Towards Web/Java based High Performance Distributed Computing – an Evolving Virtual Machine

Geoffrey Fox and Wojtek Furmanski

Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY
{gcf,furm}@npac.syr.edu

Summary

We discuss here the emergent Web based distributed environments for HPCC on the NII with the focus on Java as an enabling technology. We start with a review of the past, presence and the near term future of the "Java phenomenon", exposed here in the background of some related previous approaches towards a distributed interpretative virtual machine architecture. Next, we discuss the anticipated role of Java in building distributed, Web based computing environments. We outline an evolutionary path from the current Web technology 'soup' towards 'all Java' systems and we illustrate this process in terms of NPAC Web technology prototypes (WebVM, WebFlow, Bridge based Collaboratory) and selected applications (CareWeb, 3D Visible Human).

Contents

1 Introduction

We discuss here the recent developments on the Internet and their implications for high performance distributed computing. We outline the ongoing process of building a World-Wide Virtual Machine and we expose Java as the most promising candidate for Web based computing. We start with a brief review of the Java history (Chapter 2) and compare it with selected related VM approaches in the past such as NeXT, Telescript and MOVIE/HPFI (Chapter 3). We point out what worked, what didn't and why, and how did the Java phenomenon happen. Next, we summarize the current status of Java, including the current products and the ongoing developments (Chapter 4). We then discuss the implications of the Java technology for the HPCC community (Chapter 5), followed by the analysis of near term future projections towards 'all Java' systems (Chapter 6). This leads in a natural way to the distributed Java environments – we summarize the enabling technologies for Java multiserver systems, we present the concept of a world-wide virtual machine (Chapter 7), evolving from the current mix of Web technologies towards an 'all Java' system, and we illustrate our ideas in terms of NPAC technology prototypes (WebVM, WebFlow, Bridge Collaboratory) and selected applications (Visible Human, CareWeb).

2 Bits of Java History

2.1 Sun NeWS A comprehensive search for Java roots would lead us back to the seminal work at Xerox Parc in '70s and systems such as Cedar, Mesa or Smalltalk. Here, we backtrack only by one generation of software and restrict this history sketch to a single generation of software visionaries such as Gosling. He and Rosenthal [NeWS89] developed in late '80 at Sun a revolutionary windowing system, NeWS (Network-extensible Windowing System) which can be viewed as a direct precursor of Java.

NeWS server was an interpreter of extended PostScript with support for windowing, (non-preemptive) multithreading and object orientation. Hence, it was a virtual machine with some aspects similar to JavaVM but also with several differences. First, no familiar C/C++-like API was offered at this time, which restricted the community of NeWS enthusiasts to an esoteric group of hackers who would program in PostScript. Second, NeWS was a much larger "machine", then even further enlarged in the X11/NeWS merger to the stage that it became unmanageable. The X11/NeWS project at Sun collapsed and X took over as a windowing standard for UNIX in late '80s.

2.2 Green Project Insight into the follow-on internal activities at Sun can be now found e.g. in the Epilog of Naughton's *Java Handbook*. A group of NeWS enthusiasts at Sun, including Naughton, Gosling and Sheridian, isolated from Sun campus and started a secret Green Project with the goal to address a VM design process from scratch and to make it simple, minimal, compact – and thereby unique. The result of this activity was Oak, a new programming language developed by Gosling.

2.3 FirstPerson, Inc. A commercial venture FirstPerson, Inc. that followed the Green Project developed a consumer electronic device Star7 with Oak's VM as an operating system and tried to market it in the emergent domain of interactive television. However, attempts to get support by Time Warner or 3DO didn't succeed – FirstPerson failed and Oak was still a solution looking for a problem in early '90s.

2.4 Oak → WebRunner → Java At this point, the World-Wide Web happened and the Green / FirstPerson team made a quick move towards repackaging their Oak —> WebRunner —> Java system as a programming language for the Internet. AWT package was written on top of X/Motif, alpha HotJava was hacked and the Java system was published in early '95. Netscape responded in fall '95 by licensing Java and the Java phenomenon was started. The future of Java remained still unclear in the winter '95, with incompatible alpha and beta releases, lack of continued HotJava support, and uncertainties regarding the mission of the just forming JavaSoft. JavaOne conference in May'96 confirmed finally to the world in terms of a spectacular series of alliances and the associated API announcements that Sun/JavaSoft is aiming seriously at Java driven software revolution with profound implications for the Internet and eventually for the whole computing.

3 Some Related Approaches

Oak/Java was not the only system inspired by NeWS and pursued in late 80s/early '90s. We review here a few other efforts towards an interpretative distributed virtual machine architecture, including our prototyping activities at Caltech and NPAC.

