InfoFabric: Adaptive Services in Distributed Embedded Systems

Karsten Schwan, Christian Poellabauer, Greg Eisenhauer, Santosh Pande, Calton Pu
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{schwan, chris, eisen, santosh, calton}@cc.gatech.edu

Abstract

The key problems addressed by this work are: (1) how to run high-end applications on embedded systems, (2) how to provide high levels of flexibility in how, where, and when necessary processing and communication actions are performed, and (3) how to continuously meet end-user needs despite run-time variations in service locations, platform capabilities, and user requirements. Our approach addresses these problems by offering the following components. First, the InfoFabric *platform supports flexibility in communication and processing. It is a lightweight publish/subscribe middleware end-users can use to subscribe to information channels of interest to them whenever they desire, and apply exactly the processing to such information they require, such that processing and communication actions are dynamically mapped to the underlying distributed embedded platforms. Second, to attain high performance and to meet embedded systems requirements like low power, new compiler and binary code generation techniques dynamically generate and install code. Run-time code generation customizes code in order to match current user needs to available platform resources. Third, to deal with run-time changes in resource availability, a kernel-level resource management mechanisms, called* Q-fabric*, is associated with middleware. Combined, these mechanisms efficiently carry the performance, usage, and requirements information needed for run-time adaptation of processing and communication actions.*

1 Introduction

Background. Embedded systems are becoming increasingly heterogeneous and dynamic, where end-users assume services to be available despite varying levels of wireless network connectivity, with devices that range from cell phones to laptops, and with levels of interaction that start with the asynchronous receipt of email and end with real-time collaboration via rich data sets. Our work focuses on high-end applications and services in distributed embedded systems, with the intent of complementing ongoing industry research, which is already making rapid progress in the construction of lower end ubiquitous systems using technologies like Java. Java limitations are well-known, where typical Java JDK 'footprints' occupy substantial amounts of memory, Java communications offer high levels of flexibility rather than high performance by using serialized objects as communication units [13], and low power requirements for embedded devices are difficult to meet because of the lack of real-time solutions for Java.

Problems addressed. The key problems addressed by our work are:

- the support of high-end applications on embedded systems, such as high-quality graphics on handhelds and video/visualization data transmission across wireless links,

- the provision of high levels of flexibility in how, where, and when these applications' required processing and communication actions are performed in mixed server/embedded systems platforms, and

- the ability to continuously meet end-user needs despite runtime variations in service locations, platform capabilities, and user requirements.

Research approach. Our approach to these problems has four components:
(1) InfoFabric platform. Flexibility in communication and processing is provided by lightweight publish/subscribe middleware: end-users subscribe to information channels of interest to them whenever they desire, and they apply exactly the processing they need to such information, using dynamically generated handler functions. The processing and communication actions performed by handlers are dynamically mapped to the underlying distributed embedded platforms. The effect is that the InfoFabric operates with levels of flexibility similar to those of Java environments, but with performance akin to that existing in real-time and embedded systems commonly used in industrial control, military applications [10], and robotics.
(2) Dynamic binary code generation. To attain high performance and to meet embedded systems requirements, new compiler and code generation techniques dynamically generate and install new handler codes on the InfoFabric's platforms. Run-time code generation specializes such code in order to match current user needs to available platform resources.
(3) Configurable resource management. Run-time changes in resource availability are dealt with by a lightweight, kernel-level resource management framework, called *Q-fabric*. This framework efficiently carries the performance, usage, and requirements information needed for run-time adaptation of processing and communication actions. The intent is for middleware to have detailed knowledge of the ways in which information should be transported and manipulated before delivering it to end-users. As a result, the InfoFabric employs techniques like automatic redundancy and replication, and service (re-)location and (re-)partitioning to match changing user needs to changing platform characteristics.
(4) Evaluation with high-end applications. The applications used to evaluate our approach concern the timely delivery of critical information, ranging from the display of real-time captured sensor data, to image streaming, to high-end display actions on devices not typically capable of such actions (e.g., PDAs).

