# Integrating Adaptive Computing with Distributed Services Medicine

To appear, 12'th Annual IEEE Symposium on
Computer-Based Medical Systems, Stamford, CT, June 1999.

Barry E. Mapen, Joel Rosiene
Computational Medicine Group,
Department of Computer Science and Engineering,
University of Connecticut
barry@cse.uconn.edu, jrosiene@etc.atinc.com

**Abstract**

*We present a foundation for using reconfigurable and adaptive systems in distributed services medicine to improve the computational power available to physicians. This power can be realized in medical image processing, enhancement, and classification. Adaptive systems will become commodity items in the near future and will be modified by mobile agents. These transportation agents will bring the data, code, hardware, or any combination to the machine for processing. This will open tools to every physician that were formerly only available to advance-care institutions with powerful workstations or supercomputers.*

## 1: Introduction

Modern medicine is practiced in a distributed, collaborative setting where a variety of caregivers interact at a distance to provide care to any given patient. While a physician is physically located at one location, the computational resources s/he requests may come from another location(s). Typically, the more powerful, and expensive, computational resources reside at the advanced-care institutions and are limited to the physicians at those locations. In some cases, local hospitals may transfer data to the larger institutions for analysis. We propose going beyond this client-server model of computational resource sharing by using distributed services. To advance the entire computational network, we are investigating reconfigurable systems. This paper is not concerned with the underlying network fabric; we simply assume that TCP/IP or some other network protocol connects all of the machines. By machine, we are referring to conventional desktop computers, super computers, workstations, and adaptive computers (which are what we are most concerned with). The only hardware assumption that we impose on these adaptive computers is that they can support mobile agents [7].

A mobile agent can transport data or code. The agent may remain local, or determine that additional resources are necessary and "look" for a more suitable machine. Which machine the agent selects is an issue of dynamic load balancing and is beyond the scope of this paper [2]. Mobile agents allow us to move away from the client server model where data is moved to a fixed machine running a piece of code. We can bring data to code,

code to data, and both to a stronger processor, or cluster of processors. This opens a large computational resource to all physicians - not just those located within the advanced-care institutions.

By adding in reconfigurable computing, we retain the ability to move data and code, but gain the ability to "move" the hardware as well. With a reconfigurable processor, the internal configuration is simply a data file. When it is loaded into the processor, the internal silicon resources are configured [4]. Specialized hardware to complete a given task can be transported by the agent just as code or data would be moved. A mobile agent may leave the local machine to retrieve the hardware necessary to perform a desired operation. This hardware can then be downloaded to the reconfigurable computer and the data may be processed locally. Potentially, the code, data, and specialized hardware will migrate to a computer sitting idle in a lab, run, and return.

## 2: Architecture: overview

It is expected that reconfigurable processors will become more common place within the next five years [9]. These processors are able to change the configuration of their basic elements under run-time control by software. In our case, we are looking at mobile hardware that can be controlled by mobile agents. The framework we need for mobile agents is described below, but since Java$^{TM}$ (Java is a registered trademark of Sun Microsystems) is available now, our examples are described in terms of aglets.

### 2.1: Field programmable gate arrays (architecture)

Field Programmable Gate Arrays (FPGA) are designed in a hierarchical fashion. At the lowest level we have a small block consisting of a few combinational logic components. These components commonly include several MUXes, D-FFs, and individual logic gates. The collection of components in this block is usually called a logic cell, or cell for short. Cells are clustered into groups with wiring lines connecting them [4]. A group of cells can be thought of as a functional block. These blocks are recursively organized in a hierarchical fashion until the aggregate of the chips resources have been grouped. Hardware designers build circuitry in a hierarchical fashion that maps naturally into this model. FPGAs are designed to be configured in-circuit and modified while running. Changes to the hardware configuration may occur synchronously or asynchronously.

### 2.2: Reconfigurable vs. adaptive (synchronous vs. asynchronous)

We define reconfiguration as the synchronous changing of hardware. These changes will be directed by program control. For example, we may have an x-ray that needs to be contrast enhanced, noise reduced, and compressed. Each of these image-processing algorithms may be done on a conventional processor, but they can be done faster by specialized hardware. The hardware demands for each of these requests is sufficiently different that we would not want to use the same specialized hardware for each. Hence we reconfigure the hardware synchronously within the execution of the algorithm.

