UML-based Design of Network Processor Applications

A. Bertolino, G. De Angelis
Istituto di Scienza e Tecnologie
dell'Informazione "Alessandro Faedo" CNR
Via G. Moruzzi 1, I–56124 Pisa, Italy
{antonia.bertolino,guglielmo.deangelis}@isti.cnr.it

R. Mirandola
Dipartimento di Informatica,
Università di Roma "Tor vergata"
Via del Politecnico 1, 00133 Roma, Italy
raffaela@info.uniroma2.it

Abstract

*Network Processors (NPs) are an emerging class of embedded systems used in the telecommunication domain for functionality like routing and switching. In this paper we outline a framework currently under development for the use of a Model Driven Engineering approach to the design of NP software applications. Specifically we introduce the UML profile called NAP (**N**etwork Processors **A**pplication **P**rofile) which has been specialized so far for the design of SCTP applications.*

1 Introduction

With the growing relevance and pervasiveness of telecommunications, a good deal of interest is drawn by the study of Network Processors (NPs) which play an important role in the design of modern routers. In simple terms, NPs are specialized embedded systems filling a middle ground between totally hard-coded solutions on one side and general purpose programmable devices on the opposite one. Designing applications for NP architectures is a complex task, since such processors are required to support sophisticated packet processing functions at high line speeds, and at the same time be programmable so to easily incorporate new functionality. To support such stringent demands, NPs consist of a collection of heterogeneous processing elements, memory subsystems and on-chip communication infrastructure. Dealing with the diversity of technological solutions, the heterogeneity of processing elements and the possibly distributed software solutions make the application design very hard.

Today, the industrial approach to the design of embedded systems remains often informal rather than based on a well-defined engineering discipline. To overcome this limitation we have launched a research initiative whose goal is the definition of a rigorous approach for the design of NP applications based on the principles of Model Driven Engineering (MDE)[1]. The basic idea of MDE is to create a set of models that help the designers understand and evaluate both the system requirements and implementation. However, as stated in the call for paper *"while the use of models and related computer aided engineering tools is very common in established engineering disciplines, MDE approaches are still rarely used in the industry for the development of embedded systems"*. Our goal is to exploit the MDE approach and specialize it to the domain of NPs.

In particular, we are pursuing a design approach based on UML, which is now a de-facto standard in software industrial development, and at the core of many MDE approaches. In our view, however, a key point of a design methodology for NP should be to integrate the classical MDE development approach with an orthogonal model development taking into account cross-cutting aspects such as the performance or the reliability of the NP applications. But, as generally recognized, the original UML meta-model is too general to allow for the modelling of aspects specific to embedded systems, and NPs too. Therefore, a more specialized use of UML becomes necessary to facilitate the modeling of this specific application domain. Specifically, in this paper we start the definition of a domain specific language, a profile called *NAP (**N**etwork Processors **A**pplications **P**rofile)*, built on the top of UML infrastructure and compliant with the UML meta-model. Moreover, to take into account the performance aspects, we integrate the NAP profile with performance annotations derived from the well-known RT-UML Profile for Schedulability, Performance and Time [14]. The philosophy followed in the profile definition has been to adhere as much as possible to existing OMG standard (as the RT-UML [14]) and to exploit results and tools obtained in similar areas (see Section 2).

The paper is organized as follows: Section 2 presents some background information on NP and MDE and an overview of the presented approach. In Section 3 we describe the methodology we have followed in the definition

[1] Preliminary ideas have been presented in [1]

of our profile and section 4 is devoted to the detailed description of the proposed UML profile. A simple application example is described in Section 5 while conclusions are drawn in Section 6.

2 Background and Overview

In this section we give some background information on the main concepts of MDE, NP and UML profiles, and in Section 2.4 we shortly overview the core points of this paper's contribution.

2.1 Model Driven Engineering

The idea promoted by MDE is to use models at different levels of abstraction for developing systems. Thus, the main activity of MDE developers is to design models, instead to develop code. The advantage of having an MDE process is that it should clearly define each step to be taken, forcing the developers to follow the defined methodology (and this is in fact our own motivation for developing a MDE approach for NPs). The underlying philosophy is to begin by identifying the principal elements of the system and the ways in which they interact, and then supply any details that prove to be necessary. The refinement steps, in some application field, could be automated and different models should conform to the overall meta-model. Several high-level models can be obtained as a starting point of this top-down approach according to the objective of the study. The integration of different models at possibly different levels of abstraction is the key point to obtain high quality and efficient system design.