Results and status. InfoFabric's middleware-level implementation runs on most processor platforms, including SPARC, i86, MIPS, Itanium, and most recently, ARM and XScale. Dynamic binary code generation techniques exist for most of these platforms, enhanced by run-time methods for dynamic linking. This component of the InfoFabric is also used in commercial applications (e.g., in Delta Air Lines' operational information system [4]). To permit precise run-time quality management, a kernel-level port of InfoFabric extends its notions of publish/subscribe channels and dynamically deployed handlers to Linux operating system kernels. The result is that critical InfoFabric communications and processing actions can be performed in the kernel, satisfying the predictability and real-time requirements of critical systems. The Linux-embedded Q-fabric is partially complete. Its kernel-to-kernel information transport, termed KECho [7], has been used for the dynamic, end-to-end control of interactive video applications running across multiple machines. Its ECalls [8] facility has been shown to improve predictability in application behavior under high load conditions, using the example of a web server's responses to its current vs. newly arriving requests.
Previous work. Our research is based on our previous work with distributed, real-time and adaptive systems, with sensor-based embedded applications [10], and with high performance middleware [2]. Previous and ongoing related work performed by project members includes funding from DARPA's Quorum, PCES, and embedded systems programs. We are also leveraging prior large-scale funding from DARPA and from the National Science Foundation that has created the basic publish/subscribe middleware used in this work [2, 13]. Concerning compiler technologies for embedded devices, we have investigated new static analysis techniques like on-chip memory data allocation [11], restructuring for code compaction [5], and efficient data layouts for indirect addressing modes [9]. The InfoFabric project goes beyond such work by using a paradigm of loading mobile code on a networked embedded device 'just-in-time' and by proposing notions of slices and its associated dynamic optimizations to generate efficient code for dynamic embedded systems.

2 InfoFabric Basics

InfoFabric targets applications comprised of distributed information sources, transformers, and information sinks. Sources could be inputs by humans operating electronic devices or automated sensors that capture information from the physical environment. The information produced by such sources must generally be delivered to multiple sinks, where this information must be transformed, fused, and filtered, so that it arrives in forms useful to end-users. These 'services' applied to information flows must be performed within timing constraints determined not only by data and sensor types, but also by current end-user and situational needs. Such constraints must be continuously met, despite dynamic changes in the locations and capabilities of the sources, sinks, services, and transports applied to information.

We use the term InfoFabric (1) to indicate that a distributed source-sink system operates as a whole, where multiple nodes jointly implement complex services provided for multiple end-users, and (2) to deal with the high levels of uncertainty on wireless distributed platforms, by casting a 'fabric' of information streams across these systems. The idea is to permit applications to exploit redundancy in information sources, utilize the platforms' concurrent communication paths and processing nodes, spread the implementations of the services they provide across multiple nodes, and replicate information to enhance availability and reduce delays.

2.1 Linking Communication and Computation

Information providers and consumers subscribe to shared logical communication *channels*. Existing Java- or CORBA-based implementations of such publish/subscribe paradigms typically use concentrators to collect and re-send data sent to a channel. In contrast, InfoFabric uses direct source-to-sink links between all providers and consumers of a channel. The information on these channels is represented by *self-describing* information items, where the types of information items being transported are identified to anyone receiving these items and must match the in/out parameter types of the computations performed on these items, thus creating a tight and well-defined linkage of communication with computation. An InfoFabric *service* is defined as a meaningful set of computations applied to information items. Since such a service is typically comprised of multiple *code modules* spread across InfoFabric nodes, both the transport of information and the computations being performed on them are inherently distributed. Finally, since both the information flows in an InfoFabric and the services applied to them are well-specified and known at run-time, the operation of the InfoFabric can be changed dynamically. Such changes are made via run-time adaptations that take advantage of the rich set of meta-information about typed information flows, items, services, and code modules in the InfoFabric.

