ServiceFORGE: A Software Architecture for Power and Quality Aware Services

Radu Cornea, Nikil Dutt, Rajesh Gupta*, Shivajit Mohapatra, Alex Nicolau, Cristiano Pereira*, Sandeep Shukla**, Nalini Venkatasubramaniam

University of California, Irvine, * University of California, San Diego
** Virginia Tech. University, Blacksburg, Virginia

Abstract. We present a novel power management service in QoS-brokerage architecture that relies on multi-level middleware services to act as brokers in delivering multimedia content in distributed real-time systems under performance and power constraints. To minimize power consumption, the power management service provides for minimal interaction among the system components as well as provides dynamic control of speed scaling (such as voltage, frequency) and component shutdown. These power control "knobs" enable the service to match instantaneous application needs against available energy resources. The power management service is implemented using the Comp|OSE middleware framework using a power-aware API that allows the applications programmer to ensure a continuous dialogue between the distributed embedded systems and changing application needs. In this paper, we focus on the interfaces to the power management service that ensure semantically correct composability of power management actions in a distributed systems and its use in applications programming. We present example of a multimedia streaming server to handheld devices that demonstrates the use of automatic speed scaling knobs in minimizing energy consumption.

1. Introduction

Power management is important for a range of embedded applications from portable terminals to ad hoc sensor networks. The focus of much of the work in this area, so far has been on minimization of energy at the node level [[FengSechrest96], ChandraVahdat02]. For distributed embedded systems, the problem is much more complex because of the dynamic tradeoffs involved at several levels of abstraction between local processing and communication/coordination [Yuan et al 2003]. The correct way to think about it is to treat as a multi-level 'service' in a distributed embedded system [Cornea et al 03]. In this paper, we describe a software system architecture that enables the system architect to compose distributed power/performance related decision making and to ensure compliance with functionality and system energy constraints based on the runtime conditions. This is done by means of "brokers." These brokers are built in a model-based system specification that allows reasoning about the functional and non-functional properties of the system from the properties of the constituent components and the composition mechanism used; and to use a

middleware infrastructure that lends itself to platform specific optimization for performance and size. Specifically, we focus on adaptive and reflective middleware services to meet the application requirements and to dynamically smooth the imbalances between demands and changing environments.

Fig. 1 illustrates the fundamental levels of adaptation and reflection supported by middleware services: (a) changes in the middleware, operating systems, and networks beneath the applications to continue to meet the required service levels despite changes in resource availability, such as changes in network bandwidth or power levels, and (b) changes at the application level to either react to currently available levels of service or request new ones under changed circumstances, such as changing the transfer rate or resolution of information over a congested network. In both instances, the middleware must determine if it needs to (or can) reallocate resources or change strategies to achieve the desired QoS. Embedded applications must be built in such a way that they can change their QoS demands as the conditions under which they operate change. Mechanisms for reconfiguration need to be put into place to implement new levels of QoS as required, mindful of both the individual and the aggregate points of view, and the conflicts that they may represent.

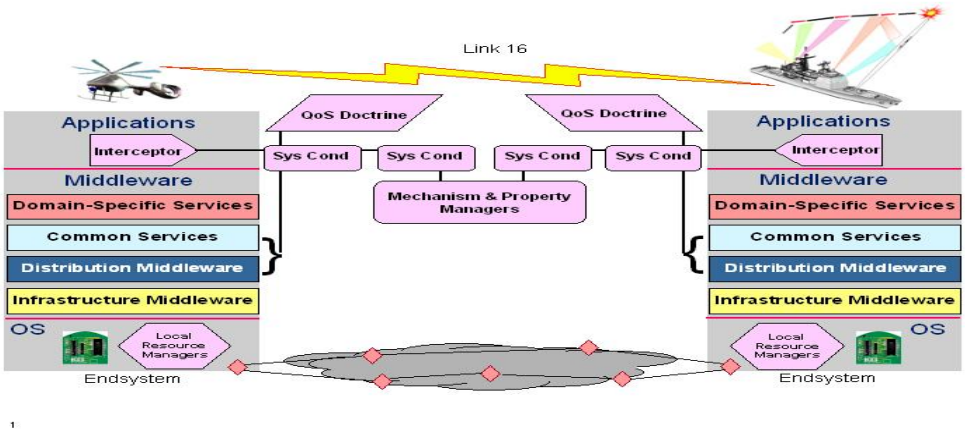

Fig. 1. Middleware Services and QoS Brokerage.

2. ServiceFORGE: A Software Architecture for Quality-Aware Services

We approach the application development in ServiceFORGE as not only specification of the desired functionality but also specification and management of a contract with

the underlying mobile software system on timing performance and energy needs. The underlying software is aware of the finite energy/power resources and makes use of its own contract and brokering services to adapt its functional needs to match low-level computing, communication and networking capabilities. Our strategy is based on building an application development framework that allows radically enhanced configurability and adaptability in pretty much all aspects of software and networking processes. These include reconfiguration algorithms that exploit adaptability in the various tasks that constitute the wireless protocols and management of application-level tasks to ensure efficient use of the dynamically changing battery resource as well as performance and energy requirements of the executing tasks. Our approach consists of two parts: power aware operating and runtime system services, and their interface to middleware services that ensure efficient brokerage of the application needs and system resource constraints.

