

# Experiments with a Runtime Component Model

Jó Ueyama<sup>1</sup>, Geoff Coulson<sup>2</sup>, Edmundo R. M. Madeira<sup>1</sup>, Thaís Batista<sup>3</sup>, Paul Grace<sup>2</sup>

<sup>1</sup> Instituto de Computação (IC), Universidade Estadual de Campinas (UNICAMP)  
13084-971 Campinas, SP – Brazil

<sup>2</sup> Computing Department, Lancaster University,  
Lancaster LA1 4WA, UK

<sup>3</sup> Departamento de Informática (DIMAp),  
Universidade Federal do Rio Grande do Norte (UFRN)  
59072-970 Natal, RN – Brazil

**Abstract.** This paper provides a brief description of a general-purpose component model called OpenCom and also outlines the results obtained from the experiments with such a component model. The evaluation adopts a dual approach: first, we measure the inherent performance properties and overhead incurred by OpenCom. Then, we present three case studies which evaluate OpenCom's adaptive architecture in constructing system software for a wide range of domains.

## 1 Introduction

The component technology has been widely adopted to build general-purpose software at the application level by both the enterprise community and research community. For example, there are numerous component technologies for application development such as browser plugins, JavaBeans/Enterprise JavaBeans, the CORBA component model and Microsoft .NET. The success of the component approach comes from the benefits that are derived from componentisation, such as: *i*) it provides a higher degree of abstraction in software design, implementation, management and deployment; *ii*) it fosters third-party software reuse.

However, the notion of using components to build software at the *system level* (e.g. for building middleware platforms, or embedded systems) is less well established. In addition, the above mentioned benefits of componentisation appear as compelling as in this domain and this has been recognized by a growing amount of work using components for building system software. For example, Pebble [8] and Koala [12] propose the use of components for constructing embedded systems; OSKit [7], THINK [6] and MMLite [9] propose component-based OSs; proposals for component-based programmable networking environments include VERA [11], NetBind [2] and MicroACE [3]; proposals for middleware platforms include K-Component [5] and LegORB [13].

The main limitation with the above-mentioned component models is that they tend to be *narrowly targeted* and non-generic. This narrow targeting is clear in both of the following areas: *i*) the targeted systems for which they were designed (embedded systems, OSs, programmable networking environments or middleware platforms); for example, OSKit, Pebble and MMLite are exclusively targeted to build component-based operating systems; and/or *ii*) the environment at which they were intended to be deployed (e.g. most of the above mentioned models were deployed on conventional desktop machines as opposed to more non-conventional environments such as PDAs and embedded systems). As an example, VERA, NetBind and NP-Click are targeted to build programmable networking systems exclusively on the Intel IXP family routers [3].

This paper examines the experiments that were carried out with our general-purpose component model for building system software called OpenCom. Our component model supports adaptability which is demonstrated by three case studies that rely on OpenCom to build systems for a variety of domains and environments. The experiments also show that the imposed overhead is negligible and therefore well-suited for systems with stringent requirements.

In the remainder of the paper, Section 2 introduces the major features of the OpenCom component model. Following that, Section 3 provides the experiments with this technology. Finally, Section 4 offers conclusions and discusses how this work is taken further.

## 2 OpenCom's Key Features

It is important to highlight that Lancaster OpenCOM [4] refers to a previous component model targeted at middleware platforms, while OpenCom concerns our new general-purpose component model. The former, i.e., OpenCOM is built on top of Microsoft COM and has been successfully applied to constructing re-configurable middleware platforms. However, the main limitation with OpenCOM is the inability to construct systems for a heterogeneous environment.

On the other hand, OpenCom is aimed at constructing systems in a way that is independent of the deployment environment (e.g. heterogeneous environments like the IXP1200 routers (see Figure 2), which are resource constrained devices) and also aimed at constructing systems for a wide range of system domains (e.g. middlewares and operating systems). As a result, OpenCom consists of a more comprehensive programming model which is summarised below. Importantly, this is only a summary of the key features. A complete description of all features is found in [14].

The OpenCom's *kernel* is kept minimal and has only the support to deploy components of a particular style that is similar to XPCOM components. Component styles abstract between different system-level implementation of components – e.g. java components, C++ components, assembly-language components. The kernel is adaptive and extra functionalities are all incrementally deployable as demanded at runtime.

In OpenCom, one builds systems by loading components and if required connects them to other components at runtime. The connection between components is called a *binding*.

The key motivation for *loaders* is to provide multiple-loading mechanisms in the underlying deploying environment. Loaders are OpenCom components that make a wide range of component styles be deployed in a heterogeneous environment, such as an embedded device. These loaders can encapsulate the complexity in loading components in such a heterogeneous environment.