2.2 Dynamic Adaptation

The automation of resource management is based on what is most important to end-users, which is the 'quality' of the information delivered by the fabric. Since 'quality' is a vague term, we develop concrete application-relevant characterizations of information quality. One approach is to create new methods for managing such quality via adaptive redundancy and replication:
(1) Information with 'quality' attributes – the *quality* of information refers to the degrees of similarity and consistency of the information acquired, processed, and delivered by an InfoFabric. An example of 'similar' information is a complete list of targets to render on a graphical display vs. a partial list (e.g., only the nearest ones). Consistency may be measured as information timeliness. Defined in this fashion, quality is an application-specific multi-dimensional metric that captures both information completeness and timeliness, thereby enabling tradeoffs across both.
(2) Adaptive replication and redundancy – to improve the performance of distributed applications, adaptive, distributed algorithms can be developed to change the ways in which information is routed and processed across a fabric. For instance, since raw image data may be available from multiple sources, one can exploit such redundancies to improve the likelihood and timeliness of delivering image information with 'good' quality to certain sinks. One approach is to simply move and compute data redundantly and

simultaneously along multiple paths of the InfoFabric.

2.3 Dynamic (Re-)Partitioning and (Re-)Location

The services (i.e., computations) an InfoFabric applies to its information flows are well-defined from the points of view of end users, but their internal composition, location, and runtime behavior are easily and dynamically varied, automatically and often invisibly. Our work goes beyond parameter- or mode-based adaptations of services, by dynamically generating, re-generating, and specializing the actual code that implements services. Specifically, in order to perform fine-grain service (re-)location and (re-)partitioning, we maintain meta-information about each service's code modules, including their locations, their relationships to each other, and their internal structures. Such meta-information will be sufficiently detailed to not only enable the run-time (re-)location of a service's statically defined code modules, but also to 'take apart' and 're-assemble' code modules to better match an information flow's transport and operation to the current capabilities of the underlying embedded system. The resulting 'lightweight' dynamic service (re-)partitioning methods will enable InfoFabric applications to operate with degrees of flexibility akin to those of Java-based systems. For instance, by employing runtime binary code generation based on precise, compiler-level intermediate specifications of a service's code modules and their internal representations, the service may be quickly (re-)partitioned to match the remaining power budget on an underlying computational node or to match the amounts of information flowing across certain InfoFabric links to the link bandwidths that are currently available.

2.4 Online Resource Management with Q-Fabric

InfoFabric adaptation cannot be performed without mechanisms for run-time monitoring and program adaptation. Such mechanisms, termed *Q-Fabric*, are being developed for the operating systems that underly the InfoFabric middleware, complementing user-level support that is based on previous research done by our group on lightweight mechanisms for system monitoring and adaptation [3]. One important attribute of this work is that the Q-fabric's resource management is performed with the same set of publish/subscribe mechanisms that implement the InfoFabric middleware itself, thereby avoiding costly (in terms of footprint) mechanism duplication[7]. Since energy is considered a first-class resource in our work, we continue with a description of a specific implementation that couples Q-fabric's operation with power-aware application processing and data transmission.

For applications using the InfoFabric middleware, Q-fabric is used to adjust the execution of their event handlers and filters to satisfy a device's need for low energy consumption. Three elements are considered to preserve energy in power-limited devices: (a) power-aware handler execution, (b) power-aware filter selection, and (c) combinations thereof. Power-aware handler execution [6] addresses scenarios in distributed real-time systems, where the early arrival of real-time events can be exploited to slow down the CPU (and hence, event handler execution), thereby reducing energy requirements. Power-aware filter selection focuses on situations where the *type* or *location* of a filter can affect the energy requirements of an application. For example, in multimedia applications, filters can change color depths, image sizes, or image resolutions, all resulting in reduced image quality, but with different effects on a device's energy requirements. Finally, both power-aware filtering and event handling considered together raise the need for distributed systems to *cooperate* in finding optimal handler/filter settings, to ensure that a distributed application's power requirements are met. These can include (a) achieving a low overall energy consumption for all involved devices, or (b) maintaining similar levels of the remaining battery power on all devices.

