BEEHIVE: AN ADAPTIVE, DISTRIBUTED, EMBEDDED SIGNAL PROCESSING ENVIRONMENT

Shahram Famorzadeh *Vijay Madisetti* *Thomas Egolf* *Tuongvu Nguyen*
Center for Signal and Image Processing (CSIP), Georgia Tech-ECE, Atlanta, GA USA 30332-0250

ABSTRACT

We propose an *open* signal processing system design and implementation environment, BEEHIVE, that allows application developers to rapidly compose and debug functional specifications in a networked, distributed computing environment, and then later migrate the application (transparently) onto an embedded, distributed, computing hardware/software platform, with the capability to reconfigure (adaptively) the resources assigned to the application to meet the dynamic real-time requirements of the implementation. Recent developments in the area of virtual machines; broker-based, distributed, transportable computing; object-oriented programming methodologies, Java and its real-time extensions; reconfigurable and programmable hardware; approximate algorithms; adaptive-load and resource-management algorithms, are harnessed in this operating environment [1].

1. INTRODUCTION

Performance requirements of signal processing applications, (e.g., ATR in real-time (RT) and non-RT) of near-term interest to the HPC community range between 2-50 Gigaflops (GFLOPS) with data bandwidth requirements between 80-1000 Mbytes/second. Due to their impressive computing and communication requirements, even evaluating and studying the performance of various algorithms on a non-realtime testbed is difficult. Implementing a chosen algorithm for field use is considerably more difficult, given the amount of customized control, diagnostic, and test software required to port the algorithms onto a heterogeneous, reconfigurable, form-constrained commercial off-the-shelf (COTS) hardware and software platform. Long prototyping times and excessive costs plague all aspects of the problem: algorithm design, development, and porting algorithm onto a real-time platform for field use.

The long-term goals for the BEEHIVE project are summarized in figure 1. Proceeding in a clockwise manner, we start with the current practice in algorithm development platforms (workstation with machine specific, compiled DSP library-based application support). Examples of these include Matlab, Mathematica, and Ptolemy. The need for parallel processing, as computing requirements increase, is depicted in the next figure, where the development platform includes two or more networked workstations with task partitioning and parallel processing features. Representative examples include the Sequent (in the commercial arena), Parallel Virtual Machine (PVM) (from ORNL) and its Java-based incarnations, and Network of Workstations (NOW) (from UC Berkeley). Industry trends in standards have supported this progression. BEEHIVE represents a big improvement in this family of architectures, providing both the required bandwidth and the real-time performance. BEEHIVE provides a distributed environment that adaptively assigns resources (computing and communication) to the application within a portable, library-based, DSP environment. Real-time embedded COTS boards and reconfigurable hardware platforms (e.g., FPGAs) are seamlessly included within the universal application design and implementation environment, through the specification of a BEEHIVE-compliant protocol for embedded boards. In the example of figure 1, transparent to the user (who still thinks that the application is running on the Sun Sparc), the FFT was computed on a BEEHIVE-compliant FPGA-based board, while the DCT was computed on a COTS DSP-based board, through use of Sun's Java-based technology called *object serialization*, and a broker-based, distributed computing paradigm [1-4]. The software developed by the user as part of the algorithm evaluation and design phase can be ported, unchanged, onto the fielded system, also transparently to the user. BEEHIVE, in this way, represents a high risk, high-payoff technology of critical importance to the goals of the embedded, real-time, high performance computing community.

2. DESIGN VS. IMPLEMENTATION

BEEHIVE proposes a *cross-development* philosophy as depicted in figure 2, where the application is developed initially on one system and ported to another run-time system, with minimal change. The library-based Java application design and development testbed is shown in figure 2(a). The application layer sits on the top of the Java virtual machine and can be ported across the network of workstations. In the implementation environment, there is an additional real-time virtual machine layer that consists of extensions to Java to support deterministic real-time scheduling and resource management (rate monotonic analysis, real-time

[1] This research was supported by the Advanced Sensors Consortium sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAL01-96-2-0001, 1996-2001 and in part by DARPA's RASSP program (1994-97). Views presented in this paper are authors' alone.

