Murdoch
UNIVERSITY

MURDOCH RESEARCH REPOSITORY

http://researchrepository.murdoch.edu.au/9797/

Joolia, A., Coulson, G., Blair, A.T., Gomes, A.T., Lee, K. and Ueyama, J. (2003) *Flexible programmable networking: A reflective, component-based approach.* In: PGNet 2003 Conference, 16 - 17 June, Liverpool, UK.

It is posted here for your personal use. No further distribution is permitted.

Flexible programmable networking: A reflective, component-based approach

Ackbar Joolia, Geoff Coulson, Gordon Blair, Antonio Tadeu Gomes, Kevin Lee, Jo Ueyama
Computing Dept, Lancaster University
[joolia,geoff,gordon,gomes,leek,ueyama]@comp.lancs.ac.uk

Abstract

The need for programmability and adaptability in networking systems is becoming increasingly important. More specifically, the challenge is in the ability to add services rapidly, and be able to deploy, configure and reconfigure them as easily as possible. Such demand is creating a considerable shift in the way networks are expected to operate in the future. This is the main aim of programmable networking research community, and in our project we are investigating a component-based approach to the structuring of programmable networking software. Our intention is to apply the notion of components, component frameworks and reflection ubiquitously, thus accommodating all the different elements that comprise a programmable networking system.

1. Introduction

Existing networks have a lot of limitations in the sense that they are not very flexible and easily adapted to growing demands in terms of services and new technologies. Therefore there is an increasing demand for openness and programmability in the actual networks. The idea is to be able to 'open' these networks up and rapidly program them in a safe and secure way, to adopt new services, protocols, architectures and constraints. Programmable networking environments differ from other networking environments by the fact that they can be 'programmed' from a basic set of APIs to provide new services, or offer the capability to inject code into network nodes so that their behaviour can be changed accordingly to what is being required from applications, users or organizations.

In our approach (NETKIT Project), we are investigating the use of *reflection*, *components* and *component frameworks* [3] to come up with a software model that can be applied ubiquitously at all the different levels in the programmable networking environment, from fine-grained low-level in-band packet handling, to active networking execution environments, to signaling. Reflection provides more openness and in a principled manner, rather than in an ad-hoc manner. In our project, we are applying our previous work on reflection, or more precisely, reflective middleware, in the area of programmable networking. Reflective middleware platforms have made significant progress in the past few years (see, e.g. DynamicTAO [1], and LegORB [2]), and they have the inherent property of being platforms that can be flexibly configured, run-time adapted and reconfigured, especially in terms of non-functional properties like timeliness, resource management, transactional behaviour and security.

In the remainder of this paper, we describe briefly our reflective middleware approach and discuss its potential for facilitating programmable networking in terms of more flexibility, deployment, and management purposes followed by an overview of our ongoing and future works.

2. Background

Reflection

Reflection [4] is increasingly being applied to a lot of areas like language design, e.g. Java Core Reflection API [5], operating system design [6], distributed systems [7],[8], concurrent language [9] and importantly to us, the area of reflective middleware[27]. Reflection provides the capability to overcome the 'black box' philosophy of existing platforms by opening up the underlying structure and accessing it. Through reflection and the appropriate operations, the internal details of platforms can be inspected, and it is also possible to change/insert behaviour of these platforms by exposing the underlying implementation.

Components

According to Szyperski [3], a *software component* is "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties". Component technologies rely a lot on composition rather than relying on inheritance for the construction of a software application. The advantage with composition and components is that new services can be readily added by the process of assembling the components together, and components can be re-used over and over again, to come up with different application logic. We believe that by making use of a component-based approach, we can populate the programmable networking environment uniformly with components being applied at all levels, from low-level OS-like system support to in-band packet handling, to active networking execution environments to high-level coordination and signaling. The presence of explicit contracts in terms of provided and required interfaces i.e dependency between specific interfaces and receptacles, together with composition process can provide on-demand loading and unloading of components (as services), and this can be very helpful in terms of resource handling, security and safety, management purposes, configuration and reconfiguration of the system.

Component Frameworks

The other technology that underpins our work on NETKIT is the use and application of component frameworks (CFs). Szyperski defines component

frameworks as "collections of rules and interfaces that govern the interaction of a set of components plugged into them". Essentially, component frameworks are reusable architectures that embody domain-specific constraints and strategies for composing components. Making use of component frameworks brings a lot of inherent advantages which helps with our design:

Component frameworks provide a means of enforcing desired architectural properties and invariants by constraining the interactions among their plug-ins (which are components assembled together through composition) in a domain-relevant manner. Therefore, they represent a very viable way to impose our rules and conditions on the way components interact with each other.