3.1 NeXT Computer NeXT Step software environment for the NeXT Computer was also VM based and used Display PostScript from Adobe as the windowing server. Objective C API was a step forward compared to NeWS but the NeXT computer failed as well — as perhaps a "too good too early" and/or "too closed/proprietary" box. Even if it didn't reach the sufficient market penetration, NeXT played a constructive role in shaping the Web history by stimulating inspired individuals. For example, Green Project got the green light from Sun because of Naughton trying to leave and join NeXT after the X11/NeWS failure. At the same time but on the other hemisphere, Tim Berners-Lee was using NeXT in his rapid prototyping of the early WWW software.

3.2 Telescript Another interesting VM framework, Telescript, was created by General Magic – a spin-off from Apple Computer, backed by a power alliance including AT&T, Sony, Motorola and Matsushita. General Magic developed a multiserver scripted environment that acted as a propagation medium for intelligent agents, instantiated as dynamic chunks of active mobile

Telescript. The user front-end, Magic Cap, was carefully designed to both fit the consumer electronic PDA hardware and to offer a powerful high level navigational paradigm and friendly interface for smart messaging, searching heterogeneous databases etc. Telescript agents, activated by the Magic Cap controls, would travel across the multiserver Telescript cloud, spawn other agents when necessary, perform the required information processing tasks and deliver the results to senders or receivers.

Telescript's 2–layer software architecture shared again some common features with Java: the low Telescript was a postfix syntax language acting as a virtual assembler, and the high Telescript offered a C++-like interpreted API.

The future of Telescript is unclear. The language specification and server technology, kept secret over several years, was finally published in fall '95. Intelligent agents are clearly one of the critical Web technologies to come but multiserver Java systems discussed in the next chapters offer now a viable alternative medium for agent propagation on the Web.

3.3 MOVIE We were also following a related, NeWS inspired prototyping path, with some design elements similar to Java, NeXT Step and Telescript. Our system, MOVIE (Multitasking Object-oriented Visual Interactive Environment) [MOVIE92] was given by a mesh of network servers – interpreters of MovieScript, a custom extension of PostScript. Higher level API was provided by extending the PostScript syntax towards a C++-style object orientation. The core MOVIE server offered APIs for windowing (MovieScript interface to X/Motif), networking (on top of UNIX socket library) and (custom, UNIX clock based) preemptive multithreading. The system was developed as a byproduct of a GIS project with the Army on terrain map analysis, which resulted in an imaging API, abstracted in the next stage as a Fortran90/MatLab/APL style matrix algebra API.

3.4 HPF Interpreter In the next step, we addressed the issue of distributed VM support for data-parallel programming and we prototyped a multiserver MOVIE based HPF (High Performance Fortran) Interpreter (HPFI) [HPFI93]. Based on a 3–tier architecture, HPFI translated first (tier 1) the HPF source to MovieScript + Message Passing intermediate form. This script was then dynamically distributed over a mesh of MOVIE servers running on a workstation cluster (tier 2) and communicating in terms of Express message passing and compiler runtime libraries (tier 3).

An early prototype HPFI was demonstrated at Supercomputing '93. A front-end MOVIE server was running on a workstation at Portland and offered interactive tools for editing HPF source, performing online translation, and finally uploading and running remote HPF applications on a mesh of MOVIE servers at Syracuse.

In a similar way as other interactive interpretative VM approaches, both MOVIE and HPFI projects were viewed as solutions looking for problems in early '90s. When the Web happened, we understood very well what is the problem and the market but, unlike Sun, we didn't have enough resources to quickly repackage our prototypes as Internet systems. And after Java happened, we are kept busy – watching and learning Java...

3.5 Summary When compared with some related previous approaches that failed, today's Java is a simpler and smaller system. It matched perfectly the Web niche for simple but interactive programmable content and succeeded. Other systems tried the full solution and run out of support or market focus or both. However, as discussed in the following chapter, Java is aiming now at the total solution for the Internet programming and we expect several design ideas from the previous related systems to reappear in the evolving Java framework. For example, a higher level scripting language seems to be a necessary add-on for rapid prototyping. Is JavaScript an adequate model for this purposes is yet to be seen. NeXT Step can provide design hints for multimedia integration, Telescript model for Internet agents and MOVIE/HPFI model for Java based HPCC support.

4 Java Today ('95/'96)

4.1 Current Products After a turbulent period of alpha->beta transition in late '95, Sun delivered on its promise to stabilize Java foundation classes: differences between beta JDK and JDK 1.0 were marginal, and between JDK 1.0 and JDK 1.0.2 just of cosmetic character. PreBeta HotJava was published in spring'96 and hence we have now a stable and complete release of the base Java environment.