As an example, consider real-time events for multimedia applications such as video conferencing. Here, the video replay at the receiver side can be 'delayed' if events, i.e., video frames, arrive before their derived deadlines (e.g., frames in a video stream with a replay rate of 20 frames per second have deadlines $50ms$ apart from each other). Frames submitted and received on a StrongARM SA1110-based handheld device via the Q-fabric are processed and displayed, resulting in the power consumption shown in Figure 1. Note that the actual frame replay starts after 21s, and before that time, the device establishes a Q-fabric with a video

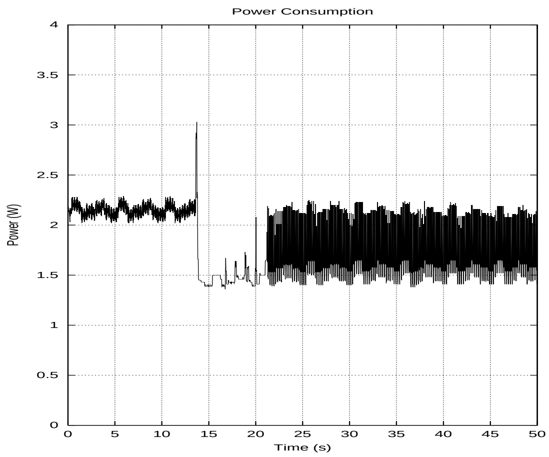

Figure 1. Power consumption without DFS.

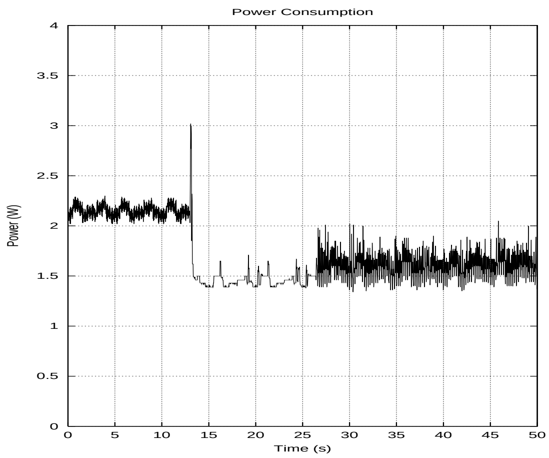

Figure 2. Power consumption with DFS.

server and disables the LCD display for better measurement results. The StrongARM SA1110 architecture allows for dynamic frequency scaling (DFS) in the range of 59MHz to 206MHz. While in the previous graph the device ran with the highest clock frequency, we now take advantage of early frame arrival and the architecture's ability to run at lower clock frequencies to reduce the overall power consumption of the device. Figure 2 repeats the previous experiment, however, this time using DFS, resulting in an average power consumption of 1.6W compared to 1.8W without DFS. To address variations in handler executions caused by varying event content or resource contention, the Q-fabric dynamically re-considers the chosen clock frequency and adjusts it if found necessary. For example, *callbacks* from event handlers can initiate clock frequency evaluations, possibly resulting in a frequency adjustment, and therefore reducing the number of missed event deadlines. These callbacks can either be placed 'manually' by the handler developer (directly in the handler code) or 'automatically' by the CPU scheduler (i.e., when a handler is preempted, the CPU scheduler calls back into Q-fabric). More details about this work can be found in [6].

3 Conclusions and Future Work

Whether realized with CORBA or Java, the computing infrastructures being developed for embedded systems will have to have several of the capabilities offered by the InfoFabric being constructed in our work. Agility in terms of rapid code deployment and re-deployment is required to match program code to platforms. Adaptivity is required to match how services are run and applied to data to create time-critical information for end-users with the quality they need. Methods for dealing with rapid infrastructure changes, especially in wireless embedded systems, have to not only recover from such changes but then adjust middleware-level actions and behavior to such changes. Some of those methods require kernel-level support to be able to operate at all or at the high levels of granularity required by certain applications.