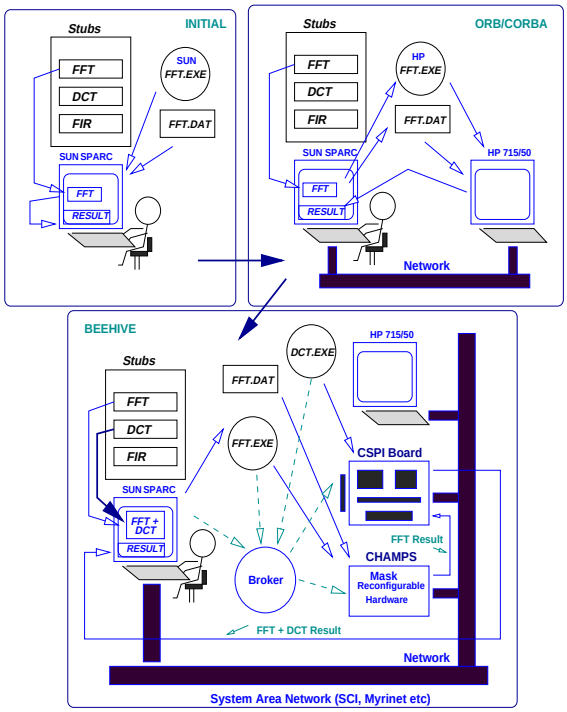

Figure 1. **Evolution of the BEEHIVE Project.**

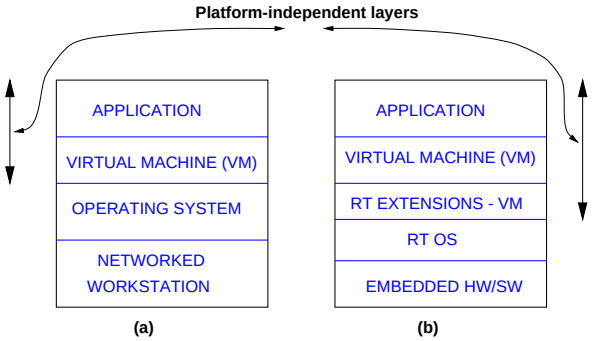

Figure 2. **BEEHIVE's cross-development platform. Note: BEEHIVE v0.1 currently supports the model of figure 2(a). Later versions plan to phase in support for figure 2(b).**

guarantees) that sits on the top of the real-time OS (RTOS) layer as shown in figure 2(b). A candidate commercial real-time virtual machine layer is PERC from NewMonics, Inc [2]. We will discuss the algorithm development environment next, followed by a description of the run-time implementation environment. Currently, BEEHIVE v0.1 supports the environment of figure 2(a).

In the next version of BEEHIVE, we plan to include real-time DSP boards into the environment through the introduction of a BEEHIVE-compliance "sign-off" procedure, where a processor on each DSP board ensures that autocoding (since bytecode may not run on the DSP board) is used to execute a function locally on each board, transparently to the user.

3. BEEHIVE V0.1

We will now discuss the current version of BEEHIVE as implemented at Georgia Tech. Java is a programming language of choice because it is architecture-neutral and portable; simpler and more efficient that C++ (with addition of multithread support, and support for real-time); provides support for run-time linking of objects (avoiding extensive recompilation) facilitating plug-and-play; and enjoys extensive support by the networking industry. Real-time extensions to Java are also currently being developed.

BEEHIVE (implemented primarily in Java) comprises of users, resources, services, and brokers as listed below.

Resource Objects: including (a) computational resources (workstations, HPC machines, PCs, special purpose hardware (FPGAs), PDAs, etc.,), (b) data source resources (keyboards, PDAs, cameras, sensors, reuse libraries), (c) data sink resources (displays, data stores, reuse libraries).

Service objects: (a) Java application subroutine libraries, (b) data handling/fusion routines, (c) FPGA/Embedded equivalents of library, (d) control monitors, (3) real-time OS, etc.

Broker objects: Objects which (a) handle requests, (b) maintain list of services, (c) schedule services on resources, (d) ensure stability, latency constraints, efficiency of distributed platform, and correctness, etc.

User COE Interfaces: (a) That request services from brokers through Java stubs or dynamic invocations via a common operating environment (COE) interface, (b) that download applications that run locally on the Java compliant-embedded system, or on remote resources through control procedures contained within the broker to enable rapid, portable, distributed computing.