On top of this layer, the first few extension APIs appeared in the last months such JDBC (Java Database Connectivity), RMI (Remote Method Invocation), Object Serialization or Java IDL (Interface Definition Language for CORBA).

4.2 Recent Announcements It wasn't obvious until the JavaOne conference which particular course of action will be taken by Sun/JavaSoft in evolving Java. On JavaOne, it became clear that the goal is nothing less but the total solution for the Internet programming. In a series of press releases, JavaSoft announced a broad spectrum of alliances with major industry vendors and the corresponding APIs to come over the next 6 months, including: Beans, Commerce, Enterprise, Media, Security, Servlet and Standard Extension APIs. Sun also announced and demonstrated a prototype of JavaOS, i.e. JavaVM kernel put directly on computer hardware.

For a comprehensive glossary of Web products and technologies as of summer '96, including all Java packages and announcements, see [Glossary96].

5 Implications for the HPCC Community

5.1 Java Performance Problems The major problem to be addressed before Java becomes a viable platform for HPCC is the performance of Java code. JavaVM is an interpreter of bytecodes generated by Java "compiler" and hence effectively slower for numerical computing by a factor 5–50 when compared with the optimized C code. On the other hand, a typical Web user of Java applets will be often satisfied with the applet performance in the GUI control sector — that's because the AWT classes are built on top of X/Motif based windowing and the corresponding X libraries are hardwired into the JavaVM binary code.

In general, designing any interpreted VM based architecture involves a continuous performance / functionality tradeoff. In particular, Java foundation classes offer high performance support for windowing, networking and multithreading in terms of native C code inside JavaVM. This base set was sufficient to build market for Java as an engine for simple interactive applets, but extending the language towards a more general purpose high performance computing requires new software engineering measures.

5.2 Performance Boosters Currently available or emerging Java performance boosters include:

1. **Native Classes** – a mechanism for extending JavaVM at the user level by C-coded classes with the usual Java API
2. **JIT** (Just-In–Time Compiler) – A hybrid interpretation / compilation technique in which a stream of bytecodes is saved during the interpretation process, peephole-optimized and cached to offer better performance on repeated invocations of a given method.
3. **JavaOS** – JavaVM viewed as alternative to UNIX or Windows, put directly on hardware and, in the next step, performance optimized for individual platforms.
4. **Advanced Compiler Techniques** – this includes emergent approaches that employ true incremental compilation at the client side such as SuperCede based on Flash Compiler (just announced by Asymmetrix)

5.3 Analogies with OSF ANDF Another worth mentioning previous/related approach towards mobile compilation techniques is OSF ANDF (Architecture Neutral Distribution Format). Based on the TDF protocol licensed from UK agency RDA, ANDF offered an universal intermediate form for multi-language multi-platform compilers. ANDF producer (analogous to server side Java "compiler") for a given language takes the source code and translates it into a mobile ANDF format. ANDF producer (analogous to a client side Java bytecode compiler), reads ANDF and generates true machine code for a given platform. High performance is achieved by parametric constructs (ANDF tokens) with late binding that allow for the installer side platform specific optimizations.

The ANDF model didn't gain sufficient acceptance in the compiler community but its ideas and techniques are likely to re-appear now in the context of JavaVM, given the OSF's strong involvement in Java technology.

5.4 Native libraries for HPCC As of today, the native methods support is the most natural and accessible performance booster. In the current Java literature, one can often find warnings against excessive development of user level native classes as violating Java portability across all Internet platforms. For the HPCC Java applications, we view the user level native class based approach as legitimate when properly and economically designed since the corresponding high performance JavaVM extensions will typically reside at the compute-server side, whereas the results of such performance optimized computation will be still broadly accessible e.g. for visualization purposes, via fully portable applets.

A natural approach for a given computationally intensive domain is to identify first a minimal set of performance critical primitives, implement this set in the native mode, and then build a higher level application toolkit in Java in terms of such hardcoded operators. In fact, several new APIs announced on JavaOne such as video (with Intel) or animation (with Macromedia) will involve performance challenges and hence must be based on native JavaVM extensions (most likely wrappers to the existing software systems such as ProShare or Director).

In a similar way HPCC community could repackage their existing software libraries and contribute to the rapidly growing collection of Java APIs in the following areas:

1. matrix algebra (Fortran90/MatLab/APL style)
2. imaging primitives (raster ops)
3. PDE primitives (mesh generators, solvers)
4. parallel compiler primitives (lexers, parsers, data descriptors, interprocedural analysis)
5. performance visualization and monitoring
6. load balancing, resource allocation, cluster management

The detailed coordination mechanism between the Web community and JavaSoft API extensions is yet to be worked out. We expect that the Standard Extension API, announced at JavaOne, will provide a framework that will minimize overlap and optimize interoperability between Java extension efforts underway in various labs.

