# Reflective Middleware-based Programmable Networking

Geoff Coulson, Gordon Blair, Antônio Tadeu Gomes, Ackbar Joolia, Kevin Lee,
Jo Ueyama, Irvin Ye

*Computing Department, Lancaster University (UK)*
[geoff, gordon, gomes, joolia, leek, ueyama, i.ye] @comp.lancs.ac.uk

## 1. Introduction

In the past few years significant progress has been made in the design and implementation of reflective middleware platforms—i.e., platforms that, through reflection, can be flexibly configured, and run-time adapted/ reconfigured, especially in terms of non-functional properties like timeliness, resourcing, transactional behaviour, and security. Recently, we have initiated a project that investigates applying our previous reflective middleware work to the demanding and novel—for reflective middleware—area of *programmable networking environments*. In general, these environments offer the capability to inject code into network nodes so that their forwarding behaviour can be tailored on behalf of individual organisations, applications, or users. The fact that programmable networking software operates in a complex, multi-language and OS, environment and has strong requirements for dynamic deployment, 24x7 operation, managed software evolution, high performance, QoS/ resource management, adaptivity and security, makes it an ideal testing ground for the reflective middleware approach [Schmid,02].

In this position paper we outline salient characteristics of programmable networking environments and discuss how our particular reflective middleware approach, which employs a *component-based architecture* as one of its central tenets, offers the potential for more deployable, more flexible, and more evolvable programmable networking infrastructures. The remainder of the paper is structured as follows. First, §2 briefly surveys our previous work on reflective middleware, and §3 briefly characterises programmable networking environments. Next, §4 outlines our approach, and §5 discusses our progress to date. Finally, §6 analyses related work (in programmable networking, in component-based systems, and in reflective middleware), and §7 presents our conclusions and indicates areas of planned future work.

## 2. OpenOrb and OpenCOM

Our reflective middleware platform, called *OpenORB* (see, e.g., [Clarke,01], [Coulson,02]), is actually more of a framework than an ORB per se. That is, it can be used to define/ configure a range of types of middleware instances (e.g. standard CORBA, real-time CORBA, a pub-sub middleware platform, an embedded ORB etc.). These instances are defined by selecting from a set of lower-level *component frameworks*, and combining these appropriately. According to Szyperski [Szyperski,98] component frameworks (hereafter, CFs) are "*collections of rules and interfaces that govern the interaction of a set of components 'plugged into' them*". We have implemented runtime CFs for a wide range of middleware-oriented functionality domains including pluggable protocols, thread management (offering pluggable schedulers), buffer management, media filtering, and extensible binding types [Coulson,02]. Essentially, these CFs provide structure for domain-specific configurations of components, and encapsulate domain-specific rules and (meta-)interfaces through which the application (or other, higher-level, CFs) can reflectively control and manage system configuration, deployment, reconfiguration, and longer-term evolution.

CFs accept plug-in components and, furthermore, are themselves built in terms of components; the whole structure is uniformly component-based. Components themselves are defined in terms of an inherently-reflective, fine-grained, language-independent, component model called *OpenCOM* [Clarke,01]. This supports any language whose compiler can generate stubs/ skeletons that interface to binary-level Microsoft COM-based *vtables* [Brown,99]. OpenCOM explicitly supports dependencies of components (in terms of 'receptacles', otherwise known as 'required' interfaces), and has built-in support for architectural/ structural reflection (the so-called *architecture* meta-model), and for language-independent *introspection* (this builds, in the Windows implementation, on Windows type libraries) and *interception* (the latter is very efficient as it is implemented at the vtable level). In addition, OpenCOM is closely associated with a privileged, per-address space CF called the *resources meta-model* [Blair,99], which enables fine-grained control over the resourcing of dynamically-delineable units of work called 'tasks' (tasks are typically orthogonal to the address space's internal architecture

in terms of components). In the resources meta-model, 'resources' subsume not only traditional system-level resources like threads, memory and network bandwidth, but also abstract, application-defined, units of allocation.

