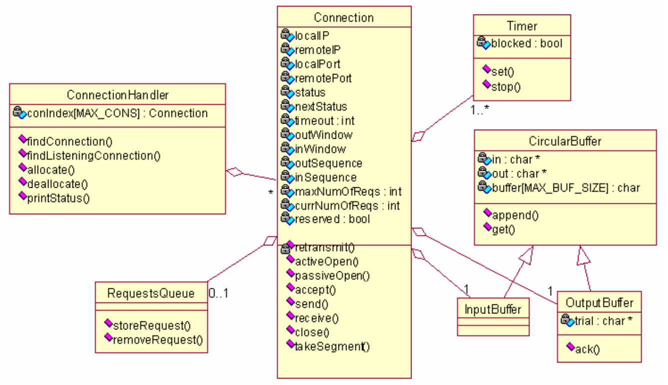

Figure 5 - Analysis class diagram of TCP (partial).

The implemented TCP layer, which was named OOTCP (Object-Oriented TCP), was developed for version 3.1 of the RTLinux kernel. For its implementation, the C++ language was selected. Compilers producing code that can be easily imported into the RTLinux kernel are available. Thread communication was implemented by real time fifos (rtfifos). For the protection of shared resources we have used mutexes (type `pthread_mutex_t`). All Connection instances are created when the TCP module is loaded into the kernel, since dynamic allocation is not supported. The operator `new` and the `malloc()` function are allowed to be used in the RTLinux kernel only in the initialization of a module (function `init_module()`), since they do not satisfy real-time constraints. All mutexes and rtfifos must also be initialized (functions `pthread_mutex_init()` and `rtf_create()`) by the `init_module()` function.

One serious problem was that the C++ code of OOTCP could not be compiled when we included several Linux header files, i.e. `<linux/skbuff.h>`. These headers are used by RTNet and define fundamental structs, like skbuf, which is a buffer used by Linux to store the contents of a datagram [18]. In order to overcome this problem, we created RT-Interface, a module written in C, to handle the interconnection with RTNet. Fig. 6 illustrates the architecture we adopted. RT-Interface has no threads of its own; it only provides a set of functions to be called by RTNet and OOTCP. When RT-Interface is loaded, it registers one of its functions to RTNet in order for RTNet to call this function every time a TPDU arrives. When this function is called, RT-Interface passes the address of the TPDU to OOTCP through an rtfifo. This fifo is registered to RT-Interface by OOTCP, when the latter is loaded. An rtfifo handler is used to wake up the OOTCP's input thread every time RT-Interface writes into the fifo. When OOTCP has finished processing a received TPDU, the skbuf containing it has to be released in order to be reused. This is done by RT-Interface, since OOTCP can access only the skbuf's data, that is the TPDU. Finally, upon sending a TPDU, the corresponding function of RTNet is called through RT-Interface.

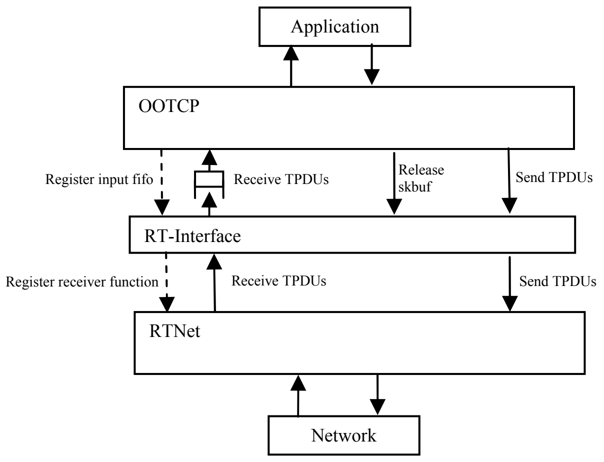

Figure 6 - TCP's interface with IP

5. DISCUSION - EVALUATION

The development of a protocol using the OO approach and the UML notation was successful. The resulting protocol implementation: a) is easy to be understood; the detailed design diagrams hide the time consuming, low level implementation details and b) is expandable; new functionality was added first in the design models and it is then translated to code. However, since the most important issue in protocol implementation is the performance, we created a testbed in order to measure, under certain conditions, the performance of the developed OOTCP/RTNet protocol stack. We performed the same measurements for the TCP/IP stack of Linux and used them as a yardstick. The main motivation of this evaluation was to verify the smooth operation of our protocol stack under heavy load. It must be clarified that the design of OOTCP and the subsequent implementation has not been optimized to increase performance.

The performance evaluation of a protocol stack involves measuring several parameters. Zanella et al. [19] for the performance evaluation of TCP Westwood and Reno used simulations as well as the application of the implementations' analytical models. They measured the TCP's throughput with respect to variations of the error probability, buffer size, bandwidth and round trip time (RTT). Perkins and Hughes [20] have investigated the performance of TCP in mobile ad hoc networks (MANETs) by evaluating the impact of path length, node mobility and routing on the throughput of TCP. Pentikousis [21] has tested TCP Tahoe, Reno and NewReno under random and burst errors as well as combinations of these two categories. He simulated these errors during the transfer of a 5MB file between two hosts over a 10Mbps network and measured the delays inserted.