6 Java – (Near Term) Future

Having reviewed the past and the presence, we now discuss the near term future of Java, based on the ongoing activities in the field and focused again on implications for the HPCC community.

6.1 Towards 'All Java' Web Successfully started as a minimal VM model, Java is making now a brave attempt at a total solution for the Internet programming. Unlike previous efforts discussed in Chapter 3, Java can likely succeed as already backed up by major vendors and the Web community at large.

The following ongoing developments that point towards the coming "all Java" systems are worth mentioning here:

1. *Netscape4* – the next release of the Netscape Navigator, already in the implementation stage, will be a full rewrite in Java, and hence a similar concept and competitive product to HotJava.
2. *Jigsaw* – an 'all Java' HTTP server has been recently published by the WWW Consortium as an experimentation platform for mobile code and object-oriented extensions of the Web.
3. *Jeeves* – a similar Java based HTTP server development API is under development by JavaSoft, with the planned beta release in July.
4. *JavaOS* – a concept of installing JavaVM directly on computer hardware and hence a product competitive to UNIX or Windows.

As seen from the above list, we will be witnessing soon an 'all Java' based Web environment (both clients and servers), ready to switch to the JavaOS platform and truly challenge the current Microsoft dominance in the PC operating system market.

6.2 How to not get 'Hooked on Java' Increasing penetration of Java raises natural concerns regarding the vendor hookup – depending on the particular use and lawyers interpretation, Java can be viewed either as a free open system or as a property of Sun Microsystems.

For those who want to do Java development but not get 'hooked on Java', Netscape offers now an interesting protocol called JRL (Java Runtime Interface). JRI formalizes the interface to JavaVM (runtime) in a modular and opaque (i.e. JavaVM implementation independent) fashion. The goal is to decouple consumers of Java services (native class libraries or Java-capable browsers) from the producer of these services (Java runtime and its implementation details).

JRI modularity is accomplished in terms of a collection of abstract interface objects such as *runtime instance* (all global data structures of JavaVM, object heap, name-handle mapping etc.), *execution environment* (thread/C stack, Java stack, Java registers), or *field and method thanks* (opaque data structures that encapsulate implementation details of object fields or methods and are passed as arguments for JRI calls).

6.3 Distributed Java Environments One extremely attractive feature of Java based Web infrastructure is its dynamic character. In a similar way as Java-capable browsers download applets on demand, Java based HTTP servers will soon offer similar capability at the backend side. Such server extensions called *servlets*, dynamically uploaded upon the user request/URL, will substantially augment and soon likely override the current CGI-based Web computational model.

One practical near term implication of the servlet technology for the HPCC community is that Java based distributed computing is imminent. Indeed, with the current exploding variety of Java APIs, it will become soon impractical to maintain such class packages by a single Java-capable Web server such as Jigsaw or Jeeves. A mesh of interactive Java daemons, customized (i.e. uploaded with suitable servlet collections) to offer specialized services appears to be currently the most promising backend architecture for the evolving Web.

Base support for communication between individual JavaVM instances is already present in the foundation classes (such as Client, Server, Socket or URL). Specialized user level message passing protocols can be constructed on top of this layer, for example Caltech Infospheres model for simple agent-style messaging. JavaSoft offers now also a set of communication tools such as JavaRMI (Remote Method Invocation) and Object Serialization, as well as network interfaces to the third party software such as JDBC (Java Database Connectivity — a portable interface to multivendor RDBMS) or Java IDL (interface to the CORBA domain).

These Java specific communication mechanisms build on top of and extend the base transfer protocol for the Web content, formulated in the MIME framework. The relative interplay between both schemes and an evolution path from the MIME to the Java object messaging will be discussed in the following Chapters.

7 An Evolving Virtual Machine

In the previous Chapters (2–4), we reviewed Java evolution and pointed out the potential major role of this language for the Web computing. However, Java still has a set of obstacles to bypass such as:

1. Java multiplatform support is still limited: JavaSoft supports only Solaris and Windows, other vendors (SGI, IBM) follow but at a slower pace.
2. JavaVM installation base is still limited due to increased webmaster skills, required for maintaining the Java environment as compared with a simple NCSA HTTP daemon.
3. Java servers such as Jigsaw or Jeeves are yet to be tested and they will face a serious competition when confronted with the current generation of Netscape server technology such as FastTrack or Enterprise.
4. Netscape4 is yet to be developed, and to come with reduced number of bugs if Netscape Corporation is to survive the competition with the recent Internet products from Microsoft such as ActiveX or Internet Explorer.