Integration is concerned on a vertical level as well as on a horizontal level. In the latter case, starting from a model at a given abstraction level it is possible to provide some refinements guided by the objective of the study. We can study, in particular, the performance of the system building performance models and solving them using some automated methodology proposed for this goal [4, 6, 20, 15, 18]. A horizontal approach can be useful also to apply an aspect-oriented development approach [17], where different aspects can be seen as different refinement of an original model. One of the most important advantages of using an MDE process is its adaptability to changes. When a change occurs, be it at the highest level of abstraction or at a lower level of abstraction, its impact is well localized and the parts that are not touched by the change are immediately reusable.

2.2 NP basic concepts

Network devices are a growing class of embedded system and include: traditional Internet equipment like routers, switches, and firewalls; newer devices like Voice over IP (VoIP) bridges, virtual private network (VPN) gateways, and quality of service (QOS) enforcers; and web-specific devices like caching engines, load balancers, and SSL accelerators.

A NP is a programmable device with architectural features for packet processing at rates of 1 Gbit/s and above. The main NP functions are: header classification; deep packet analysis; packet processing; policing and statistics; and traffic management.

Although a wide variety of NP architectures exists, they share some common features. A key point with all architectures is that they all employ multiple programmable processing engines (PPE) [11]. The main function of the NP is to process packet data at wire-speed, so the existence of PPEs. On the other hand an NP needs to manipulate control and management packets that generally don't need to be processed at such a high rate but have complex processing requirements. This reason leads to have a Control Processor Element (CPE) [11] which in some cases can execute simple control plane applications and is responsible of other functions such as booting or hosting interface. Besides, NPs, as any programmable devices, clearly also require memories to store programs, lookup table, packet or queue information. NPs supply a very compact internal memory that is mainly used to store the code for packet processing by the PPEs. External memories are anyhow required to run applications; for these, NPs provide one or more external memory interfaces managed by controllers, often specialized for a particular role such as queueing or table look-up handling. Data plane tasks require a small amount of code, but a large amount of processing power. In contrast, control plane tasks require little processing power, but a large amount of code. Many NPs provide also an interface to Fabric Switch (FS).

As well as the different solutions have some common hardware feature, recurrent languages approaches exist for programming NPs. Mainly, the C language (or a variant of it) is used to program the CPE and in some cases also the PPEs, although often for the latter an assembly approach is preferred (like Intel or Freescale solutions) [13, 12]. Many PPEs, in fact, have an architecture that is not a good target for a C program, lacking for example a support for pointers in its instructions set. Moreover the code is too hardware dependent, including register swapping or clock cycles waiting. Finally, there are some families of NPs that allow to program their elements by means of functional or 4^{th} generation programming languages (like Agere solution) [2]. The reason of this choice is because classification is an important part of NP software, and this kind of operations can be simply described by means of rules applicable under specified conditions.

2.3 Meta-models and UML Profiles

Two emerging trends in software development are: the use of UML as a de-facto standard in the industrial environment; and the use of specialized notation to make easier the modeling of specific application domains. An active research direction that attempts to merge these two aspects leads to a more specialized use of UML yielding to the definition of domain specific languages, called profiles, built on the top of UML infrastructure and (possibly) compliant with the UML metamodel (see for example [7, 8]). In the following of this section we briefly survey some works concerning the use of UML for modeling real-time and embedded systems.

A UML Profile for Schedulability, Performance and Time (RT-UML) has been proposed and adopted as an OMG standard [14] as a response to the exigencies of introducing in UML diagrams quantifiable notions of time and resources usage. RT-UML is not an extension to the UML metamodel, but a set of domain profiles for UML. Basically, the underlying idea is to import as annotations in the UML models the characteristics relative to the target domain viewpoint (performance, real-time, schedulability, concurrency), in such a way that various (existing and future) analysis techniques can usefully exploit the provided features. In fact, the RT-UML profile is intended to provide a single unifying framework encompassing the existing analysis methods, still leaving enough flexibility for different specializations.

A different profile, called UML-RT [16], has been proposed to model software architectures in real-time systems. UML-RT extends UML with stereotyped active objects, called *capsules*, to represent system components, where the internal behavior of a capsule is defined using state machines. The interaction with other capsules takes place by means of protocols that define the sequence of signals exchanged through stereotyped objects called *ports* and specify the desired behavior over a *connector*.