- They increase systems understandability and maintainability, and simplify component development through design reuse.
- They can be used to structure the architecture of a system into a set of specialized and focused domains.
- They can be used to constrain the scope of dynamic reconfigurations and ease the task of integrity maintenance.

Our previous research on the above technologies indicates that reflection, software components and component frameworks are highly complementary. Reflection provides the necessary level of openness to access the underlying platform architecture while components can be used for structuring the architecture appropriately. Configuration and reconfiguration of the underlying architecture is inherently possible due to the compositional nature of components, and finally, appropriate constraints and strategies can be imposed on these architectures through component frameworks.

3. The Programmable Networking environment

This section gives a broad representation of the programmable networking space design. This is just a brief overview of the environment, and more insight can be found in [14, 15, 16, 17, 18, 20]. The reference stratification depicted below is representative of the different areas where programmable networking projects and efforts are being carried out.

3.1 The different paradigms

Historically, there have been *two* main paradigmatic approaches to the provision of openness and programmability in networks:

- Active networking paradigm (see e.g. [10]) – where special packets called 'capsules' or smart packets carry programs that can be executed on 'active nodes'. These active nodes are usually programmable routers. Active networking usually operates in a Java-based environment [10].
- Open signaling paradigm (see e.g. [11]) – in which routers export 'control interfaces' through which they can be remotely (re)configured by out-of-band, application specific signaling protocols.

However, recently the state of the art is that the paradigms are beginning to converge. For example, some open signaling systems now support downloadable modules on routers and are therefore more dynamic. This leads to a third approach which has become quite popular, and which we call *out-of-band active* paradigm. These systems differ in their support for kernel vs. user space modules, and in the way in-band functions can be adapted/managed/configured.

Active networking systems tend to be the most dynamic approach to programmable networking since they operate at a very-fine grain (capsules), but they are not as easy to deploy as the other approaches, tend to be language specific (e.g. Java) and prone to security threats. Open signaling is less dynamic and more coarse-grain (since it uses interfaces) but it is easier to deploy (especially for complex distributed services), easier to secure, and has better performance than active networking. The third paradigm, out-of-band active, inherits the properties of both classic approaches in terms of deployment, management, flexibility and security.

3.2 Stratification of the programmable networking design space

Figure 1 shows a reference stratification of functionality in the programmable networking environment. It should be noted that we have used 'stratum' instead of layer to avoid confusion with layered protocol architectures.

Stratum 1, *hardware abstraction,* contains the minimal OS-like functionality like threads, memory allocation, scheduling, library loading, and access to network hardware) which provide higher-level network programmability and are present on all participating nodes (router) of the network. The stratum tries to provide a uniform structure and services to the upper strata, thus masking the low-level complexities and hardware heterogeneity that can exist at that level. This is quite interesting to our work because we plan to experiment with software PC-based routers and specialized programmable network processors like the IXP1200 [12] and the IBM PowerNP [13]. Services present at this layer would determine QoS capabilities (e.g. predictability, throughput and latency) and flexibility of the system at the higher level.

4- coordination
3- application services
2- in-band functions
1- hardware abstraction

Figure 1: Programmable networking design space

Stratum 2, *in-band functions*, represents the processing and handling of all of the packets going through the system/router. For example, the traditional functionalities present in routers for packets, like packet filters, classifiers, schedulers, shapers, validators and so on). Since they occur at a low-level, are in-band and fine-grained, these functions are performance critical, to ensure that software and hardware are complementing each other speed wise, and hence need to be designed and implemented with great care.

The stratum 3 *application services* comprise coarser-grained functions/processes that are less performance critical and act on pre-selected packet flows. These are more flexible and dynamic thus offering application-specific control, management and deployment of services e.g. introducing new media-filters, or security control. They usually occur in the active networking execution-environment [10].

The 4th stratum, *coordination*, populates out-of-band signaling protocols that enable distributed coordination like configuration/adaptation of the lower strata. Examples are protocols that coordinate resource allocation (e.g. RSVP) on a set of routers participating in a dynamic VPN. Systems like Genesis [14], Draco [15], Darwin [16] employ such an approach.