While waiting for Java to grow and stabilize, we are currently prototyping at NPAC a distributed *WebVM* framework, based on the currently available Web technologies, and fully open for Java technology insertion. We are also prototyping a Java applet based visual programming environment, *WebFlow* on top of the WebVM computational layer. In the following, we describe our WebVM/WebFlow concepts [WebTools94] [WebWork95] [WebHPCC96] in more detail and we illustrate the evolution path from the current to 'all Java' Web infrastructure.

7.1 WebVM – a minimal model Our minimal WebVM is given by a collection of CGI extended Web servers, typically with an independent server running on each node of a workstation cluster. The CGI extension includes a library of system CGI modules which are provided as part of the WebVM distribution, and a set of custom, user/application provided CGI modules. The overall computational paradigm is given by dataflow with user CGI processes acting as computational nodes/modules, and with the system CGI processes providing the required system control and management functions.

Each module receives a set of input MIME objects, performs an arbitrary local computation, and returns a set of output MIME objects. Both input and output objects are URL-addressable, i.e. they reside as MIME documents in a server document tree, and can be transferred between servers during the dataflow actions.

Each module reads input files from and write output files to the local disk. Server-to-server object transfer is implemented via the HTTP protocol and carried out by the system CGI module called *Flow Executive*. Fig, 1 illustrates the steps involved in transferring the output object $O_1^{(1)}$ created by module M_1 from the server S_1 document tree to the server S_2 document tree, where it is stored as the input object $I_1^{(2)}$, ready to be used by the module M_2. The transfer starts after the module M_1 finished its computation and wrote $O_1^{(1)}$ to the local disk. After completing the write action, M_1 opens connection to S_2 and sends a POST message to the Flow Executive F_2, passing the current URL of stored ${O_1}^{(1)}$. F_2 sends the GET message to S_1, retrieves ${O_1}^{(1)}$, and stores it as ${I_1}^{(2)}$. Next, F_2 starts module M_2 in the background and exits. At this point, M_1 can also exit and all connections between servers S_1 and S_2 are closed. Module M_2 reads its input files ${I_1}^{(2)}$, ${I_1}^{(2)}$, performs local computation, writes its output files ${O_1}^{(2)}$, ${O_2}^{(2)}$, and notifies the Flow Executive processes in the destination servers.

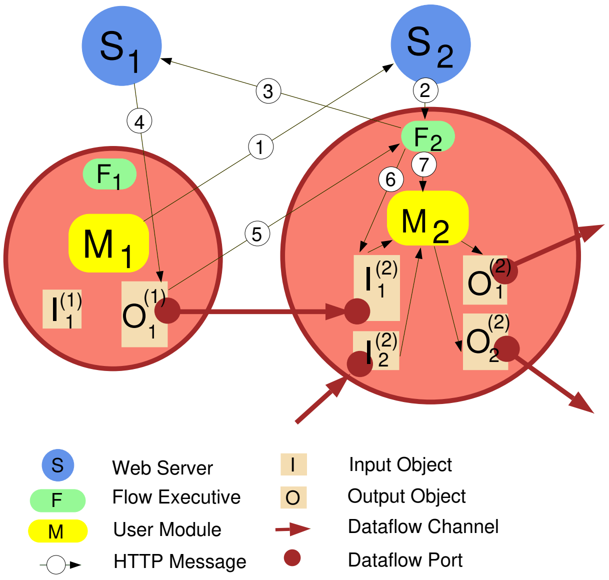

Fig. 1: Minimal model for HTTP+CGI based Server-to-Server Communication discussed in Section 2.2

Fig. 1 illustrates only two neighboring nodes M_1, M_2 and one arc $({O_1}^{(1)} \rightarrow {I_1}^{(2)})$ of a larger computational graph called *compute-web*. A given web server can host and handle modules from several compute-webs, belonging to different users. Local connectivity information is stored in configuration files in each server. Initial or idle state of a WebVM involves a set of (always) running Web servers which act as network daemons, and no other WebVM-specific active CGI processes. Dataflow is typically initiated by a browser which activates the initial module, e.g. M_1 in Fig. 1, via the usual URL addressing mode and the CGI convention. The subsequent M processes are activated whenever the state of any of their input objects changes as a result of some previous dataflow action, and the F processes are activated upon each object transfer request, issued by the M processes.