A different profile, called UML Platform Profile (UML-PP) has been proposed in [5]. The basic principles of UML-PP has been inspired by the work done with the Metropolis design environment [3] and its semantics is defined in terms of the Metropolis Metamodel by establishing a direct correspondence between modeling elements of UML-PP and elements of the Metamodel. In this view the behavior of an embedded system can be captured at different levels of abstraction using Use Case (for an abstract representation of a system), Interaction (among system components), State Machine and Activity Diagrams (for the behavior of individual components).

2.4 Overview

Our long-term goal is to devise an MDE approach for applications over NPs. Hence our domain spans over the three fields summarized above: MDE, NPs and UML profiles (as illustrated in Fig. 1). The methodology we are developing hence, fully complies with the philosophy of MDE; with the latter we have incorporated knowledge specific to NP and more in general embedded system design inspired by the traditional Y-chart [9], and also including timing and performance requirements. The technological implementation is however filtered through standard UML element and meta-model extensions via the usual approach of a profile definition. In this paper we cover the UML NAP profile (which is at the core of our approach, as shown in Fig. 1) definition and quickly hint at performance analysis techniques which we intend to incorporate.

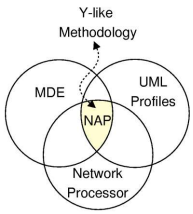

Figure 1. Covered domains

3 The Y-like Methodology

The methodology we have developed, and described in the next Section, is inspired to the conventional Y-chart approach, commonly used for hardware-software co-design. Developed in 1983 [9], the Y-Chart makes the assumption that a design can be modeled in three basic ways emphasizing different aspects [10]. As depicted in Fig. 2, a Y-Chart has three axes representing Behavioural, Structural and Physical Domains.

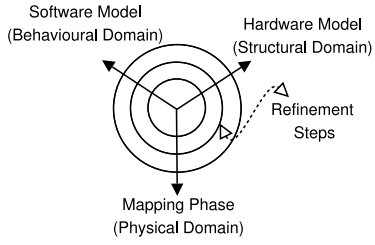

Figure 2. The Y-like Methodology

The Behavioural Domain represents software features: the specification, the functionalities and the architecture, which must be defined as platform-independent as possible.

On the other side, the Structural Domain axis describes the platform features in terms of physical elements, such as processors or memories, and communication channels such as busses. Non-functional annotations such as delay time or processor speed are considered here as well.

In the Y bottom, the Physical Domain, the software processing units and structures are mapped onto the hardware resources, taking into account possible non-functional annotations and requirements. If a refinement is required, a rollback step can be followed, modifying the Behavioural or Structural Domain.

As for any significative complex design, creating the right solution in a single step is quite impossible. Furthermore mixing high level concepts with lower level ones clearly reduces the solution reusability. For these reasons each domain can be dealt at different abstraction levels. This notion is depicted in Fig. 2 by the concentric circles around the mapping.

Y-Chart represents a general approach to solve design problems, and, depending on the specific context, can be implemented by means of different technologies such as algebras or Object Oriented. In an heterogeneous field such as the one of embedded systems, we believe that the notion of Y-Chart can be quite usefully and effectively combined with the MDE techniques. In other words, we intend to follow the Y-Chart scheme as the general guidance for design, by identifying the Behavioural and Structural Domains for the NPs. For expressing the design, we intend to use instruments proper to the MDE methodology, such as profiles and (automated) transformation mechanisms. The advantages we see in combining these two approaches are of two kinds, on one level the capability to manage typical problems in design of embedded systems in a simpler, more natural way than proceeding in an ad hoc (non model-driven) fashion; on the other, the possibility to reuse, also in part, solutions (for instance only in one of the Y-Chart domains) and maintain the design at a high-abstraction level.

4 UML Profile

This section describes our proposal for the Beavioural Domain in the Y-like methodology. Our intent is to define a UML Profile to describe applications for NPs, called NAP (**N**etwork Processors **A**pplication **P**rofile), in order to maximize software design reuse. Our original motivation was to implement SCTP [19] endpoints on such processors. However, the proposed profile is organized into packages so that it can be easily extended to other domains of interest, by describing the possible new features into another imported package.

As usually done, we present first the domain concepts underlying our profile and then the related UML concept. In particular, a UML class diagram notation is used in Section 4.1 to describe the domain viewpoint; then, in Section 4.2, we describe how the elements of the domain model are represented by means of UML extensions [1].