We view adaptive computing as the evolution of hardware layout to meet specified goals. For example, while a conventional processing system is completely characterized by contents of memory, registers and a fixed architecture, the presence or absence of interconnections

between processing elements further characterizes an adaptive computer. For Telepresence, with a large variety of data sets, the data being streamed at any point in time may be different. If the hardware to decode the data set is transmitted preceding the data set, the hardware can adapt to perform the decoding of the new data set. For example, the decoding of a compressed data stream is particularly time consuming for a general purpose processor, since the operations are often conditioned on a single bit. Many multimedia processors have dedicated hardware to decode a bitstream (for example MPEG2), which is idle if the bitstream is something else. Consider also a set of finite reconfigurable resources being used by multiple processes. It is clearly obvious that resources may shift from lower-priority tasks to higher-priority tasks if the higher-priority tasks impose a requirement for resources not available for use [5].

### 2.3: Specialized hardware

It has been shown that specialized hardware can accelerate computations significantly [9]. In general-purpose processors resources are fixed. If a static resource is not being used then we can consider it to be a waste of silicon. For example, the FPU on modern processors may occupy up to 70% of the net real estate of the chip. If we are running an integer-based algorithm, then we are only using 30% of the available processor. Reconfigurable hardware can reduce this waste by only allocating hardware that will be used. This also opens up the potential to allocate many small processors or one large processor in a single FPGA. Algorithms that can benefit from parallel computation or multiple threads can take advantage of this property by allocating as many subprocessors as the underlying chip can support [1]. Another example is if we are working with image data. It may be encoded as 12-bit data, but conventional processors only have 16- or 32-bit registers. Reconfigurable solutions help reduce the problem of wasted resources and allow for parallelism that would originally require multiple processors and/or machines. Again, we do not loose the ability to use multiple processors and machines, but increase our flexibility and computational power.

### 2.4: Unique solution, a HOS

The current view of reconfigurable hardware is a pool of components that are stored in memory somewhere [8]. When a part is needed, it is downloaded, used, and then removed to make space for the next component. This approach has several drawbacks. First, there are a large number of components that one might want to download. For example, to contrast enhance an image, noise filter it, and then compress it, we need those three pieces of hardware. Perhaps we have a batch of images that we need to process. Ideally, we would take full advantage of the hardware at our disposal, so we want to drop all three components in at once and push the images through each stage in pipeline fashion. To do this, we need to get the configuration file that contains all of these pieces. Generating all of the files that would be needed to cover all possible configurations for a moderate number of components is an intractable problem.

The second drawback to the current reconfigurable systems is the variations between manufacturers. The internals of reconfigurable chips vary not only between manufactures but even vary between product lines within a single manufacture. If we know that we are working in a homogeneous environment then this may not be a problem. However,

homogeneity cannot be guaranteed in the medical environment and technology will most likely be upgraded in stages, further preventing a homogeneous system from developing.

As a solution to both of these problems, we propose a Hardware Operating System (HOS). This system will allow an agent to download a mobile hardware file to the hardware without making assumptions about which chip is under the agent. The HOS is responsible for translating the generic configuration into its specific hardware. While the hardware is running, the HOS can begin to optimize it based on a set of goals (e.g. space, speed, power consumption, etc). Since the agent knows the algorithm it needs to execute using this hardware, it may choose to download multiple copies of a piece of hardware for parallel computation or pipeline stages for synchronized throughput. This decision will be based on the algorithm/operation selected and the agent will be responsible for controlling the hardware.

If the HOS is able to make any optimizations, it may choose to cache this configuration for future use. Similarly, the agent may know that it is making a common request and may ask for a copy of the optimized hardware back from the HOS before returning to its originator with the resulting data. The next time the agent arrives, it can bring the improved configuration with it.

### 2.5: Role of agents

Agents in the system we are proposing are simply couriers of information such as data, code, or hardware. They must be able to move from host to host over the network fabric that connects the hosts. An agent needs to be able to communicate with the reconfigurable processor either directly or indirectly. While no standard exists at this time, development is underway with Xilinx, a major reconfigurable processor manufacturer, to add Java$^{TM}$ support to their chips [3]. This interface will allow a Java$^{TM}$ program, in this case and aglet, to reconfigure the processor. We expect to see this type of interface standardized within the next few years. As far as security issues are concerned, we leave this up to the designer of the agent technology. For example, IBM has a working model with their aglet transport protocol [6].

Since the translation process to download the hardware requires some computation we would like to avoid this if we repeatedly use common hosts for common tasks. If the agents can query a host to discover its specific hardware, then the aglet can request a chip specific copy of the hardware back. This is advantageous because the next time the aglet goes to this host (or one with the same reconfigurable chip), it can bring a fast downloading copy of the hardware with it. Providing this does not violate the security model of the agents, the hardware will not need to be translated from the mobile hardware form into the chip specific form.