## 3. Characterising Programmable Networking Environments

We think of the design space of programmable networking in terms of the broad-brush stratification depicted in figure 1 (we use the term 'stratum' rather than 'layer' to avoid confusion with layered protocol architectures). In the figure, the *hardware abstraction* stratum (stratum 1) contains the minimal operating system (OS) functionality (e.g. threads, memory allocation, and access to network hardware) that must be available on any participating node (e.g. router) to support higher-level network programmability. Services in this stratum typically try to mask underlying hardware complexity and heterogeneity. Furthermore, the nature of the stratum 1 services largely determines the QoS capabilities (e.g. predictability, throughput and latency) of prog. networking software in the higher strata.

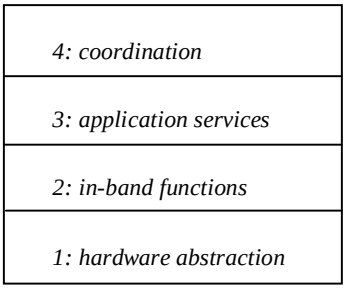


**Figure 1: Software stratification of prog. networking**

Second, the *in-band functions* stratum comprises packet processing functions (e.g. packet filters, checksum validators, classifiers, diffserv schedulers, shapers, etc.) that touch all packets. As these functions are inherently low-level, in-band, and fine-grained, this is a highly performance-critical area in which machine instructions must be counted with care.

Third, the *application services* stratum comprises coarser-grained 'programs'—in the active networking execution-environment sense [ANTS,02]—that are less performance critical and act on pre-selected packet flows in application-specific ways (e.g. per-flow media filters). Here, security is typically more of a concern than raw performance.

Finally, the *coordination stratum* comprises out-of-band signaling protocols that perform distributed coordination and (re)configuration of the lower strata. Examples are RSVP, or protocols that coordinate resource allocation on a set of routers participating in a dynamic private virtual network, as employed by systems like Genesis [Campbell,99].

## 4. Our Approach

Our recently-initiated NETKIT project, supported by the UK EPSRC and Intel Corp., is aiming to provide a generic framework for programmable networking software. Because it is based on OpenCOM, the NETKIT approach is inherently component-based, fine-grained, programming language-independent, and platform-independent and provides explicit support for implementation, deployment, reconfiguration, and system evolution.

Crucially, we are applying OpenCOM-based CFs *in all strata* of the programmable networking environment—from low-level OS-like system support, to in-band packet handling, to active networking execution environments to signaling and coordination. This should yield a 'vertically integrated' programmable networking environment, which, we argue, will offer the following benefits:

- *uniformity and simplicity*—a uniform development environment (similar tools, architectural models, etc.) and uniform run-time support for deployment, inspection, (re)configuration, and evolution;
- *can analyse software on a node as a single composite*—e.g. for consistency or integrity;
- *enables bespoke software configurations*—by selecting appropriate CFs in each stratum, desired functionality can be achieved while minimising memory footprint; trade-offs will vary for different system types (e.g. embedded, wireless devices; large-scale core routers);
- *facilitates ad-hoc interaction*—e.g. application or transport layer components can (subject to access control) straightforwardly obtain 'layer-violating' information from the link layer (this is increasingly recognised as indispensable in mobile environments).

Furthermore, we are deploying NETKIT not only in standard PC-router environments, but also in a specialised programmable router hardware environment—viz. the Intel IXP1200 [Intel,02]—and in embedded, wireless and mobile devices (we assume an 'ad-hoc' programmable networking environment). This heterogeneity is crucial in validating the claimed generality of our approach. In all cases, the challenge is to maximise the commonality without

compromising either (re)configurability or performance.

## 5. Progress to date

Our initial focus has been on designing an initial, but non-trivial, programmable networking-oriented CF. More specifically, we have designed a stratum 2 'Router CF' which accepts, as plug-ins, OpenCOM components that perform arbitrary user-defined packet-forwarding functions (we also provide 'standard' components that interface to network cards and wrap efficient kernel-user space communication mechanisms). All components (see figure 2) are required by the CF to conform to the following rules, which are checked by the CF at run-time:

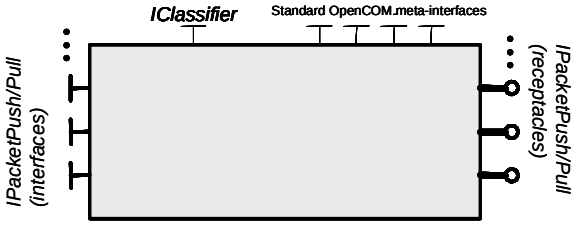


**Figure 2: A component acceptable to the Router CF**

- compliant components must support appropriate numbers and combinations of specific packet-passing interfaces/ receptacles (called *IPacketPush* and *IPacketPull*: these respectively enable push- and pull-oriented inter-component communication); it is possible to dynamically add/ remove instances of these interfaces as long as the CF's rules remain satisfied;
- compliant components may (optionally) support an *IClassifier* interface which exports an operation *register_filter()* that is used to install packet-filters; if *IClassifier* is supported, the component must honour the semantics of installed filter specifications in terms of the particular named outgoing *IPacketPush* or *IPacketPull* interface(s) on which each incoming packet should be emitted;
- compliant components may be *composite*, in which case all their internal constituents must (recursively) conform to the CF's rules; additionally, composite components should contain a so-called *controller* component that manages and configures the other internal constituents (see figure 3).

The CF also supports, on a per-component basis, the dynamic addition/ removal of arbitrary constraints. These are implemented as interceptors on OpenCOM's 'bind' primitive, and are mainly used to constrain the internal topology of composite components (see figure 3 for an example composite). Such addition/ removal of constraints is policed by an ACL managed by the composite's controller.

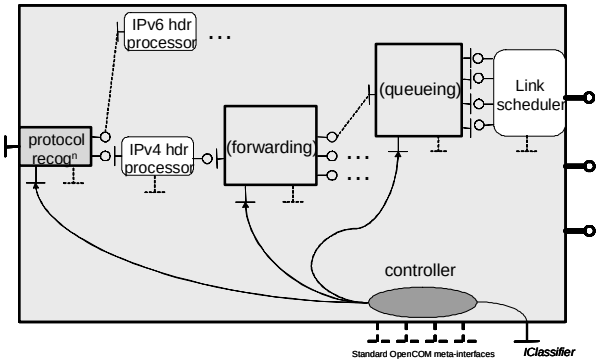


**Figure 3: A composite component accepted by the Router CF**

Additionally, to prevent untrusted constituent components from maliciously tampering with the code/ data of other constituents in the same address space, or from accidentally taking down the whole router by crashing, untrusted constituents can be instantiated, and remotely managed by the parent composite, in a separate address-space from the parent [Schmid,02] (inter-component bindings in this case are transparently realised in terms of OS-level IPC mechanisms rather than intra-address space vtables).

Finally, the CF exploits OpenCOM's resource CF so that composites (subject to access constraints) can control the resourcing of designated tasks and map these flexibly to their constituents. Components can also take advantage of our existing buffer management CF. The design of the CF is now fairly mature and we are starting to implement it for a PC-based router. We hope to be able to validate its performance and flexibility in the near future.

When the PC-based implementation is complete, we plan to re-implement the CF on the above-mentioned Intel IXP1200 programmable router [Intel,02]. This features an exotic hardware architecture comprising multiple processors—both a StrongARM control processor and Intel-proprietary 'micro-engine' processors—together with distributed/ hierarchical memory arrays. Implementing the CF on this platform will open up challenges in two main areas. First, the issue of component *placement* comes to the fore: in the PC design, we already, as described above, choose to place components in different address spaces according to security/ safety considerations; in the IXP

environment we need to additionally place components (whether on the control processor or a micro-engine) according to performance and load-balancing considerations. We think that the CF itself should contain the 'intelligence' to transparently manage this placement, but with the possibility to control/ override this via a 'placement' meta-model. Second, as there is no OS running natively on the IXP1200, we need to support the OpenCOM runtime with appropriate OS (stratum 1) functionality. This implies the design of appropriate OS-related CFs (e.g. concurrency, basic memory allocation). We are basing our work in this area on pioneering research on OS componentisation [Clarke,98], [Fassino,02].