7.2 Management Modules The minimal model in Fig. 1 contains only one system module (Flow Executive) per node. A more realistic version of WebVM includes additional management modules. For example, *Link Manager* system module is responsible for creating or destroying compute-web nodes and the corresponding updates of the compute-web configuration files. These configuration files require some management support as well to assure atomicity and concurrency control in the multi-user environment. Such responsibility is assigned to *Configuration Manager* module which offers interface to a relational database that maintains the compute-web connectivity information. *Resource Manager* module assures fault tolerance, static resource allocation, and dynamic load balancing via module mobility. Finally, *Script Executive* discussed in the following offers the scripted programmatic extensions of the static compute-web topology.

7.3 WebVM – Java multiserver model The HTTP+CGI platform for WebVM described above has the advantage of large installation base of HTTP servers and the disadvantage of high latency due to HTTP network connections, document tree I/O overhead, and heavy-weighted multitasking of CGI processes.

A Java server (e.g. Jigsaw or Jeeves) version of the same architecture is much more attractive in a longer run since it offers higher performance compute-server operation and more elegant programming environment. In a Java server based WebVM, modules are represented as Java objects, dynamically uploaded by Flow Executives (which are also Java objects) and possibly cached, ready for repeated use. Web server document tree is replaces by JavaVM memory, native OS multitasking by JavaVM multithreading, and disk I/O operations by inter-object communication within the JavaVM addressing space. The canonical MIME type set is gradually extended by the required domain specific Java objects and the corresponding dataflow links are implemented using the Java RMI and Object Serialization tools.

7.4 Portable module API Our main design objective is to provide a framework which would a) allow each W3C compliant Web server to participate in WebVM computations, and b) provide hooks for more advanced servers to exploit their advantages. The Virtual Machine concept provides a natural methodology to satisfy both constraints. The essential point is the distinction between module API, i.e. "what the user sees" and the WebVM implementation layer. We already described above the specific HTTP+CGI based implementation of the minimal model and pointed out that Java server implementation would look quite different. However, a generic WebVM user does not usually see or need to care about many of these details. Users develop application specific modules using their favored languages (C/C++, Fortran, COBOL, ADA, Java etc.) and they follow the dataflow metaphor. The minimal module API is built around the abstraction of a named dataflow port (heavy dot in Fig. 1) and two function calls: *inp_handle = port_read(port_name)* and *port_write(port_name,out_handle)*. For example, module M_1 would execute *port_read* twice to receive handles for input objects $I_1^{(2)}$, $I_1^{(2)}$, perform the computation, then execute *port_write* twice to pass handles for the output objects $O_1^{(2)}$, $O_2^{(2)}$ to the system layer, and exit. This minimal API is identical for various WebVM implementation modes, be it pure CGI, Java or Netscape/LiveWire. Various sites publishing their WebVM compliant modules can select their favored implementation strategies and in fact a given Intranet site can contain a mixture of implementation techniques which are most adequate for individual application needs.

7.5 Scripted Extensions Conventional dataflow model is most useful for possibly heterogeneous but topologically static distributed applications. However, several relevant scientific computing and information processing problems require dynamically varying topologies. For example, Adaptive Mesh Refinement (AMR) algorithm for hyperbolic PDEs generates dynamically new transient tasks for refined grids, and a WebAMR programmer would like to distribute such processes efficiently over the WebVM nodes.

Similar expansion needs arise in Intelligent Agents based information processing problems as discussed previously in the Telescript model. Agents, sent to the 'cloud' in search for specific information, find it often useful to spawn other agents/slaves, send them with suitable subtasks to specialized subdomains, and then collect, compare and integrate the individual information scores.

We initially focus on simpler, more structured and deterministic dynamic compute-webs such as required by AMR. One possible way to handle such problems is to extend the module API by domain specific calls such as *grid_refine* etc. We chose instead a more elegant and problem-independent approach based on scripted extensions of the base WebVM model. In terms of the minimal model and its management modules, we introduce another system process called *Script Executive*, and one more routine in the module API – *script_execute(string)* — which passes scripts in the application-specific *little language* to the Script Executive for immediate execution/interpretation. Script Executive will offer a default set of commands such as creating, destroying or modifying the compute-web topologies, as well as mechanisms for installing new application specific commands by users or their domain system administrators.

7.6 WebHPL – a distributed VM environment Scripted extension of WebVM discussed above can be viewed as an embryonic support for distributed interpretative virtual machine architecture. Our HPFI prototype, described in Section 3.4 was based on a more complete support for parallel VM, including interpreted data parallelism. We refer to the Web version of such architecture as *WebHPL* (Language) — the detailed design and the language selection (e.g. should one switch to Java or stay with HPF or HPC++) is currently being discussed by the HPCC community and will be addressed in the next stage of the evolving WebVM design.