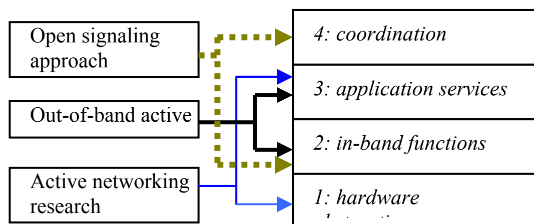

Figure 2: Paradigm mapping onto our stratification

Figure 2 depicts a general summary of how the paradigms discussed above map onto the different strata. It is interesting to observe that much programmable networking research addresses only a subset of the strata. Furthermore, an overview of the different projects/research going on the field of programmable networking shows that systems that tend to be self-consciously paradigm-independent usually address only stratus 1 and/or stratum 2. For example, the Click modular router [17], Netbind component binding system [18], Washington University pluggable router framework [19], and the IEEE P1520 router component model [20] are all targeted at stratum 2.

4. Implication

Considering the significant research occurring in the programmable networking environment, we argue that most work to date has focused on specific and limited areas, instead of overlooking the overall design space (see section 3.2). Therefore most solutions are either partial' ones or are difficult to be integrated with each other to produce more comprehensive solutions.

Hence, we believe that what is missing from the state-of-the-art is a generic framework which is *paradigm*-independent (i.e. independent of whether the approach is active networking, or open signaling, or out-of-band active), **and** also equally applicable to all the strata in Figure 1. while being able to provide unified and explicit support for implementation, deployment, reconfiguration and system evolution. Ideally, this framework should also be *programming* language and *platform* independent.

Having a ubiquitously-applied component model promises a uniform environment for the development, deployment, configuration, reconfiguration and evolution of programmable networking systems at all levels and granularity. For example, functions as diverse as in-band packet handling and signaling can be developed, deployed, configured and reconfigured in a common manner and can rely on common support such as dynamic remote instantiation, reflective services, and security and safety support. More specifically, software on a single node can be analysed separately as a composite for consistence, security and integrity purposes. Our approach enables appropriate third-party components to be assembled to achieve the desired functionality (e.g. functionality will vary for various systems like embedded wireless vs. large-scale routers vs. specialized routers).

5. Ongoing and future work

In the NETKIT project we are addressing the provision of a generic toolkit support for the programmable networking environment, which takes in account the above considerations. NETKIT is based on OpenCOM [21] (see below), which is a simple, lightweight, efficient, and language-independent software component-based computational model resulting from research work on reflective middleware at Lancaster University. OpenCOM is a component-based, fine-grained, programming and platform-independent computational model, and we assert that these characteristics are very well suited to be used in implementing systems from the paradigms discussed previously: active networking, open signaling and out-of-band active.

The idea is to populate all the strata in the programmable networking environment with component frameworks (CFs) based on OpenCOM. Our aim is to deploy NETKIT in a heterogeneous environment to validate our claim of a general approach to programmable networking, while at the same time, trying to maintain as much commonality/uniformity as possible without compromising flexibility, (re)configurability or performance. As such we plan to illustrate NETKIT working not only in a standard PC-router environment,

but also in a specialized programmable router/network processor environment - here we are targeting the Intel IXP1200 [12] - and in ad-hoc networking environments.

5.1 OpenCOM

OpenCOM [21] adopts a computational model embodying the key concepts: *component, interface, receptacle, local binding and container* (see below for detail). This is illustrated in the figure 3, which shows a local binding between two components; or more specifically, between a receptacle and interface of two different components.

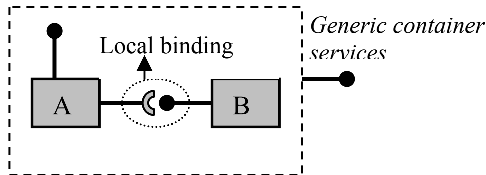

Figure 3: Receptacle from component A *local binding* to Interface from component B

A receptacle expresses a unit of service requirement, declaring an explicit dependence of one component on one another. This property of receptacles means that when a component is dynamically loaded, it is possible to determine which other interfaces (hence components) should be present for it to work properly. Containers provide a run-time environment for a set of component instances that are mutually participating in local bindings. They provide generic services for dynamically loading and unloading components, and for creating and destroying local bindings. They also offer the possibility to impose security and safety constraints wherever possible and appropriate.