Our future work focuses on the rapid access to remote live (i.e., sensor) data. Enhancing the InfoFabric's current ability to dynamically deploy code to target platforms, we are developing power-aware code generation and specialization techniques to further refine the parameter- and version-based adaptations now used by InfoFabric applications. The vision is to automate key application-level adaptations, while exploiting redundancy in information sources, utilizing concurrent communication paths and processing nodes, spreading the implementations of services across multiple nodes, and replicating information to enhance availability and reduce end-to-end delays:

(1) Adaptive replication and redundancy. We will improve application performance by developing distributed algorithms that change the ways in which information is routed and processed across a distributed embedded platform, resulting in *overlay*

networks [1].

(2) Improved service through dynamic adaptation. The services (i.e., computations) an InfoFabric applies to its information flows present well-defined interfaces to end-users, while at the same time, their internal composition, location, and run-time behavior are easily and dynamically varied, automatically and invisibly to end-users. Novel 'in place' and 'remote' compilation techniques will generate new code to be deployed at certain overlay network nodes.

(3) Fine-grain service (re-)location and (re-)partitioning. Middleware and code generation will not only enable a component of an InfoFabric service to be re-located, but also to be 'taken apart', 're-assembled', and specialized to better match an information flow's transport and operation to the current capabilities of the underlying embedded system. The lightweight dynamic service (re-)partitioning methods resulting from this research will enable embedded software to operate with degrees of flexibility akin to those of Java-based systems. Experimental results attained with a Java-based version of the InfoFabric [12] demonstrate that automatic handler partitioning via compiler methods [13] can substantially improve the performance of information flow-based applications, in wired and wireless systems.

Interface with standard platforms. .NET and CORBA-based interfaces are being constructed for InfoFabric software. Q-fabric's implementation is based on RedHat Linux rather than using specialized, real-time Linux kernels. The intent is to avoid a potential divergence of our work from ongoing university and industry Linux OS developments.

References

[1] D. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[2] G. Eisenhauer, F. Bustamente, and K. Schwan. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing (HPDC-2000)*, 2000.

[3] G. Eisenhauer and K. Schwan. An object-based infrastructure for program monitoring and steering. In *Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools (SPDT'98)*, pages 10–20, August 1998.

[4] A. Gavrilovska, K. Schwan, and V. Oleson. Adaptable mirroring in cluster servers. In *Proceedings of the 10th Intl. Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.

[5] V. Jain, S. Rele, S. Pande, and J. Ramanujam. Code restructuring for improving real time response through code speed, size trade-offs on limited memory embedded dsps. *IEEE Transactions on CAD*, 20(4):477–494, April 2001.

[6] C. Poellabauer and K. Schwan. Power-aware video decoding using real-time event handlers. In *Proc. of the 5th Intl. Workshop on Wireless Mobile Multimedia (WoWMoM), Atlanta, Georgia*, September 2002.

[7] C. Poellabauer, K. Schwan, G. Eisenhauer, and J. Kong. Kecho - event communication for distributed kernel services. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS'02), Karlsruhe, Germany*, April 2002.

[8] C. Poellabauer, K. Schwan, and R. West. Lightweight kernel/user communication for real-time and multimedia applications. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001. Port Jefferson, New York.

[9] A. Rao and S. Pande. Storage assignment optimizations to generate compact and efficient code on embedded dsps. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99)*, pages 128–138, June 1999.

[10] D. Rosu, K. Schwan, S. Yalamanchili, and R. Jha. On adaptive resource allocation for complex real-time applications. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS)*, December 1997.

[11] W. Tembe, L. Wang, and S. Pande. Data I/O minimization for loops on limited on-chip memory embedded processors. To appear in IEEE Transactions on Computers.

[12] D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. Jecho - supporting distributed high performance applications with java event channels. In *Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2000)*, 2000.

[13] D. Zhou, K. Schwan, G. Eisenhauer, and Y. Chen. Supporting distributed high performance application with java event channels. In *Proceedings of the 2001 International Parallel and Distributed Proceeding Symposium (IPDPS)*, April 2001.