4.1 Domain Viewpoint

Fig. 3 shows how the domain concepts are grouped and what are the dependencies among them. The definitions of *PAprofile* and *RTtimeModeling* are imported from the OMG RT-UML Profile [14]. *NAPCore* defines general concepts and relations to operate and define NP software applications; while *NAP SCTP* defines additional specific concepts used to describe an SCTP application.

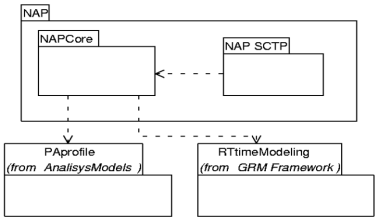

Figure 3. Domain Groups of NAP Profile

As depicted in Fig. 4, we conceive an application for the management of a generic communication protocol as logically organized in a bi-dimensional matrix, highlighting two orthogonal concepts: the rows represent the architectural elements which compose the application, while the columns represent the aspects which a protocol involves, e.g., send, receive and control. In such logical representation each cross point identifies a particular aspect for a specified architectural element. Not every architectural element has to appear in all the aspects.

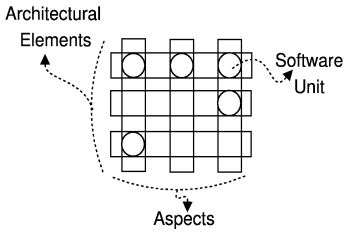

Figure 4. Applications Logical Organization

Accordingly, NAPCore organizes applications in *SoftwareUnit*s, each specifying an architectural element (*archElement*) and characterized by an *aspect* (see Fig. 5). A SoftwareUnit is composed by one or more *Atom*s (the implemented functionalities). An Atom can communicate with another Atom of the same SoftwareUnit by means a *Call*; different Atoms can also share SoftwareUnit resources.

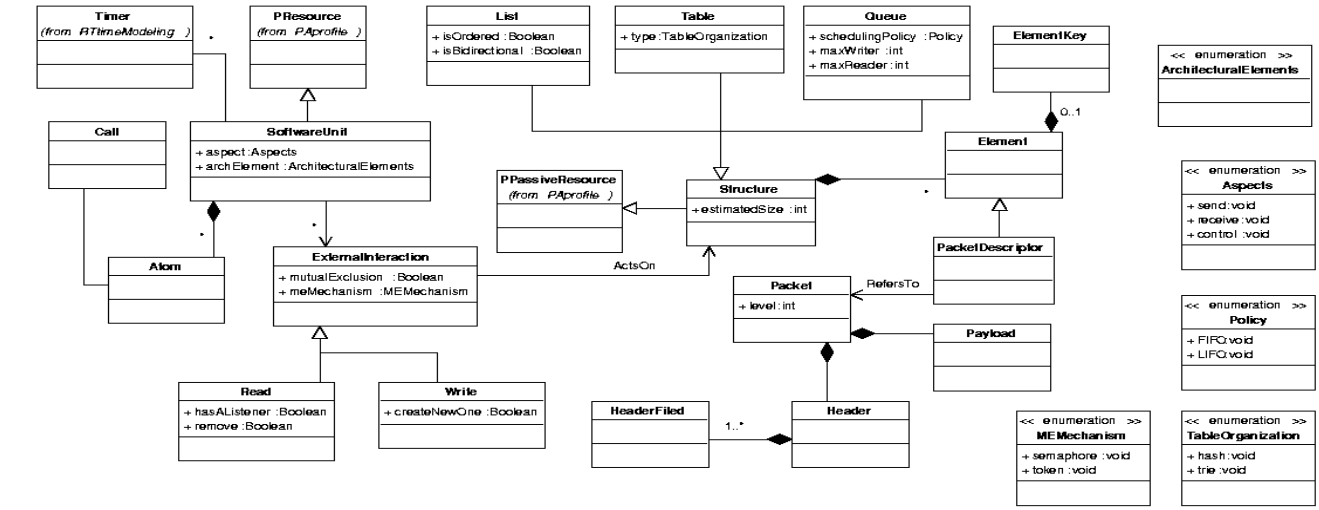

Figure 5. NAPCore Domain Viewpoint

A SoftwareUnit acts on shared *Structure*s by means of *Read* or *Write* operations. Access to the structures can be direct or protected by a mutual exclusion mechanism. The Read can be performed on demand or via a listener connected to the SoftwareUnit. A *Structure* can be a *List*, a *Table* or a *Queue*; all of them are composed by *Elements* which can have a key (*ElementKey*). A particular Element can be a *PacketDescriptor* that refers to a *Packet*. A Packet belongs to a *level* according to the OSI stack, and is composed by a *Header* and a *Payload*. A Header supports one or more *HeaderField*s.

