Balancing configurability and efficiency in network support tools*

Hugo Miranda

Universidade de Lisboa
FCUL, Campo Grande,
1749-016 Lisboa, Portugal
hmiranda@di.fc.ul.pt

Luís Rodrigues

Universidade de Lisboa
FCUL, Campo Grande,
1749-016 Lisboa, Portugal
ler@di.fc.ul.pt

March 29, 2000

Abstract

New applications require dynamic configuration capability. A protocol kernel is a middleware application that glues a user defined set of protocols, allowing them to cooperate. One difficult in building protocol kernels is to balance the flexibility of adding and removing protocols with the efficiency of the mechanisms that support them. The paper presents an novel attempt to balance these requirements, extending recent results in this area. The proposed system, called *Apia*[1] has the capability of dynamically select the protocols to be used, on a per message basis while allowing run-time reconfiguration of protocol stacks.

1 Introduction

As personal information devices move towards ubiquity, new challenges arise. Available bandwidth, error rate and even changes betwen disconnected and connected operation are unpredictable. In a matter of minutes, a PDA can switch from a wired connection to a infrared network and latter to a radio network.

Furthermore, "unplugged" devices still have limited processing power and memory. In order to provide an acceptable service, protocols should make an economical use of resources, avoiding expensive and non crucial operations.

The design of a single protocol suite that satisfies all these requirements is hard if not impossible. The resulting code would not provide the optimal performance on all mediums. One can, for instance, define a protocol assuming a high error rate penalizing wire connections or one expecting high latency, delaying infrared devices connections. Efficiency is often hard to conciliate with flexibility.

Several recent systems, such has for example Ensemble [4] and Coyote [1], have achieved significant improvements in the area of supporting flexible and (re-)configurable protocol architectures. Related work has addressed mainly on two different issues: avoid unnecessary processing delay by layers not interested in some events and support for run-time reconfiguration of the protocol stacks. To the extent of our knowledge, the combination of these features has not yet been provided in a clean and generic way.

The paper is organized as follows. Section 2 presents a initial motivation to our work. Section 3 describe the innovative aspects of Apia. This work is related with the current state of the art in section 4. Section 5 concludes the paper.

2 Motivation

x-Kernel [6] is one of the early influential results in the area of protocol support. Using x-Kernel, micro-protocols are layered in the form of stacks. Considering that the application layer is on the top of the stack and the physical medium on the bot-

*This work was partially supported by Praxis/ C/ EEI/ 12202/ 1998, TOPCOM.

[1] *Apia* was one of the most important roads (*Vias*) of the Roman Empire.

tom, layer properties are enforced on messages as they cross the stacks on the sending and receiving endpoints. Reusability is defined as the extent to which a micro-protocol can be used in multiple protocols [1]. The reusability of a micro-protocol is strongly influenced by its independence to other micro-protocols.

Inter-layer communication is made by the exchange of events. The implementation of the event system is framework dependent. On x-Kernel events are coded as function calls [6] while on Ensemble [4] events are data structures externally routed by the framework. After x-Kernel, several works have presented different models to bind layers to stacks and event propagation.

In Coyote [1], micro-protocols are able to create and register the events they are interested in, thus improving efficiency by reducing layer processing time. However, events are freely defined by micro-protocol programmers. Combining micro-protocols developed by different programmers imposes the existence of a name convention scheme.

Stack definition on Bast [2] is made through inheritance of existing micro-protocols, thus simplifying stack composition and micro-protocol definition. However, Bast flexibility privileges protocol algorithms, which can be changed at run-time. On the other hand, interaction between protocols, in particular the configuration of the stacks, are defined at coding time and can not be changed latter.

Horus [8] first and Ensemble [4] later have strong concerns with efficiency. Both are group communication frameworks. They propose a strictly vertical stack composition where messages must cross all the stack layers. Both frameworks predefine a fixed set of events with a known semantic. For efficiency reasons, Horus and Ensemble define particular exceptions to the strictly vertical model. With predefined sets of events and stack configurations, Horus allows some layers to be bypassed using the FAST protocol.

The state of the art represents two clearly distinct approaches: either protocol composition and event definition is flexible or protocol kernel is able to intervene on message routing, providing efficiency. We believe that modern protocol structures require a framework that supports non-strictly vertical composition but that supports also adaptive routing of messages and unrestricted event definition. To the extent of our knowledge, no previous system combines these possibilities.

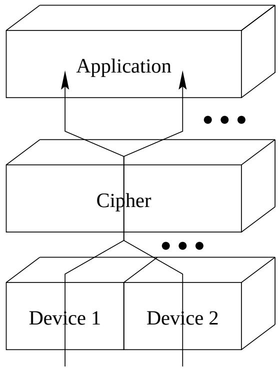

Figure 1: Two channels sharing a single cipher session.