OpenCOM provides support for reflection through the presence of three meta-models. First, there is the *interception* meta-model which allows the programmer to insert code (as interceptors) before or after the invocation of a particular interface to carry out new behaviour. Secondly the *introspection* meta-model which allows the inspection of types of interfaces/receptacles at run-time, and thirdly the *architecture* meta-model which allows the programmer to view the internal structure of a system (through behavioural/structural reflection), which is represented as a topology of components in a *system graph*.

These meta-models can be usefully applied to the Programmable networking domain. For example, the use of interceptors to do dynamic switching in and out of components to determine a new architecture, packet counting, introducing logging and security checks for access control, QoS monitoring at the higher-levels like Stratum 2/3. The system graph provides information on all the receptacles/interfaces/bindings which e.g. can be used to guide the reconfiguration of forwarders and classifiers in routing systems. The meta-interface is very useful in enabling CFs to check the type of interfaces offered by plug-ins e.g. in a router, for safety reasons, to prevent the whole system from crashing and maintaining integrity.

A more exhaustive and complete explanation of OpenCOM can be found in [21] [22].

5.2 OpenCOM development

For our project, we have been working on the current OpenCOM to adapt it to our needs because although the current implementation satisfies a lot of requirements (has reflective capabilities, component-based, lightweight) discussed previously, it is still deficient in certain key areas as discussed below.

Firstly, we have been working towards the porting of OpenCOM on Linux. OpenCOM initially ran only on MS Windows platforms and was quite closely attached to MS COM. The reason for moving towards Linux is because the latter has better networking support (and is more open) than Windows. We proceeded by making use of XPCOM [23], which is a lightweight component model that is built atop of the core subset of Microsoft's COM; XPCOM is interesting because it claims to be platform-dependent, and has been deployed on over 15 different platforms, Windows and Linux inclusive. This has involved freeing OpenCOM from its MS dependencies, and adapting it to be more platform-independent by making use of facilities offered by XPCOM.

Currently, we have successfully ported OpenCOM to Linux our next aim is to work towards removing the dependency on XPCOM, and to define a very minimal set of (XP)COM-core functionalities. This would provide us with a more self-contained minimal OpenCOM implementation. This is quite important to us, because our objective is to use OpenCOM on the specialized network processors (e.g. Intel IXP1200[12]) which usually have sparse resources and may even have no operating system environment.

We are also investigating techniques to optimize OpenCOM performance (e.g. temporarily bypassing vtables by using partial evaluation techniques [24] to reduce overhead of cross component calls) and to support components written in bytecode languages like Java and C# (this is not supported at the moment in OpenCOM) to achieve seamless co-existence on both compiled and bytecode components in the same container.

5.3 Router CF

While our aim for NETKIT is to apply components and OpenCOM-based CFs uniformly at all levels of the programmable networking environment (in the different strata), we currently have only a sparsely populated space with components and CFs. We have been looking into a range of CFs and components for the fast-path and per-flow packets handling areas (stratum 2), plus some

specific signaling/coordination oriented CFs and components.

We have focused on designing a stratum-2 CF which we call the 'Router CF' [26], which accepts OpenCOM components as plug-ins that perform arbitrary user-defined packet-forwarding/processing functions.

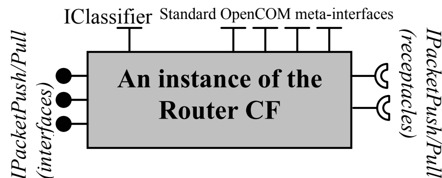

Figure 4: Router CF component

We have defined a set of rules and constraints that components within the Router CF should conform to. These rules are enforced at run-time by the CF:

- There are two main base-level interface types *IPacketPush* and *IPacketPull* which enable both "push" and "pull" oriented flow of packets, and compliant components *must* satisfy the appropriate combination of these interfaces/receptacles.
- Compliant components may support a meta-level interface *IClassifier*, which offers operations to install packet filters. Such components are called *classifiers* and their inputs and outputs are appropriately specified in terms of *IPacketPush* and *IPacketPull* for dealing with packets.
- In the case where compliant components are composite, their internal constituents must conform to the CF's rules. Additionally, there is a *controller* component (Figure 5) present to manage and configure the internal constituents of composite components.