The operation of BEEHIVE v0.1 can be described as follows. When user U_1 logs in via the COE, BEEHIVE starts a new thread, a *PersonalityBroker*, that obtains user identification, priority, and current application needs. The *PersonalityBroker* contacts a *DomainInformationServer* to obtain information on what domains are currently supported within the environment (figure 3). As shown in figure 4, domains could be Rice University, Georgia Tech, Sanders, or Berkeley, who each have their own *DomainResourceBrokers*. The *DomainResourceBroker* lists the services and resources within its domain (including FFTs, DCTs,

-

FPGA boards, Mercury boards, and others). The *PersonalityBroker* then selects those services (e.g., primitives, software libraries) that user U_1 could find valuable, and composes a menu (only function stubs, without executables or bytecodes) that it presents to the user. U_1 then selects various services to *compose* an application specificed together with form-function-fit constraints. As shown in figure 4, this application could consist of a 2D-FIR, followed by an FFT operation.

After the application is composed and launched by U_1, BEEHIVE assigns an *ApplicationBroker* for its implementation. The *ApplicationBroker* then maps and assigns domains where these services (FFT, FIR) can be executed (based on the initial replies from the *DomainResourceBrokers* when U_1 logged in) and uses object serialization to migrate the tasks to the assigned *DomainResourceBrokers*. For instance, the FFT was sent to the Georgia Tech *DomainResourceBroker*, while the FIR was sent to the Sanders *DomainResourceBroker*. Each *DomainResourceBroker* then invokes a local *ResourceBroker* that handles the assignment of FFT to a specific local resource. Each resource is provided with a *TaskManager* that supports object serialization and is responsible for handling the interface with the *ResourceBroker*.

In a real-time embedded implementation, for instance, if the FFT sent to the *DomainResourceBroker* were to be mapped to a FPGA board, then the *TaskManager* for the FPGA board will accept the FFT.DATA, configure the FPGA board to compute the FFT and then send the result back to the *ApplicationBroker*. In BEEHIVE v0.1 all resources are networked so the task is simply migrated onto to the target processor (say, a Sparc at Georgia Tech). All these activities are transparent to the user U_1, who for all purposes thinks that the entire application and the resource library are resident on his local machine.

Any stable dynamic load balancing algorithm can be implemented within BEEHIVE. For instance, the *ApplicationBroker* and the *ResourceBrokers* could collect load information dynamically and migrate tasks to other resources if quality of service guarantees were not being met by the current resource.

Real-time extensions to Java, namely PERC [2], will be phased into the next version of BEEHIVE to target embedded applications where precise load and memory usage guarantees are required to meet real-time performance constraints.

4. CURRENT AND FUTURE WORK

Current work is being done on the following four fronts:

1. *Library of Primitive Functions*: A library of DSP functions (FFT, DCTs..) in Java is being created for use within a variety of domains. A number of approximate algorithms, in addition to programmable masks for FPGA boards will be included within the primitive COE.

2. *BEEHIVE Infrastructure*: We are working extensively on the development of the COE, and on the implementation of the various application and resource brokers. Efficiency, scalability, and quality of service issues are the primary metrics in the various tradeoffs that we are considering.

3. *Real-time Extensions*: A number of enhancements are required before BEEHIVE can be utilized as a cross-development platform as desired. The role of *TaskManagers* for resources in ensuring compliance with object serialization protocols within Java remain to be finalized. These additions will ensure that reconfigurable hardware, and other embedded processing resources (e.g., cameras, COTS boards, printers, display devices, and sensors) can be included within the prototyping environment. A specific BEEHIVE-compliance protocol is being formulated for ensuring that DSP boards can be seamlessly incorporated into the environment.

4. *Adaptive Resource Management:* In real-time applications, task deadlines must be met with guarantees. Java currently does not provide determinstic guarantees on bounds on memory usage, and on task execution times. Recent extensions to Java, such as PERC, provide these features, and we will investigate these extensions with distributed, adaptive, and dynamic load balancing algorithms that are scalable and robust [3].