As said, in NAP we import some definitions from [14]. In particular, a SoftwareUnit extends PResources in package PAprofile defining the number of read (*nRead*) or write (*nWrite*) operations on the different Structures, the memory occupation for its local state (*localStateSize*) and its estimated source code lines (*codeLines*). Furthermore a SoftwareUnit can refer to one or more Timer from package RTtimeModeling. On the other hand, a Structure can be considered as PPassiveResources of PAprofile package [14].

NAP SCTP defines SCTP features using the general concepts described in NAPCore. In this package the notion of Element is specialized (see Fig.6) in *SCTPInstance* and *ChunkDescriptor*. An SCTPInstance is characterized by a *SecretKey* (used by an endpoint to compute the authentication codes), an *SCTPPort* and a set of *TransportAddress*es. Finally, an SCTPInstance can *Manage* one or more *SCTPAssociation*s. A ChunkDescriptor element refers to a *Chunk*, which is composed in turn by a *ChunkHeader* and an optional *ChunkPayload*.

This package also defines the set of admissible architectural elements for an SCTP application and the supported chunk types.

4.2 UML Viewpoint

This section exposes how the domain concepts introduced in Sec. 4.1 can be represented and mapped in the UML modelling environment by means of stereotypes and tagged values. In the following, stereotypes are written, as usual, between $\ll$ and $\gg$ parentheses and tagged values in italic. All tagged values have a null value as default, while the boolean one has to be considered as set to False. Table 1 shows for each domain concept: the UML concept which it extends and the name of the related stereotype.

A $\ll SoftwareUnit \gg$ is an extension of a Class; it is tagged with an *aspect*, an *archElement* and one or more references to elements stereotyped as $\ll Atom \gg$.

The interactions among different $\ll Atom \gg$s of the same $\ll SoftwareUnit \gg$ are described by means of the stereotyped association $\ll Call \gg$, while the interactions among different $\ll SoftwareUnit \gg$s by means of associations $\ll Read \gg$ and $\ll Write \gg$ over some global structure denoted as $\ll List \gg$, $\ll Table \gg$ or $\ll Queue \gg$. A $\ll Read \gg$ or $\ll Write \gg$ stereotyped association can be tagged with a *mutualExclusion* flag and possibly with a *meMechanism*. Moreover, a $\ll Read \gg$ can be tagged also with a *remove* flag or *hasAListener* one; a $\ll Write \gg$, instead, can only have the *createNewOne*, which, if set, allows the associated $\ll SoftwareUnit \gg$ to create a new element into the referenced structure.

All the structures support the annotation *estimatedSize*; furthermore, considering a $\ll List \gg$, this can be characterized by the flags *isOrdered* and *isBidirectional*; a $\ll Table \gg$, by a *type*; and a $\ll Queue \gg$ by its *schedulingPolicy* and the number of *maxReader*s and *maxWriter*s.