Finally, *binders* are merely OpenCom components designed to provide a wide range of 'binding mechanisms'. Through binders, developers are free to implement any binding mechanism that might be required in the underlying deployment environment. As a concrete example of this, we implemented a binder that creates bindings between primitive Microcode components. This binder is employed in our first case study outlined in Section 3.

The notion of loader and binder gives support for adaptability as developers may utilize appropriate loaders and binders in order to construct systems for aimed environments.

## 3 Experiments with OpenCom

### 3.1 Overview

This section discusses performance evaluation and overhead incurred by OpenCom. It also talks about three case studies that use OpenCom to build systems for a variety of domains. The aim of the case studies is to demonstrate the generality and adaptability of our component model.

The experiments outlined in this section were all carried out in a Dell Precision 340 series workstation with 4 x 1.6GHz Pentium CPUs, 512 MB of RAM, and running Linux Redhat 8.0. With regard to software, all the measurements were collected relying on an OpenCom kernel implemented in C++ for Linux.

### 3.2 Performance and Overhead

OpenCom kernel required a minimum memory overhead around 32Kbytes. In particular, this is comparable to the MMLite's kernel memory overhead which was measured as 26Kbytes. Such a memory overhead can

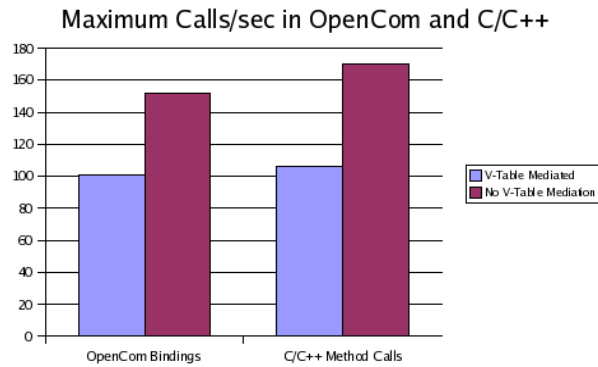
enable us to deploy OpenCom in a wide range of resource constrained devices. As a consequence OpenCom is employed to construct sensor networking systems for running in the Motes and Gumstix as outlined in [1].

Table 1 illustrates a comparison between the overhead incurred by OpenCom and C++ to load and instantiate components. The loading time includes the time taken to read a shared object library from the disk storage and extract component information from that library. This is quite comparable to that obtained when loading a null C++ object packaged in the same shared object. The instantiation includes the time taken to instantiate a null already-loaded component. The differences in this measurement can be attributed to the larger file size that is required by OpenCom components to accommodate information for the meta-data.

Operation	OpenCom	C++
Loading Time	9.8 $\mu$ s	7 $\mu$ s
Instantiation Time	0.47 $\mu$ s	0.28 $\mu$ s

**Table 1.** OpenCom and C++'s Overhead

The graph in Figure 1 illustrates the number of calls per second that was achieved by both OpenCom and C/C++. It indicates that OpenCom incurs a negligible overhead compared to that of C/C++ method calls. The number of calls per second achieved by OpenCom was obtained using two implementations of binding (i.e. one with v-table mediation and the other one without). In short, a v-table is essentially a table containing pointers to virtual functions. Calls using v-tables are more expensive given that they need an extra reference to invoke the requested method.



**Fig. 1.** Number of calls/sec in primary bindings and in C/C++ method calls

### 3.3 Case Studies

A qualitative analysis has been conducted to evaluate OpenCom in building systems independent of the target system and deployment environment. This has been verified by three case studies involving the construction of systems targeted at different domains.

The *first* case study applies the OpenCom programming model to the IXP1200 router environment [3] (see Figure 2). This router is particularly interesting for our research because it is *i*) heterogeneous (e.g. it has a number of processors, including the microengines that are specialised for packet processing); *ii*) resource poor having a small amount of memory; and finally *iii*) performance constrained (i.e. packets must be processed at line speeds). In terms of the implementation, we have developed a number of loader and binder components to deploy a wide range of component-styles in diverse environments. In particular, we implemented a Microcode binder that uses the *code morphing* approach which was pioneered by the NetBind project [2]. Essentially, a component in Microcode is bound to another at runtime by rewriting a branch instruction so that execution jumps to the desired target.

In short, Figure 2 illustrates the outline of the IXP1200 based router, which is employed in the above case study. The router is composed of *i*) a StrongARM CPU running Linux, which acts as the general control processor in the router; *ii*) an array of six so-called microengines RISC CPUs that are attached to each other and share three types of memory shown in Figure 2 (i.e. Scratchpad, SDRAM and SRAM).