Our tests were conducted between two dedicated hosts, directly interconnected via crossed Ethernet cable of 100Mbps. One host was running the server application, which was using the TCP/IP stack of MSWindows. The client application on the other host was, in the first case, an RTLinux application utilizing the services of OOTCP/RTNet, and a Linux application using the services of Linux TCP/IP in the second case. Once the client established a connection with the server, it received a 5MB file and then sent it back. The results of the performance test are given in Table 1. The receiving and sending time is the time taken for each protocol stack to receive and send the file respectively. The TCP throughput [22] is calculated as:

*BytesSent * 8 / Sending time*

It can be seen that the results are almost identical for both stacks, even though an overhead was expected from the OO implementation.

Table 1 - Performance evaluation results

	Linux TCP/IP	**OOTCP/RTNet**
Bytes received	5242900	5242900
Receiving time (sec)	1.181	1.201
Bytes sent	5242900	5242900
Sending time (sec)	0.821	0.882
TCP throughput (Mbps)	51	47.5
Total time (sec)	2.002	2.083

6. CONCLUSIONS

In this paper, an OO methodology to facilitate the development of communication protocols was presented. This methodology is a tailoring of our methodology that we have used successfully for many years in many application domains. The methodology exploits the OO approach and the widely accepted UML notation. In our attempt to address the design issues of protocol engineering, a number of extensions to the UML notation were proposed. The methodology was utilized for the development of a TCP protocol stack for the RTLinux RTOS. The experiment was successful. The resulting implementation not only has enhanced readability, modularity, and expandability but also presents performance characteristics comparable with those of the corresponding protocol stack of Linux, even though no extra optimization techniques were used to enhance performance.

However, better results are expected using the profile of UML for real-time modeling. Working in this direction it will become clear if a specific UML profile for communication protocols should be defined. Such a profile will give the modelers: a) access to common model elements and b) terminology from the communication protocol domain. This is the first step towards a model driven approach for the development of protocol software.

REFERENCES

[1] Rational home page, http://www.rational.com

[2] K. Thramboulidis, C. Tranoris, "Developing a CASE tool for Distributed Control Applications", The International Journal of Advanced Manufacturing Technology, Springer-Verlag (forthcoming).

[3] Thramboulidis, K. "Development of Distributed Industrial Control Applications: The CORFU Framework", 4th IEEE International WFCS, Sweden, August 2002.

[4] Lineo's OpenSource Development Repository Project Page, http://opensource.lineo.com/projects.html

[5] K. Thramboulidis, P. Parthimos, G. Doukas, "Using RTLinux to Interconnect Field Buses: The Profibus Case Study", ICMEN 2002, October, Greece.

[6] Jeremy Bentham, "TCP/IP Lean, Web Servers for Embedded Systems", CMP books 2000.

[7] Jan Ellsberger et al, "SDL: Formal Object-Oriented Language for Communicating Systems", Prentice Hall 97.

[8] Estelle, ISO International Standard IS8807, July 1989.

[9] Lotos, ISO International Standard IS8807, Feb. 1989.

[10] H. Huni, R. Johnson, R. Engel, "A Framework for Network Protocol Software", 10th OOPLSA, October 95.

[11] Sekaran K. C., "Development of a link layer protocol using UML", Proceedings of IEEE International Conference on Computer Networks and Mobile Computing, October 2001.

[12] Jaragh M., Saleh, K. A., "Protocols modeling using the Unified Modeling Language", Proceedings of IEEE TENCON, August 2001.

[13] J. Pärssinen, N. Knorring, J. Heinonen, M. Turunen, "UML for Protocol Engineering–Extensions and Experiences", TOOLS 33, June 05 - 08, 2000, St. Malo, France.

[14] Thrampoulidis, K. K. Agavanakis, "Introducing Object Interaction Diagrams: A technique for A&D", Journal of Object-Oriented Programming (JOOP), June 1995.

[15] J. Pärssinen, M. Turunen, "Patterns for Protocol System Architecture", PLoP2000, August 13-16, Illinois 2000.

[16] Alan Moore, "Extending UML to Real-Time Systems", Embedded Developers Journal, March 2001.

[17] McLaughlin, M. Alan Moore, "Real-Time Extensions to UML", Dr. Dobb's Journal December 1998.

[18] Jon Crowcroft, Iain Philips, "TCP/IP and Linux protocol implementation", John Willey, 2002.

[19] A. Zanella, G. Procissi, M. Gerla, M. Sanadidi, "TCP Westwood: Analytic Model and Performance Evaluation", Globecom 2001, November 2001, San Antonio, Texas.

[20] D. Perkins, H. Hughes, "Investigating the performance of TCP in mobile ad hoc networks", Computer Communications, Vol. 25, Issues 11-12, July 2002.

[21] K. Pentikousis, "Error Modeling for TCP Performance Evaluation", Master's Thesis, State Univ. of New York, May 2000.

[22] Andrew S. Tanenbaum "Computer Networks", 4th edition, Prentice Hall 2002.