2.1 Examples

We give two examples of protocol stacks that have non-trivial structures.

In the first example, we considered a single logical channel that exploits the availability of two different transport-level qualities of service. One of the transport protocols offers low latency delivery of short out-of band messages at a significant price, while the other channel offers a cheaper way of transferring bulk data (with much higher latency). Both QoSs are used in the context of the same logical connection and for security reasons, communication is encrypted. For efficiency and practical reasons, it is interesting that both QoSs share the same session at the ciphering layer, such that negotiation is executed only once (figure 1). On the other hand, it would be interesting to make the use of the two different QoSs transparent to the CIPHER layer, such that no specialized version of that layer would have to be written for each specific configuration.

The second example is derived from the implementation of Light Weight Groups [7], a micro-protocol that maps several user-level groups in a single virtual synchrony group (Heavy Weight Group). When two or more groups have similar membership, utilization of Light Weight Groups(LWGs) promotes resource sharing and improves performance. The LWG layer hides the existence of user-level groups from the layers bellow it and different micro-protocol combinations can be multiplexed on the same Heavy Weight Group by the LWG layer. The model of LWGs is presented in figure 2. The mapping between user level groups and virtual synchrony groups is dynamically established by the LWG protocol. This mapping may change on run-time for performance reasons.

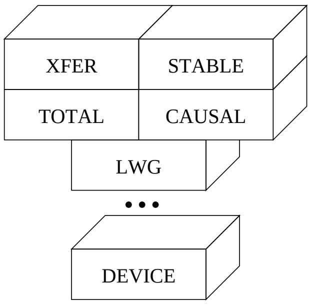

Figure 2: Light Weight Groups model

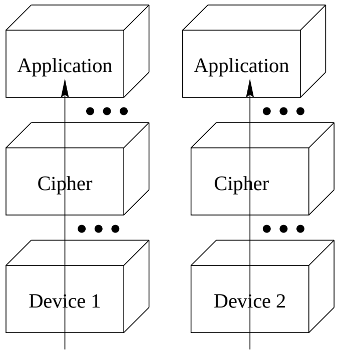

Figure 3: Two independent channels.

3 The Apia System

We define a *layer* as the implementation of a micro-protocol. All protocols implement the same *event interface*. The format and semantics of these events will be presented latter in this paper.

We define a *session* as an instance of a layer [5]. The session maintains state that is used by the layer to process events. A layer that implements ordering may keep a sequence number or a vector clock as part of each session state. In connection oriented protocols, the session also maintains information about the endpoints of the connection. Note that it is often useful to maintain several active sessions for the same layer even when they share the same endpoints: for instance, one might want to have different FIFO channels for different priority streams.

The clear separation between layers and sessions is key to our design. These two concepts can be combined to satisfy multiple QoS requirements as follows:

A *QoS* is defined as an ordered set of layers. The QoS defines which protocols must act on the messages and by which order they must be traversed. A *channel* is an instantiation of a QoS and is characterized by a stack of sessions of the corresponding layers.[2] Sessions must not be uniquely used by a single channel. Events flow between layers in FIFO ordering. The stack interfaces the communication media through a DEVICE layer. This layer is just an abstraction of any protocol outside the control of our communication kernel (for instance, TCP, UDP or IP Multicast). Figures 1 and 3 illustrates these concepts on the example presented in previous section. For clarity, only relevant layers are represented.

Sometimes, a multi-channel session may generate events that are not bound *a priori* to a given channel. For instance, the Light Weight Groups layer may exchange group management control information with its peers. To avoid ambiguities, every multi-channel session must be associated to a *default* channel that is used to route all events which are not bound *a priori* to a given channel.

3.1 Model

The model honors the distinction of QoSs from channels, having the later as instantiations of the former. QoSs are defined as an ordered set of layers. Data doesn't flow on a QoS but on specific channels. This is fundamental to our approach. In many systems, each layer individually has to perform a local demultiplexing operation to retrieve the appropriate session context in order to process each message. In our system, like in Ensemble, the demultiplexing is performed only once, when the

[2] At a finer level of granularity, one could also consider different parameterizations of each protocol as an independent QoS.

messages enters the system and the target channel is retrieved.

Upon creation, a channel is as an array of "typed empty slots". Each of this slots must be filled with a session of the layer specified in the QoS for that position. Sessions can be bound to the slots by the channel creator (explicit binding) or by other sessions (automatic binding). By default, new sessions will be bound to the remaining slots.