7.7 WebFlow – visual programming layer Several high level programming paradigms can be implemented on top of the WebVM layer discussed in Chapter 2. Based on our previous experience with AVS and Khoros, we view visual authoring of application-specific compute-webs as particularly attractive and useful system integration technique. Java capable browser such as HotJava offers a natural implementation vehicle for such high level tools. We illustrate it in Fig. 2 where the WebVM modules are associated with Java threads at the client side and visually editable via Java GUI/AWT classes. Visual feedback, corresponding to currently instantiated dataflow channels can be implemented e.g. via notification messages sent by individual Flow Executives to the JavaVM at the client side.

We expect such *WebFlow* i.e. visual Web programming paradigm to become increasingly popular with the growing complexity of the multi-language Web. First commercial tools in this domain are already available (e.g. JFactory, Symantec Cafe or Visual Java) and simple Java applet based visual editor prototypes are now also within the reach of the Web research community. We expect a family of such authoring tools to appear soon, specialized for various editing tasks and operating at various granularity levels, ranging from the fine grain Java class or VRML behavior editors, to coarse grain problem solving and system integration toolkits.

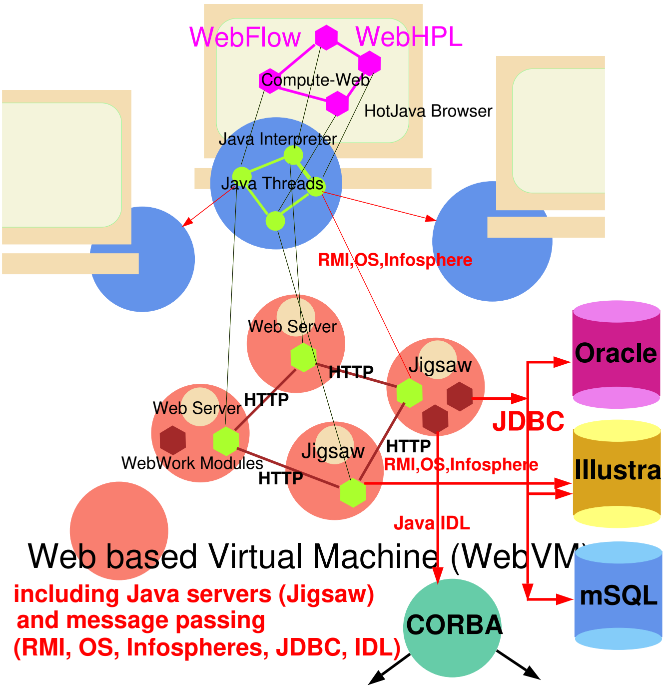

Fig. 2: Overall view of the WebVM/WebFlow architecture which illustrates an evolution path from the current Web technology 'soup' towards 'all Java' systems.

The computational backend in Fig. 2 is given by a mesh of Web servers, forming WebVM as discussed in the previous Sections. An arbitrary heterogenous collection of servers is allowed, including based NCSA HTTD+CGI as well as Java servers such as Jigsaw or Jeeves. We already discussed performance advantages of a Java server based WebVM in Section 7.3. In Fig. 2, we also illustrate that new APIs such as JDBC or IDL can be conveniently used to provide uniform and consistent Java based connectivity to heterogenous database management systems as well as to the CORBA domain.

7.8 Bridge based collaboratory Having reviewed the main components of the WebVM/WebFlow architecture, we now move towards applications and services, and we illustrate our integration concepts in terms of the ongoing application prototyping activities at NPAC.

Most of the real world Web systems will involve multi-user collaboratory modules. Even for scientific computing, complex toolkits such as WebAMR will be most conveniently supported by interactive consultation between developers and users. Collaboratory multi-user components will be further enhanced in enterprise, commerce and community systems. This is illustrated in our recent telemedicine prototype for the nursing triage. Here we start from the collaboratory component involving nurses, nurse practitioners and pediatricians, and we gradually add HPCC components (medical imaging, agent–based diagnosis etc.)

We view the *Bridge* topology [Bridge96], underlying such telemedicine systems, as a promising generic framework, applicable also for other problem domains such as command & control or distance education. A generic bridge includes "points of need", "points of expertise" and intelligent middleware that manages information resources and provides connectivity between customers and optimal services.

7.9 Case Study: Web Based Telemedicine We illustrate here selected components of a telemedicine "bridge" service under development at NPAC. Current projects include 3D Visible Human and CareWeb collaboratory for the school nursing.