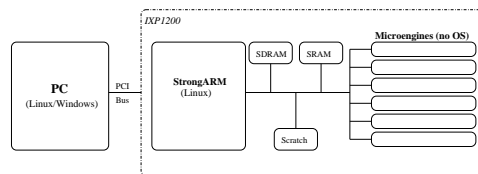


Fig. 2. Intel IXP1200 router architecture employed in our case study I

The *second* case study verifies the use of OpenCom to construct middleware for parallel environments called FlexPar. This research is funded by the São Paulo Research Council in Brazil. Essentially, this case study employs OpenCom to construct a flexible middleware that can be adapted according to the target parallel application. For example, if a developer desires to construct CSP (*Communicating Sequential Processes*) [10] based parallel software, appropriate loaders and binders are deployed to load and bind processes that implements the CSP Model. In short, CSP consists of processes and the communication mechanisms between them. It helps to avoid problems that are often encountered in multithreaded programming. In our experiments, the FlexPar kernel occupies only 60Kbytes which is suitable for most devices that runs parallel software.

Finally, in the *third* case study, OpenCom was employed to construct low-level software targeted at Sensor Motes. Sensor Motes are very primitive environments that consist of a small circuit board that hosts a number of electronic sensors and a simple 8-bit microcontroller. This case study aims to demonstrate that OpenCom can be applied in a way that is sufficient to construct low-level software running in resource-constrained devices such as a Sensor Mote. To deploy OpenCom in the Sensor Motes, the kernel had to be built on top of a simple microcontroller monitor program called *Contiki*. The implemented Contiki-kernel occupies only 30Kbytes, which demonstrates that it is portable to a wide spectrum of deployment environments. The sensor networking environment outlined in [1] also employs OpenCom to provide adaptability.

## 4 Conclusions and Further Work

This paper outlined the major experiments carried out with our general-purpose component model called OpenCom. The experiments demonstrated comparable results to that of other development tools such as C/C++. In addition, OpenCom incurred a low memory overhead which is particularly beneficial for poor resource devices such as a sensor node. The case studies in Section 3 demonstrated that the key features of

OpenCom gave the support for adaptability, which enabled OpenCom to build systems for a wide spectrum of domains and environments.

In our ongoing research we are looking at deploying OpenCom in a wider range of application domains and deployment environments. In particular we are investigating the use of OpenCom to build an architecture for networked embedded systems that encompasses dedicated radio layers, networks, middlewares, and specialised simulation and verification tools.

## Acknowledgements

Jó Ueyama would like to thank the National Council for Scientific and Technological Development (CNPq - Brazil) for sponsoring his PhD scholarship at Lancaster University (Ref. 200214/01-2). The first and third author would also like to thank FAPESP for funding the FlexPar research project (Ref. 2006/06576-8).

## References

1. *An Intelligent and Adaptable Flood Monitoring and Warning System*, September 2006.
2. A.T. Campbell, M.E. Kounavis, D.A. Villela, J.B. Vicente, H.G. de Meer, K. Miki, and K.S. Kalaichelvan. NetBind: A Binding Tool for Constructing Data Paths in Network Processor-based Routers. In *5th IEEE International Conference on Open Architectures and Network Programming (OPENARCH'02)*, June 2002.
3. Intel Corporation. Intel IXA SDK ACE Programming Framework Developer's Guide, June 2001. Part Number A71582-001.
4. G. Coulson, Blair G.S., M. Clarke, and N. Parlavantzas. The Design of a Highly Configurable and Reconfigurable Middleware Platform. *ACM Distributed Computing Journal*, 15(2):109–126, April 2002.
5. J. Dowling and V. Cahill. The K-Component Architecture Meta-Model for Self-Adaptive Software. In *Reflection 2001*, Kyoto, Japan, September 2001. LNCS 2192.
6. J.P. Fassino, J.B. Stefani, J. Lawall, and G. Muller. THINK: A Software Framework for Component-based Operating System Kernels. In *USENIX 2002 Annual Conference*, June 2002.
7. B. Ford, G. Back, G. Benson, J. Lepreau, A Lin, and O. Shivers. The Flux OSKit: A Substrate for Kernel and Language Research. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 38–51. ACM Press, 1997.
8. E. Gabber, C. Small, J. Bruno, J. Brustoloni, and A. Silberschatz. The Pebble Component-Based Operating System. pages 267–282.
9. J. Helander and A. Forin. MMLite: A Highly Componentized System Architecture. In *8th ACM SIGOPS European Workshop*, pages 96–103, Sintra, Portugal, September 1998.
10. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
11. S. Karlin and L. Peterson. VERA: An Extensible Router Architecture. In *4th International Conference on Open Architectures and Network Programming (OPENARCH)*, April 2001.
12. R. Ommering, F. Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. 33(3):78–85, March 2000.
13. M. Roman, M. Mickunas, F. Kon, and R. Campbell. LegORB and Ubiquitous CORBA. In *IFIP/ACM Middleware'2000 Workshop on Reflective Middleware (RM2000)*, pages 1–2, Palisades, NY, USA, April 2000.
14. J. Ueyama. *A Runtime Component Model for System Software*. PhD thesis, Lancaster University, 2006.