## 3: Application domain

Medical image processing can provide physicians with valuable information. For example, a diagnosis may indicate that a patient has a fracture. By looking at the x-ray the physician cannot see anything wrong. If the image can be contrast-enhanced quickly, the physician can examine the image to confirm the initial diagnosis. Agents and mobile hardware can provide the speed necessary to make the medical image processing useful to physicians at any location (not just advanced-care institutions).

In this example of contrast enhancement, the physician's computer creates several agents to retrieve the necessary tools. One goes to fetch the newest contrast-enhancement tools from some service provider while another retrieves a lossless compression tool in case the data needs to be shipped to another machine. A diplomatic agent is created to look for the most appropriate computer (or cluster of computers) to run the enhancement on. When the compression tool returns to physician's machine, it uses the internal reconfigurable processor to compress the image to reduce bandwidth for transportation. The diplomatic agent returns with the best machine(s) for this job. If the diplomatic agent selects the local machine, the optimized compression tool is cached and the enhancement tool is loaded. The computation is performed using the non-compressed data and the results are displayed. If remote machine(s) are selected, the compression and contrast enhancement tools are encapsulated with the data inside an agent that goes to the processing node(s). When the results are available, the agent brings them back to the physician's computer for display and analysis.

While contrast enhancement is a fairly simple process computationally, the same type of scenario can be applied to volumetric MR scan data. This massive image data is generally in the order of several megabytes. A difficult patient may require a dozen of these images to be processed. Three-dimensional reformatting, three-dimensional rendering and image morphology lend themselves to high parallelism and can benefit from mobile hardware solutions. The reconfigurable mobile hardware benefits are greatest in areas where the algorithm does not map efficiently to general-purpose hardware. For example, some algorithms use blocks of image pixels for the foundation of their operations. In contour-following algorithms, the topology can change based on the interpretation of a pixel. When the topology changes, the degree of parallelism changes, as does the image stack and the hardware can be reconfigured to support these changes. More importantly, the HOS will be working on adapting the hardware to a more efficient representation within the specific reconfigurable chip. When the agent's algorithm causes the hardware to be reconfigured, the HOS can cache the optimized versions it has created.

## 4: Conclusions

This is an exciting new area of distributed services medicine with lots of future directions. This paper has shown how physicians at an arbitrary location can take advantage of computational resources that were only available to advanced-care institutions. Computations may be performed locally, remotely, or by a distributed mixture of this. Agents allow for code and/or data to migrate to a machine or cluster of machines. The mobile hardware abstraction facilitates the transfer of hardware to any reconfigurable platform. By allowing agents to carry mobile hardware in addition to code and data, they can make the hosting machine a special purpose system optimized for the desired operation. Lastly, algorithms that can make use of multiple threads or parallelism can use multiple real machines by taking advantage of the extra real estate in the chip.

## References

[1] Ian R Greenshields. A dynamically reconfigurable multimodal architecture for image processing. In P.M. Dew, R.A. Earnshaw, and T.R Heywood, editors, *Parallel Processing for Computer Vision and Display*, pages 153–166. Addison-Wesley, Reading, MA, 1989.

[2] Ian R Greenshields and Zhihong Yang. Architecture of a distributed collaborative medical imaging system. In *Proc. 11'th IEEE Symposium on Computer-Based Medical System*, pages 132–136, Lubbock, TX, June 1998. IEEE.

[3] Xilinx Inc. Java applets empower internet reconfigurable logic. [Online]. Available at `http://www.xilinx.com/prs_rls/java2.htm`.

[4] Xilinx Inc. *XACT Step Series 6000 User Guide*. Xilinx, Inc., Scotland, 1996.

[5] Krishna Kavi, James C Browne, and Anand Tripathi. Computer systems research: The pressure is on. *IEEE Computer*, pages 30–39, January 1999.

[6] Danny B Lange and Yariv Aridor. Agent transfer protocol – atp/0.1. [Online]. Available at `http://www.trl.ibm.co.jp/aglets/atp/atp.htm`.

[7] Jeffrey A. Meunier. A virtual machine for a functional mobile agent architecture supporting distributed medical information. In *12'th IEEE Symposium on Computer-Based Medical System*, Stamford, CT, June 1999. IEEE.

[8] Doug Smith and Dinesh Bhatia. Race: Reconfigurable and adaptive computing environment. *Lecture Notes in Computer Science*, 1142:87–95, 1996.

[9] Jean E Vuillemin. On computing power. [Online], 1994. Draft. Available at `ftp://ftp.digital.com/pub/DEC/PRL/research-reports/PRL-RR-39.pdf`.