Finally, we are progressing work to adapt OpenCOM to better support the programmable networking environment. First, we have ported OpenCOM to Linux (as the latter has better support for networking than MS Windows). This has involved freeing OpenCOM from its MS dependencies (OpenCOM was initially developed on Windows and builds on the core of COM). We have also investigated optimising OpenCOM in terms of *performance* (e.g., temporarily bypassing vtables, using partial evaluation techniques [Jones,96], to reduce the overhead of a cross-component call to that of a C function call) and *memory footprint* (e.g. our Windows CE implementation now has a footprint of only 18Kbytes). We are also working on supporting OpenCOM components written in Java.

## 6. Related Work

A number of 'component frameworks' for routers have been described in the literature. The *Click* modular router [Kohler,99] employs a fine grained C++-based component model with flexible support for the configuration (but not reconfiguration) of packet scheduling, route lookup and queue drop modules etc. The *NetBind* component binding system [Campbell,02] is similar in concept to Click but is lower-level and targeted at network processors. Both of these are stratum 2 only systems. Washington University's *pluggable router* framework [Decasper,98] is a framework for pluggable per-flow (i.e. stratum 3) modules in the NetBSD environment. The IEEE P1520 router component model [IEEE,02] is working towards a standardised, language-independent, component model for modular routers. Slightly more generally, the Knit system [Reid,00] has been used for both in-band packet handling and OS-like functions on conventional PC architectures (not on specialised programmable routers). Similarly, the VERA extensible router architecture [Karlin,01] supports in-band packet handling and OS functions on programmable router hardware. However, both VERA and Knit offer a far less general and flexible component model than OpenCOM. Also, none of the above component models are language and platform independent, have no runtime support for reconfiguration management, and are not designed for application at all system levels.

Aside from specifically router-oriented work, there has been considerable research on software componentisation in general, and on component-based middleware. Like OpenCOM, XPCOM [Mozilla,01] is a lightweight cross-platform component model that is compatible with MS's COM. However, XPCOM does not provide any special support (e.g. reflection, CFs) for dynamic reconfiguration. MMLite [Helander,98] is an operating system built using COM components. It offers limited support for dynamic reconfiguration through a 'mutation' mechanism which enables the replacement of a component implementation at run-time. However it has no specific support (e.g. in terms of reflection and CFs) to control and manage this process. Also, neither of these models has so far been targeted at the programmable networking environment. THINK [Fassino,02] is another component model that is targeted at the construction of system software. It is close to OpenCOM in its goals but does not support anything analogous to OpenCOM's CF-based structuring or support for dynamic reconfiguration.

In the middleware context, researchers have investigated lightweight reconfigurable component architectures—like us, they aim to build systems (e.g. middleware itself) in terms of components as opposed to merely supporting components on top of monolithic middleware. Prime examples are the University of Illinois' dynamicTAO [Kon,00] and LegORB [Roman,00]. These are flexible ORBs that employ a dependency management architecture that relies on a set of 'configurators' that maintain dependencies among components and provide hooks at which components can be attached or detached dynamically. OpenCOM supports a similar capability but as an integrated part of the component model.

## 7. Conclusions and Future Work

We believe that a fine-grained, reflective, language-independent component model, as discussed here, offers significant potential as the basis of an 'integrated' approach to the structuring of programmable networking software.

The most obvious potential benefit of the approach is that its ubiquitously-applied component model

promises a uniform environment for the development, deployment, (re)configuration, and evolution of programmable networking software at all levels and at any appropriate granularity and using any appropriate programming language. For example, functions as diverse as in-band packet handling and signaling can be developed, deployed etc. in a common manner and can assume common support such as dynamic remote instantiation, and standard meta-models. In addition, the approach is, in principle, sufficiently general to be applied to any programmable networking paradigms (e.g. active bettworks or open signaling), and to a range of hardware/ OS architectures. Also, because components are language independent and can be applied at a wide range of granularities, they offer a solid basis for incremental deployment of *existing* programmable networking software into a common environment.

Finally, in addition to the IXP1200-related future work mentioned in §5 above, we are currently working with Columbia University to re-engineering Columbia's Genesis system [Campbell,99]. This system supports dynamic private virtual networks, each potentially with its own semantics (addressing, routing, QoS, etc.). Apart from the opportunity to investigate the componentisation of an existing programmable networking system (with a view to enhancing its deployability and (re)configurability), this is also particularly interesting to us as an exemplar of a richly functioned stratum 4 system.