A structure is composed by $\ll Element \gg$s, which

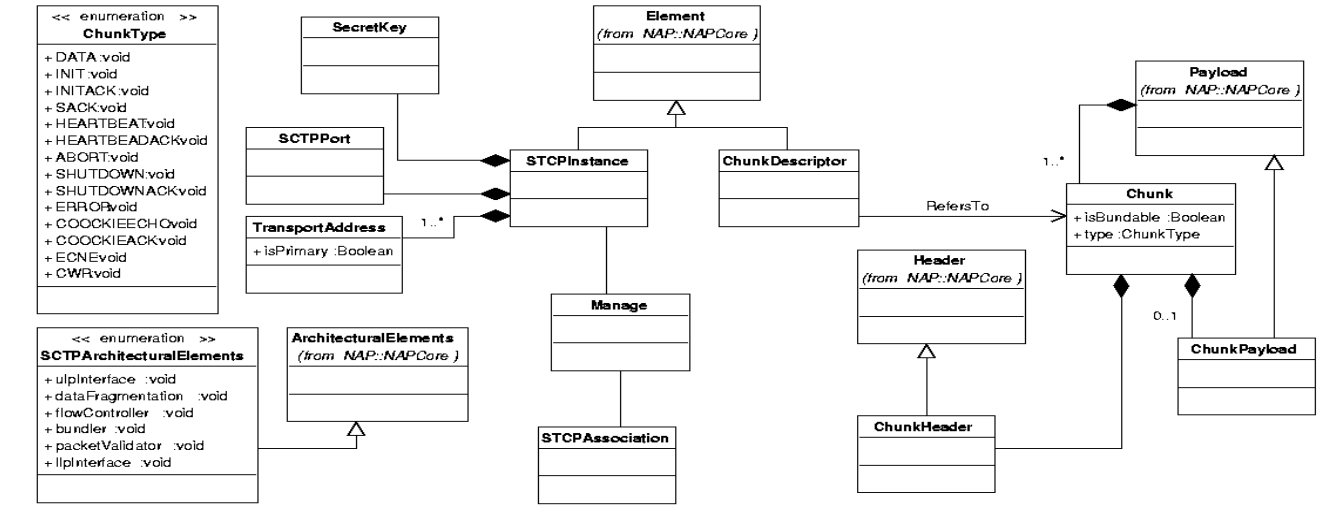

Figure 6. NAP SCTP Domain Viewpoint

can reference a $\ll Key \gg$. A particular element is a $\ll PacketDescriptor \gg$ which has to reference to a $\ll Packet \gg$. The $\ll Packet \gg$ stereotype also extends the Class, and is tagged with OSI *level* and references one or more $\ll HeaderField \gg$s and a $\ll Payload \gg$.

Concept	Extends	Stereotype
SoftwareUnit	Class	$\ll SoftwareUnit \gg$
Atom	Attribute / Class	$\ll Atom \gg$
Read	Association	$\ll Read \gg$
Write	Association	$\ll Write \gg$
Call	Association	$\ll Call \gg$
List	Class	$\ll List \gg$
Table	Class	$\ll Table \gg$
Queue	Class	$\ll Queue \gg$
Element	Class	$\ll Element \gg$
ElementKey	Attribute / Class	$\ll Key \gg$
PacketDescriptor	Class	$\ll PacketDescriptor \gg$
Packet	Attribute / Class	$\ll Packet \gg$
Payload	Attribute / Class	$\ll Payload \gg$
HeaderField	Attribute	$\ll HeaderField \gg$

Table 1. NAPCore in UML Viewpoint

As shown in Table 2, the NAP SCTP package in the UML viewpoint extends the concept of Class by means of the stereotype $\ll SCTPInstance \gg$, which provides a reference to an element stereotyped as $\ll SecretKey \gg$, an element stereotyped as $\ll SCTPPort \gg$ and one or more $\ll TransportAddress \gg$es. One or more $\ll SCTPAssociation \gg$s can refer to an instance by means of a $\ll Manage \gg$ association.

As for the descriptor of a packet, a $\ll ChunkDescriptor \gg$ is a Class extension which has to refer to an element stereotyped $\ll Chunk \gg$; the latter is then tagged with its *type* and a *isBundable* flag. Each element stereotyped as $\ll Chunk \gg$ again refers to one or more $\ll HeaderField \gg$s and an optional $\ll ChunkPayload \gg$.

Concept	Extends	Stereotype
SCTPInstance	Class	$\ll SCTPInstance \gg$
TransportAddress	Attribute	$\ll TransportAddress \gg$
SCTPPort	Attribute	$\ll SCTPPort \gg$
SecretKey	Attribute	$\ll SecretKey \gg$
SCTPAssociation	Class	$\ll SCTPAssociation \gg$
Manage	Association	$\ll Manage \gg$
ChunkDescriptor	Class	$\ll ChunkDescriptor \gg$
Chunk	Attribute / Class	$\ll Chunk \gg$
ChunkPayload	Attribute / Class	$\ll ChunkPayload \gg$

Table 2. NAP SCTP in UML Viewpoint

5 Working Example

This section exposes an illustrative example of the profile use. For space contraints, only a part and the main ideas of a send architecture for an SCTP protocol implementation is presented here (see Fig. 7).