The goal of the 3D Visible Human project [VisHuman96] is to perform 3D reconstruction of the human body, represented as a set of 2D image slices in the Visible Human dataset. Some proprietary solutions already exist. Our goal is to develop public 3D representation, accessible on the Web via Java and VRML. Currently, we use AVS to develop and test imaging algorithms for 2D image segmentation. In parallel, we develop Java based WebFlow editor, Java based imaging toolkit (with raster ops as native class library), and a VRML based body navigator. We also develop collaboratory tools to be used both in the reconstruction process (interactive consulting with anatomists) and in the medical applications later on (virtual surgery).

CareWeb project [CareWeb96] is a community collaboration including NPAC, SU College of Nursing, Syracuse City School District and SUNY Health Science Center (Univ. Hospital). Initial goal is to provide electronic student health record database, healthcare education and Web based interactive consultation between nurses, nurse practitioners and pediatricians. Trial demo implementation has been completed in May'96. Trial deployment in select New York and North Carolina schools is planned for the summer/fall '96.

CareWeb core module is given by Oracle database at NPAC with WOW/OWA Internet gateway, remotely accessed by CareWeb customers. The system integrates and offers customized access to ~30 databases, managing information about users, health education resources, and patient health records. Typical CareWeb databases include: Customers, Connections, Transactions, Schools, Teachers, Nurses, Nurse Practitioners, Doctors, and Record Components such as Immunizations, Screening Tests, Health Histories, Progress Notes, Visit Logs, Assessment Forms etc.

CareWeb Information Pages offer customized educational support for healthcare personnel as well as students and parents, and a decision tree support to be used in the next project stage for the agent-based automated diagnosis generation and verification.

Interactive consulting is based on shared record pages, optionally synchronized via phone chat or/and WebCast support, and VIC/VAT video support for 'talking heads' and/or video feeds from (Welch Allyn) multi-purpose fiberscopes for ear, nose and throat inspection.

The system offers multi-level security, including Internet guests (with anonymous limited access), CareWeb friends (with registered restricted access) and CareWeb customers (nurses, nurse practitioners, doctors, parents) with secure password based access, individual home pages and customized information/operational spaces.

Next steps to be addressed during summer '96 for CareWeb and related Web collaboratory service development include: a) migration from Oracle specific PL/SQL environment towards WebVM framework with Java/JDBC and FastTrack/LiveWire nodes to support portable interfaces to heterogeneous distributed DBMS, and b) WebFlow support for rapid visual prototyping of specialized systems for individual customers (e.g. schools, hospitals, HMOs, etc.). Early prototype of such WebVM/WebFlow based collaboratory infrastructure will be demonstrated during the HPDC-5 conference.

8 References

[NeWS89] Gosling, J., Rosenthal, D.S.H., Arden, M., *The NeWS Book*, Springer Verlag, 1989.

[MOVIE92] Furmanski, W., “MOVIE – Multitasking Object-oriented Visual Interactive Environment”, NPAC Technical Report, June 1992, in Fox, G.C., Messina, P. and Williams, R., *Parallel Computing Works*, Morgan and Kaufman, 1994, http://www.npac.syr.edu/projects/hpsin/movie.html

[HPFI93] Fox, G.C., Furmanski, W. and Niemiec, J., "High Performance Fortran Interpreter", demonstrated at Supercomputing'93, Portland, OR, http://www.npac.syr.edu/projects/hpsin/hpfi.html

[WebTools94] Furmanski, W. “WebTools”, http://king.syr.edu:2006/WebTools.html

[WebWork95] Fox, G.C., Furmanski, W., Chen, M., Rebbi, C., and Cowie, J., “WebWork: Integrated Programming Environment Tools for National and Grand Challenges”, NPAC Technical Report, SCCS-715., http://www.npac.syr.edu/techreports/html/0700/abs-0715.html

[Bridge96] David Warner, D. and Balch, D., “Medicine Meets VR: BRIDGE”, http://www.telemed.med.ecu.edu/bridge/bridge_1.htm

[WebHPCC96] Fox, G.C. and Furmanski, W., “Web based HPCC at NPAC”, http://www.npac.syr.edu/projects/webpsace/webbasedhpcc.html

[CareWeb96] SU College of Nursing, NPAC, Syracuse School District, SUNY HSC, “CareWeb – a Web based community oriented healthcare communications system”, http://www.npac.syr.edu/projects/careweb, http://king.syr.edu:2006/WebMed/Demos/CareWeb

[VisHuman96] NPAC 3D Visible Human Project, http://www.npac.syr.edu/projects/3Dvisiblehuman

[Glossary96] Furmanski, W., “Glossary of Web Products and Technologies, Summer '96”, http://king.syr.edu:2006/IWT/96/Glossary.html