## REFERENCES


**[ANTS,02]** The ANTS Toolkit, http://www.cs.utah.edu/flux/janos/ants.html.

**[Blair,99]** Blair, G.S., Costa, F., Coulson, G., Duran, H., Parlavantzas, N., Delpiano, F., Dumant, B., Horn, F., and Stefani, J.B., "The Design of a Resource-Aware Reflective Middleware Architecture", 2nd International Conference on Meta-Level Architectures and Reflection (Reflection'99), St-Malo, France, Springer-Verlag LNCS, Vol 1616, pp115-134, 1999.

**[Brown,99]** Brown, K., "Building a Lightweight COM Interception Framework Part 1: The Universal Delegator", Microsoft Systems Journal, Jan 1999.

**[Campbell,02]** 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", 5th IEEE International Conference on Open Architectures and Network Programming (OPENARCH' 02), June 2002.

**[Campbell,99]** Campbell, A.T., Kounavis, M.E., Villela, D.A., Vicente, J.B., de Meer, H.G., Miki, K., Kalaichelvan, K.S., "Spawning networks", IEEE Network Magazine, Vol 13, No 4, pp16-29, July/Aug 1999.

**[Clarke,98]** Clarke, M., Coulson, G., "An Architecture for Dynamically Extensible Operating Systems". Proc. 4th International Conference on Configurable Distributed Systems (ICCDS'98), Annapolis MD, USA, May 1998.

**[Clarke,01]** Clarke, M., Blair, G.S., Coulson, G., "An Efficient Component Model for the Construction of Adaptive Middleware", IFIP/ACM Middleware 2001, Heidelberg, Nov 2001.

**[Coulson,02]** Coulson, G., Blair, G.S., Clark, M., Parlavantzas, N., "The Design of a Highly Configurable and Reconfigurable Middleware Platform", ACM Distributed Computing Journal, Vol 15, No 2, pp109-126, Apr 2002.

**[Decasper,98]** Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., "Router Plugins: A Software Architecture for Next Generation Routers", ACM SIGCOMM 98, 1998.

**[Fassino,02]** Fassino, J.-P., Stefani, J.-B., Lawall, J., Muller, G., "THINK: A Software Framework for Component-based Operating System Kernels", Usenix Annual Technical Conference, Monterey (USA), June 2002.

**[Helander,98]** Helander, J., Forin, A., "MMLite: A Highly Componentized System Architecture", 8th ACM SIGOPS European Workshop, pp96-103, Sintra, Portugal, Sept 1998.

**[IEEE,02]** IEEE P1520 Proposed IEEE Standard for APIs for Networks, http://www.ieee-pin.org/.

**[Intel,02]** Intel IXP1200; http://www.intel.com/IXA.

**[Jones,96]** Jones, N.D., "An Introduction to Partial Evaluation", ACM Computing Surveys, 28(3), pp480-504, Sept 1996.

**[Karlin,01]** Karlin, S., Peterson, L., "VERA: An Extensible Router Architecture", IEEE Conf. on Open Architectures and Network Programming, OPENARCH 2001, Anchorage, Alaska, pp3-14, Apr 2001.

**[Kohler,99]** Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F., "The Click Modular Router", ACM SOSP 1999, pp217-231, Dec 1999.

**[Kon,00]** 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.

**[Mozilla,01]** Mozilla Organization, XPCOM project, 2001, http://www.mozilla.org/projects/xpcom.

**[Reid,00]** Reid, A., Flatt, M., Stoller, L., Lepreau, J., Eide, E., "Knit: Component Composition for Systems Software", OSDI 2000, pp347-360, Oct 2000.

**[Roman,00]** Roman, M., Mickunas, D., Kon, F., and Campbell, R.H., "LegORB", IFIP/ACM Middleware'2000 Workshop on Reflective Middleware, IBM Palisades Executive Conference Center, NY, Apr 2000.

**[Schmid,02]** Schmid, S., "A Component-based Active Router Architecture", Lancaster University PhD Thesis, http://www.mobileipv6.net/~sschmid/PhD_Thesis.ps.

**[Szyperski,98]** Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1998.