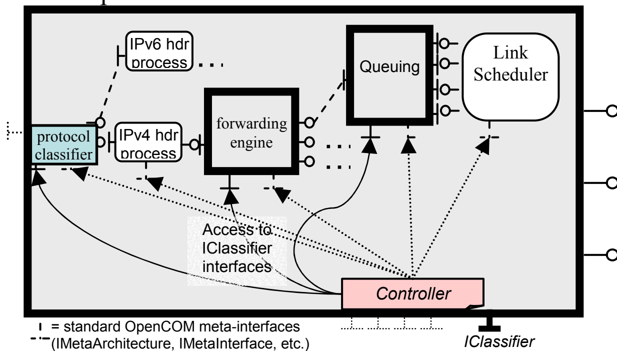

Figure 5: A Router Instance (composite components)

The CF also has the following key properties:

- Single components can support dynamic addition/removal of constraints through the use of interceptors as outlined in the discussion of OpenCOM. These constraints are policed by the controller.
- The CF can also exploit OpenCOM's resource CF [22] to control the resourcing of tasks (e.g. in terms of allocation of memory to new packet queing components, thread allocation for forwarders) and mapping of these resources to their constituent components.

5.4 Issues with PC-based implementation

While our conceptual modeling of the Router CF is in a fairly mature state, a lot of crucial factors have to be taken into consideration in a 'real-world' implementation. In particular, when concerning PC-based routers:

- We will certainly face the problem of untrusted components. Since our approach is a component-based one, we expect to use third-party components to illustrate the flexibility of our system, which means we need to take safety and security into consideration. As such, we are thinking to manage these untrusted components remotely by the parent composite (in a different address space), or use techniques like sandboxing to prevent malicious tampering with the whole system.
- the implementation will be running on the main processor and as such, it is crucial to decide which services/operations/processing should be given priority. Hence, this involves a good evaluation of how resource partitioning and thread management can be done, again in a componentized-way.
- the implementation is being based on a Linux environment, and Linux being 'open', we intend to make full use of its networking capabilities available. As such, we are working towards having a stratum-1 support for the PC-based router, which would provide functionalities like thread management, scheduling, library loading and memory allocation.

At the first stage, we have started implementation with an application-level gateway, with very basic functionality. It is being implemented in terms of Socket calls, just to illustrate a simple data-path, with a null forwarder i.e. a packet coming into the gateway and then going out without any processing done. And then, gradually, we plan to build on this prototype, to come up with more complicated forwarders (e.g. IP protocol processing, MPLS processing, etc…). Since our aim is to illustrate the configuration/reconfiguration/flexibility of the router in terms of components composition, basic IP functionality will be treated as services to the system, to make the system open for other protocols. At present, the interface to the Gateway prototype is socket-based packet delivery, and we hope to come up with a component which would interface transparently to the NIC hardware and even to the IXP1200 board in a later phase.

The Router CF is in an early implementation phase, and the design might be subject to some slight change in the future. We hope to come up with a working implementation soon on a PC-based router, and to be able to validate its performance and flexibility.

5.5 Working with the IXP1200

We are in the process of implementing the Router CF on the Intel IXP1200 network processor (see [12] for a more complete description). Unlike the implementation of the Router CF on the PC-based platform which runs entirely on the main processor, the implementation of the Router CF on the IXP1200 can take advantage of the specialized hardware. The main characteristic of the IXP1200 is the ability to perform packet processing on different parts of the IXP1200 architecture.

For example on our proposed PC-based router implementation, we plan to implement both the data plane and control plane on the host processor, whereas the IXP1200-based router implementation will have the data plane running on the Microengines while the control plane can run on the StrongARM processor and also the host PC's processor. This is illustrated in figure 6, which shows that fast-path processing occurs on the Micro-engines whilst out-of-band processing like router-changes or reconfiguration of the Router CF occurs on the StrongARM or host-PC processor.

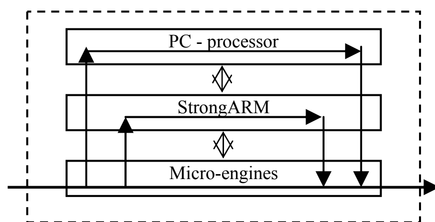

Figure 6: Possible switching paths through the processors

A second issue we are looking at is the need for support at Stratum-1 (figure 1) and the development of OS-related CFs to support our higher level Router CF. These will take the form similar to those in THINK [25] and will allow OpenCOM like-facilities at the level of resource reconfiguration. We perceive that taking this path will lead to increased performance, flexibility and robustness (security) of our architecture.