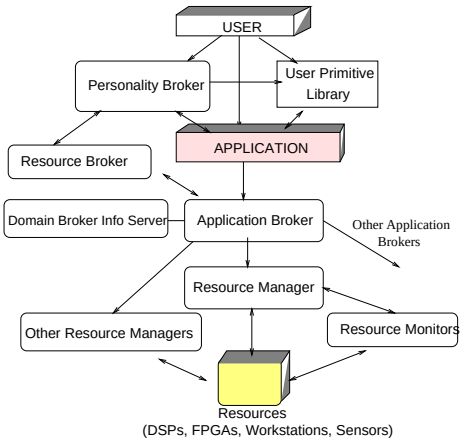

Figure 3. Architecture of Brokered Computation Model

5. SUMMARY

The DARPA and ARL-sponsored BEEHIVE project at Georgia Tech is one of the first efforts towards a distributed, adaptive, embedded, signal processing environment for both algorithm design and evaluation, and for its eventual implementation. A number of promising state-of-art technologies have been included in its implementation and an initial release. BEEHIVE v0.1 is currently available for demonstration purposes. While BEEHIVE v0.1 only supports a networked workstation based environment, research new extensions to real-time embedded platforms with programmable hardware is under way.

6. REFERENCES

1. *The Java Language Overview*, Sun Microsystems, Inc. 1995.

2. K. Nilsen, "Embedded Realtime Development in Java", NewMonics, Inc. *Java Developer's Journal*, 1996.
3. V. Madisetti, *VLSI Digital Signal Processors*, IEEE Press, Piscataway, NJ, 1995.
4. V. Madisetti (PI), *BEEHIVE: An Environment for DASP*, Georgia Tech Research Corporation, A DARPA BAA 96-16 Proposal, Vol. 1-2, June 5, 1996.

Authors may be contacted at vkm@ee.gatech.edu (http://www.ee.gatech.edu/users/215/).

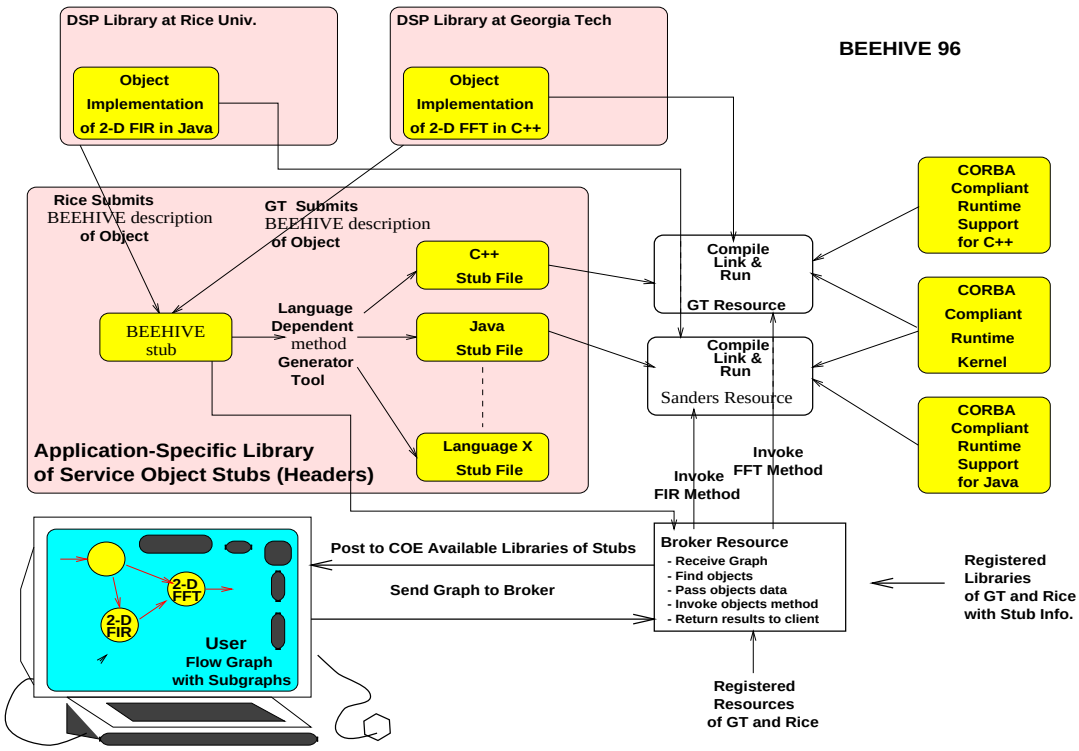

Figure 4. BEEHIVE Environment v0.1. Next version plans to include seamless support for BEEHIVE-compliant DSP cards and FPGA boards.