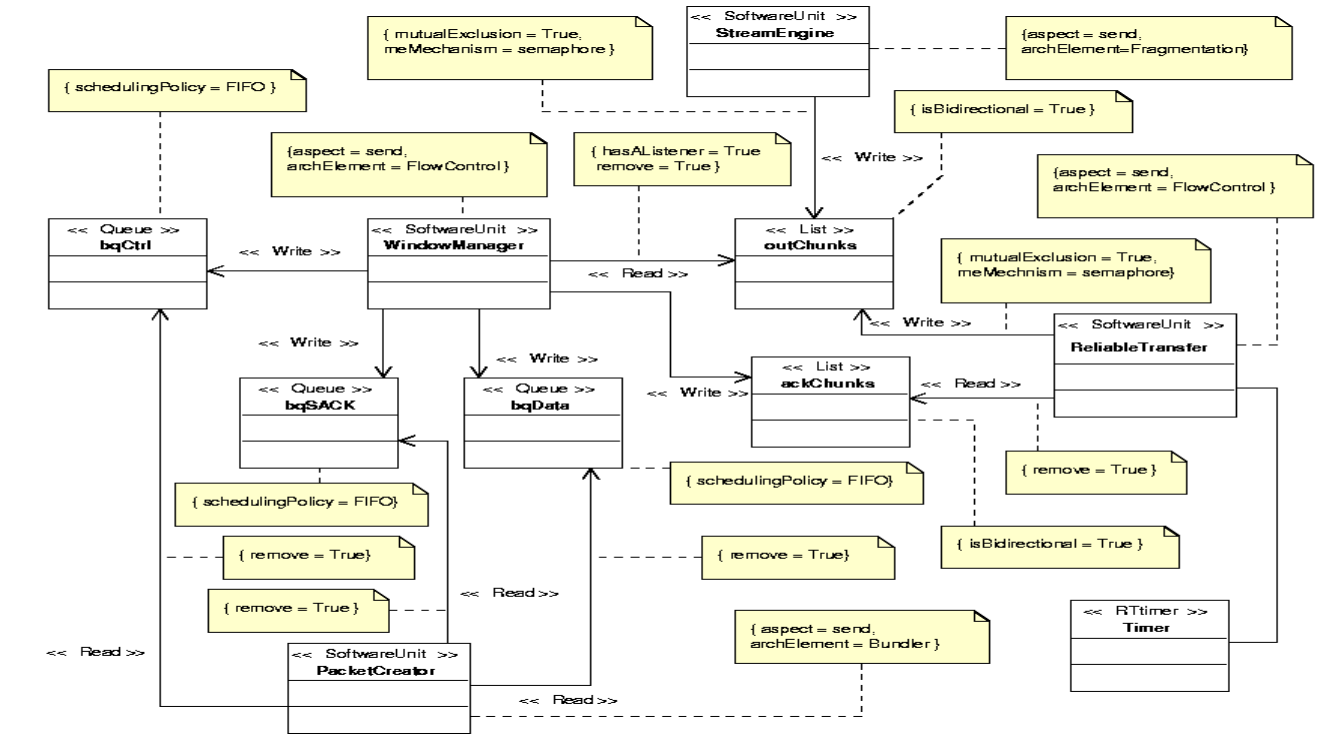

Figure 7. SCTP Architecture for Send Aspect

The $\ll SoftwareUnit \gg$ *StreamEngine* receives data from the upper layer protocol interface (not depicted in Fig. 7), divides the messages in chunk sets and writes them in *outChunks* $\ll List \gg$. The writing access to the structure is regulated via a semaphore.

The *WindowManager* supports a listener on this list; when the transmission window allows to operate it reads and removes a chunk from the list, processes and forwards it on a boundling queue depending on the chunk's type. Three kinds of $\ll Queue \gg$ are required: *bqCtrl* which contains control chunks, *bqSACK* which contains SACK chunks and *bqData* which contains data ones; the three queues are caracterized by a FIFO scheduling policy. For the data chunk an acknowledgement is required, so the WindowManager adds them to an *ackChunks* $\ll List \gg$.

ReliableTransfer, another Flow Control's $\ll SoftwareUnit \gg$, according to $\ll RTtimer \gg$s [14] or incoming acknowledgement for data chunks (not shown in Fig. 7), decides to remove chunks from *ackChunks* or moves them on *outChunks* again, so they can be trasmetted again. Also for *ReliableTransfer*, the writing on the *outChunks* $\ll List \gg$ is protected by a semaphore.

At last, *PacketCreator* is a bundler's $\ll SoftwareUnit \gg$ for the send aspect. Its role is to extract chunks from the differents queues and wraps them up into SCTP packets according to [19].