6. Conclusion

We believe that although there has been significant research in the field of programmable networking, the presence of a ubiquitously-applied component model at all levels of the strata, at any granularity and at any appropriate language is still missing. What we illustrated in this paper is our approach to a more integrated structuring of programmable networking software, and we intend to achieve our goal by making use of OpenCOM, which is a reflective lightweight language-independent model.

Most of our work so far has involved adapting OpenCOM to our needs for NETKIT and dealing with its omissions and restrictions. We have designed a CF for stratum-2 which is in the implementation phase at present, and we hope to come up with OS-like components and an appropriate CF at stratum-1 level very soon, in conjunction with the component placement on the IXP1200 processor.

Our project is in quite an early phase, and our aim is to populate all the strata so as to illustrate that our generic approach works well, in terms of performance, flexibility and transparency, in the future.

References

[1] Kon, F., Román, M., Liu, P., Mao, J., Yamane, T., Magalhães, L.C., and Campbell, R.H., "Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB". IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000), New York. Apr 3-7, 2000.

[2] Roman, M., Mickunas, D., Kon, F., and Campbell, R.H., "LegORB", IFIP/ACM Middleware'2000

[3] Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1998.

[4] Kiczales, G., J. des Rivi航s, D.G. Bobrow, "The Art of the Metaobject Protocol", MIT Press, 1991

[5] Sun Microsystems, "Java Reflection", URL: http://java.sun.com/j2se/1.3/docs/guide/reflection/index.html

[6] Yokote, Y., "The Apertos Reflective Operating System: The Concept and Its Implementation", Proc. OOPSLA'92

[7] McAffer, J., "Meta-Level Architecture Support for Distributed Objects", Proc. Reflection 96, pp 39-62

[8] Okamura, H., Ishikawa, Y., Tokoro, M., "AL-1/D: A Distributed Programming System with Multi-Model Reflection Framework"

[9] Watanabe, T., Yonezawa, A., "Reflection in an Object-Oriented Concurrent Language", In Proceedings of OOPSLA'88, p306-315

[10] The ANTS Toolkit, http://www.cs.utah.edu/flux/janos/ants.html

[11] Lazar, A.A., Bhonsle, S.K., Lim, K., "A Binding Architecture for Multimedia Networks", in David Hutchison, André A. S. Danthine, Helmut Leopold, Geoff Coulson

12] Intel IXP1200; http://www.intel.com/IXA.

[13] IBM PowerNP NP4GS3 Network Processor Solutions Product Overview, April 2001

[14] Campbell, A.T., Kounavis, M.E., Villela, D.A., Vicente, J.B., de Meer, H.G., Miki, K., Kalaichelvan, K.S., "Spawning networks"

[15]Isaacs,R., Leslie, I.,"Support for Resource-Assured and Dynamic Virtual Private Networks".

[16]Chandra, P., Fisher, A., Kosak, C., Ng, T.S.E, Steenkiste, P."Darwin:Customizable Resource Management for Value-Added network services".

[17] Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F., "The Click Modular Router"

[18] Campbell, A.T., Chou, S., Kounavis, M.E., Stachtos, V.D., and Vicente, J.B., "NetBind: A Binding Tool for Constructing Data Paths in Network Processor-based Routers"

[19] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., "Router Plugins: A Software Architecture for Next Generation Routers"

[20] IEEE P1520 http://www.ieee-pin.org/

[21] Clarke, M., Blair, G.S., Coulson, G., "An Efficient Component Model for the Construction of Adaptive Middleware"

[22] Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform"

[23] Mozilla Organization, XPCOM project, 2001, http://www.mozilla.org/projects/xpcom.

[24] Jones, N.D., "An Introduction to Partial Evaluation"

[25] Fassino, J.-P., Stefani, J.-B., Lawall, J., Muller, G., "THINK: A Software Framework for Component-based Operating System Kernels",2002

[26]Coulson G, Blair G, Gomes A.T.A, Joolia A, Lee K, Ueyama J, Ye I, "Reflective Middleware-based Programmable Networking" (to be published)

[27]Kon, F.,Costa F, Blair G.S, Campbell R, "The Case for Reflective Middleware: Building Middleware that is flexible, Reconfigurable, and yet Simple to use", 2002