Using explicit binding, the channel creator associates specific sessions to the channel. The sessions may either be already in use by another channels or may be intentionally created for the new channel. Explicit binding enables the user intervention on the channel configuration. For instance, on our first example a single session of the CIPHER layer should be explicitly bound to all channels in the stack.

Using automatic binding, the channel creator delegates on already bound sessions the task of specifying the remaining unbound sessions. This capability is important for layers such as Light Weight Groups [7] which are responsible for selecting the lower sessions according to the target Heavy Weight Group (depending on run-time mapping decisions [3]).

Channels may also be *cloned* and *renewed*. The former operation accepts a channel and creates a new one where every session is shared with the input channel. The later accepts a channel, disposes all the existing sessions and creates new ones for all the slots.

Channel may have *attributes*, string pairs in the form ("attribute name","attribute value"). Attributes are opaque to the framework. Sessions may query channels for attribute values. This allows sessions to share channel specific parameters such as the port number on a TCP/IP stack or the group name on a virtual synchrony stack.

User arguments are addressed to layers and delivered to sessions at channel startup time.

3.2 Layer composition

A protocol is defined as *channel-aware* if its algorithm recognizes and acts differently upon reception of events flowing from different channels. As noted before, it is desirable to have the CIPHER layer to be channel-unaware while the LWG layer is, by definition, channel-aware (it manipulates the channel' sessions to fit its mapping decisions). Protocol reusability could be limited in Apia if channel-awareness would be mandatory.

We propose a model where sessions may or may not be channel-aware. A channel identifier *cid* is presented to layers on every event delivery. This value can be considered opaque by the layer, maintaining the usual stack semantic on single channel frameworks (send up/send down calls). Layers should use the default channel, defined at stack creation time, to exchange control information unrelated with any specific channel flow. On the other hand, layers should propagate the opaque *cid* value to send events causally related to events already bound to a channel, such as replies or confirmations. Many of the layers that can be found in existing stacks are channel-unaware.

Channel-awareness allows greater configurability possibilities. Sessions can use the channel information to learn the available QoSs and then choose which channel to use for their own events or even take different actions according to the properties of the available channels.

3.3 Events

Some frameworks support only a pre-defined set of events. The knowledge on the semantic of each event is then used to implement event-specific optimizations. We say that these frameworks support a *closed* event model. Closed models make very hard to use the framework in contexts that are different from the ones they were initially targetted, even if these differences are small. The pre-defined set of events may be not enough to express in an efficient manner other protocols. On the other hand, a framework that uses an *open* model, allows new events to be defined. Naturally, it is harder to implement optimizations in event routing when the set of events is unknown *a priori*. The model presented here tries to merge the advantages of a closed model with the flexibility of an open model.

Events in Apia are object oriented data structures, all descendant of the `ApiaEvent` main class. The main `ApiaEvent` class has a set of fundamental attributes: the boolean attribute `sendable`, stating whether the event should be forwarded by the network, `direction` and `channel`. New events can be created by deriving from a previously defined event class (in particular, directly from `ApiaEvent`).

In order to allow future event refinement, event types tests are always performed on the weakest class satisfying the desired requisites. The goal is to enforce event specialization using heritage. This way, older protocols, unaware of the event new attributes will continue to execute correctly.

3.4 Run-time compatibility check

At QoS definition time, layers are requested to declare three event-related sets:

Ψ_l Events layer-l requires to provide a correct execution

Φ_l Events layer-l is willing to receive

Γ_l Events layer-l will generate

A **correct stack** is one for which every element of Ψ belongs to Γ (where Ψ is the union of all Ψ_l sets on the stack). QoS definitions not respecting the above will return an error and will not be created.

Although it is not a complete stack validation tool, the model improves previous frameworks in this respect. For sanity, it is expected that every event declared in a Ψ_l set is also declared on the corresponding Φ_l set.

3.5 Efficient event routing

Under normal execution, only a few protocols add valuable information to messages [1, 8, 4]. For example, in a group communication stack not every layer is interested in receiving view change information. Horus's Fast protocol bypasses a predefined set of layers under specific conditions. Our approach allows layers to explicitly state which events they are interested in.

The event sets specified at QoS definition time are used for optimizing execution in run-time in the following way.

Φ sets are used at QoS definition time to construct event routing tables. For each event $e \in \Gamma$, an ordered set of layers will be constructed containing those that mentioned event e in their Φ set. Upon channel creation, corresponding QoS event tables will be ported to the channel. This operation will map QoS layers on instantiated sessions.