Starting form the CD in Fig.7, we can now specify the behavior of this architectural element using the conventional UML interaction diagrams. At this point, we can employ all the "classical" instruments that have been and are being still proposed for analyses to be carried on the NP properties of interest. This is not a straightforward point but we have already done a step ahead using standard notations for the performance aspects. Specifically, we plan to follow automatic approaches like the one proposed in [4, 20, 15] to evaluate characteristics like application throughput and response time

6 Open Issue and Future Work

We have presented a Y-like methodology inspired to MDE principles and techniques for the development of applications for Network Processors.

In order to focus on NP software design and increase its reuse across the different hardware platforms, we have developed NAP, a modular UML profile which allows us to describe NP software applications. NAP is composed by a core package (NAPCore) and imports some concepts from RT-UML [14]; if necessary, NAP can be extended according to the application features. So far we have developed NAP SCTP to introduce in the profile specific concepts required

by the design of an SCTP application. Finally the paper illustrates a NAP SCTP application to describe a portion of a software architecture for an SCTP endpoind.

Future work will focus on a complete description of the SCTP case study, introducing also OCL constrains in order to ensure some kind of consistency into the model and the annotations. On the other hand, we intend to define a model-to-model transformation engine. Our goal is to translate the software scenario described in NAP into a lower level model which also includes hardware aspects of a specific NP (maybe coming from an hardware model). A possible target for the future transformation engine can be Metropolis meta-model [5], while the NP that we would like adopt is one of the Freescale C-5 family. Finally, we are also investigating on how to express transformation rules in some platform independent language (e.g. QVT).

Acknowledgments

Guglielmo De Angelis's PhD grant is supported by Ericsson Lab Italy in the framework of the Pisatel initiative.

References

[1] S. Afsharian, A. Bertolino, G. De Angelis, P. Iovanna, and R. Mirandola. A Model Based Approach to Design Applications for Network Processor. In *Proc. of RISE 2004*, volume LNCS 3475. Springer, 2005.

[2] Agere. *The Challenge for Next Generation Network Processors*, 2001. White Paper.

[3] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An Integrated Electronic System Design Environment. *Computer*, 36(4), 2003.

[4] A. Bertolino and R. Mirandola. CB-SPE tool: putting component-based Performance Engineering into practice. In *Proc. of CBSE7*, volume LNCS 3054. Springer, 2004.

[5] R. Chen, M. Sgroi, L. Lavagno, G. Martin, A. Sangiovanni-Vincentelli, and J. Rabaey. *UML for real: design of embedded real-time systems*, chapter 5 – UML and platform-based design. Kluwer Academic Publishers, 2003.

[6] V. Cortellessa and R. Mirandola. PRIMA-UML: A Performance Validation Incremental Methodology on Early UML Diagrams. *Science of Computer Programming*, 44(1), 2002.

[7] D. Di Ruscio, H. Muccini, and A. Pierantonio. A Data Modeling Approach to Web Application Synthesis. *International Journal of Web Engineering and Technology*, vol.1(num.3), 2004.

[8] D. Di Ruscio and A. Pierantonio. Model Transformations in the Development of data-intensive Web Applications. In *Proc. of CAiSE 2005*, 2005. –To appear–.

[9] D. Gajski and R. Kuhn. Guest Editors' Introduction: New VLSI Tools. *Computer*, 16(12), 1983.

[10] A. Gerstlauer, D. Gajski, H. Yu, and J. Peng. System Design Methodology and Tools. Technical report, April 21 2003.

[11] A. Heppel. An introduction to network processors, January 2003.

[12] D. Husak. *Network Processors: A Definition and Comparison*. C-Port, 2000. White Paper.

[13] Intel. *Intel IXP2400 Network Processor: Flexible, High-Performance Solution for Access and Edge Applications*, 2002.

[14] OMG. *UML Profile for Schedulability, Performance and Time Specification*, OMG Document – formal/05-01-02 edition, Jannuary 2005.

[15] D. C. Petriu and H. Shen. Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specification. In *Proc. of TOOLS 2002*, volume LNCS 2324. Springer-Verlag, 2002.

[16] B. Selic and J. Rumbaugh. *Using UML for modeling complex real-time systems*, 1998. White Paper.

[17] The Aspect Oriented Software Development Web Site. http://aosd.net.

[18] C. U. Smith and L. Williams. *Performance Solutions: A practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2001.

[19] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, and L. Zhang. Stream Control Transmission Protocol. Technical Report RFC 2960, IETF, October 2000.

[20] J. Xu, M. Woodside, and D. C. Petriu. Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time. In *Proc. of TOOLS 2003*, volume LNCS 2794. Springer-Verlag, 2003.