Event routing is static for each channel. On event arrival, the event table defined for its type at the target channel is associated with the event. Thus, an event is able to find the next session to be visited by simply keep a pointer to an array. In Apia, most of event routing overhead is clearly pushed to QoS definition time, which is expected to happen at application startup. This way, the introduced flexibility will not compromise execution performance.

4 Related Work

Current frameworks tend to favor one of this two conflicting goals: performance or flexibility.

x-Kernel [6] was pioneer on the separation of protocols from sessions, which is the base model for our work. However, x-Kernel provides no support for flexible event exchange, allowing only message passing and control functions as inter-protocol communication tools. This has been perceived has a constraining factor in the design of micro-protocols [1, 8]. Furthermore, on x-Kernel, message routing within the stack is determined on a per-layer basis. Each layer has to demultiplex each message explicitly and to keep state to ensure that the message is forwarded to the appropriate next layer. Although allowing compatibility with existing protocol stacks such as TCP/IP, the model introduces additional overhead by requiring every layer to analyze message headers.

Coyote [1] offers no hierarchical composition of layers. Events on this framework are intended to be handled in parallel by all layers and have no syntactic or semantic restriction. Concepts as "session" and "channel" have no representation. Reusability is hard to achieve with Coyote as it requires the existence of event names conventions between protocol programmers.

Bast [2] is the one that most resembles a "plug and play" framework by having protocol dependencies more deeply expressed, namely at protocol coding time. Providing different Qualities of Service simultaneously in Bast is easy as long as all of them share the same subset of protocols with the stronger QoS. Run-time configuration in Bast is limited to the strategies that implement each layer and not to the stacks (i.e., which layers participate in the stack).

Dynamic event routing and multiple Qualities of Service were both focused by Horus [8] with strong restrictions. The former was mainly supported by two assumptions: event types and protocol semantic are known by the framework [9]. Fast (Protocol Accelerator) layer allows optimizations that are not currently supported by the Apia model: events are processed by layers depending on the session state. On the other hand Fast is not generic and can only be used in a very limited fashion: the set of protocols to be bypassed is well defined. Concurrent Qualities of Service in Horus were developed in the context of Light Weight Groups [7]. The LWG layer allows several user-level stacks to be merged at the lower layers for performance reasons. The model presented in this paper is not restricted to LWGs. Apia can support layers implementing LWG protocols but also more complex composition scenarios.

Ensemble [4] was strongly focused on formal protocol validation. In order to accomplish stack correction guarantees, strong restrictions where made. Although it is a follow up from Horus, Ensemble can't handle in an efficient manner Light Weight Groups neither different concurrent qualities of service. In Ensemble, run-time reconfiguration can only be accomplished by total stack reconstruction.

5 Conclusions

This paper introduces Apia, a protocol kernel that tries to balance the flexibility in protocol composition with run-time efficiency. With Apia, protocol stack designers specify which events are produced and subscribed by each layer. In run time, the application may construct the sequence of protocols layers that is needed to enforce the desired semantics. Specialized event dispatchers for each QoS ensure the efficient routing of event in the kernel. Instances of Qos, called channels, may share sessions. This allows the construction of complex stacks, where different channels may share common properties.

References

[1] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, and Wanda Chiu. Coyote: A system for constructing fine-grain configurable communication services. *ACM Transactions on Computer Systems*, 16(4):321–366, November 1998.

[2] B. Garbinato and R. Guerraoui. Flexible protocol composition in Bast. In *Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS-18)*, pages 22–29, Amsterdam, The Netherlands, May 1998. IEEE Computer Society Press.

[3] K. Guo and Rodrigues. Dynamic light-weight groups. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'17)*, Baltimore, Maryland, USA, May 1997. IEEE.

[4] M. Hayden. *The Ensemble System*. PhD thesis, Cornell University, Computer Science Department, 1998.

[5] N. Hutchinson and L. Peterson. Design of the x-Kernel. In *Proceedings of the SIGCOMM'88: Communications Architectures and Protocols*, Stanford, USA, August 1988. ACM.

[6] N. Hutchinson and L. Peterson. The x-Kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[7] L. Rodrigues, K. Guo, A. Sargento, R. van Renesse, B. Glade, P. Veríssimo, and K. Birman. A transparent light-weight group service. In *Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems*, pages 130–139, Niagara-on-the-Lake, Canada, October 1996.

[8] R. van Renesse, Ken Birman, and S. Maffeis. Horus: A flexible group communications system. *Communications of the ACM*, 39(4):76–83, April 1996.

[9] Robert van Renesse. Masking the overhead of protocol layering. In *Proceedings of the 1996 ACM Conference on Applications, technologies, architectures, and protocols for computer communications*, pages 96–104, Palo Alto, CA USA, August 28–30 1996.