Power Aware Operating and Runtime System Services

For effective system-level power management, it is important that an application is able to monitor and control both electrical knobs (such as voltage and scheduling), as well as exercise control on the task scheduling and shutdown states of different parts of the client device (node). The operating system is an important place for making power management decisions since it has the knowledge about timing, usage and traffic patterns of applications. We modify traditional OS services to make them power-aware vis-à-vis their execution so that energy is used in an efficient fashion. At the system-level, power management consists of three parts: (a) system components as *resources* that provide service such a computation or I/O and consume power, (b) application functionality as *tasks* that utilize the resources, and (c) system timing performance and result quality requirements as *service contracts*. Each resource may have one or more *operating modes* (e.g. a CPU core may be in run or shutdown mode, and a radio subsystem may be in transmit or receive or idle mode). The transitions between operating modes of a resource are dictated by the availability of tasks needing that resource, and the functional requirements of those tasks (e.g. a packet sender task would put the radio in the transmit mode). Further, a resource in each of these modes can be put in one of multiple operating points in a mode-specific *power-speed subspace* (and more generally *power-speed-quality subspace*) via "control knobs" that could be relatively universal or perhaps resource-specific. An example of a relatively universal control knob is the processor supply voltage. Supply voltage change, with a coordinated change in the clock frequency, leads to multiple power-vs.-speed points. The transitions between the power-speed-quality operating points will be dictated by the system timing performance and result quality requirements [Choi02]. At any given instant the system may be viewed to be in a specific *power state* corresponding to a specific permutation of the operation mode and power-speed-quality operating point for each resource. The key to system-level power/performance optimization is a power management control strategy consisting of task and resource specific algorithms to decide the power state evolution of the system. We describe our approach to control node-specific power/performance constraints in the next section, followed by a description of their integration with the middleware brokerage services using the example of a streaming video server.

3. Power Aware Nodal Services in ServiceFORGE

As mentioned earlier, the chief goal of power awareness at the node level is to enable **a continuous dialogue between the application, the OS, and the underlying hardware**. This dialogue establishes the functionality and performance expectations (or even contracts, as in real-time sense) within the available energy constraints. We describe here our implementation of a Power Aware Software Architecture (PASA). PASA is composed of two software layers and the RTOS kernel. One layer interfaces applications with operating system and the other layer makes power related hardware "knobs" available to the operating system. Both layers are connected by means of corresponding power aware operating system services as shown in **Fig. 2**. At the topmost level, embedded applications call the API level interface functions to make use of a range of services that ultimately makes the application energy efficient in the context of its specific functionality. The API level is separated into two sub-layers. PA-API layer provides all the functions available to the applications, while the other layer provides access to operating system services and power aware modified operating system services (PA OS Services). Active entities that are not implemented within the RTOS kernel should also be implemented at this level (threads created with the sole purpose of assisting the power management of an operating system service, such as a thread responsible for killing threads whose deadlines were missed). We call this layer the power aware operating system layer (PA-OSL).

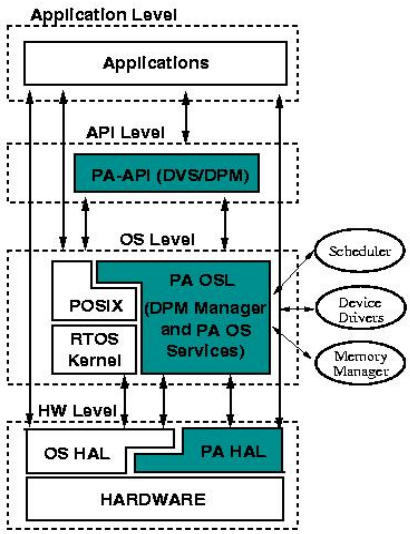

Fig. 2. Nodal Services and API for Power

To interface the modified operating system level and the underlying hardware level, we define a power aware hardware abstraction layer (PA-HAL). The PA-HAL gives the access to the power related hardware ``knobs" in a way that makes it independent of the hardware.

Table below lists the functions relevant to the implementation of power aware scheduling techniques. At the PA-API layer there are functions to create types (informing the real time related parameters) and instances of tasks, to notify start and end of tasks (needed by the OS in order to detect whether the task execution is over and the deadline of a task has been met), and to either inform the application about the execution time predicted by the OS or tell the OS about the execution time prediction estimated by the application (which can be based on application specific parameters). At the PA-OSL layer there are functions to manipulate information related to the power aware scheduling schemes that are maintained within the kernel (such as the type table in the case of the scheduler), the thread responsible for killing threads whose deadlines were missed (assuming that the threads

whose deadlines were missed are no longer useful). The alarm handler notifies the killer thread, which in turn kills the thread and re-creates it. The overhead of having an extra thread is minimal since the killer thread is constantly blocked unless a request to kill another thread is received. When it happens, the killer thread wakes up and finishes the execution of the proper thread. At the PA-HAL layer functions to manipulate processor frequency and voltage levels and low power states are present. These are called by the RTOS scheduler when slowing down the processor or shutting it down. For processor frequency and voltage scaling, different platforms have different precautions that have to be taken care of before doing the scaling. These precautions might have to be done before the scaling, after it or both before and after. For these the functions pahal_pre_set_frequency_and_voltage and pahal_post_set_frequency_and_voltage are provided and must be implemented by the OS programmer according to the platform. And finally functions to poll the status of battery based platforms are also important in order to enhance their lifetime.

Layer	Function name
PA-API	paapi_dvs_create_thread_type(), paapi_dvs_create_thread_instance() paapi_dvs_app_started(), paapi_dvs_get_time_prediction() paapi_dvs_set_time_prediction(),paapi_dvs_app_done(), paapi_dvs_set_adaptive_param(),paapi_dvs_set_policy(), paapi_dpm_register_device()
PA-OSL	paosl_dvs_create_task_type_entry(),paosl_dvs_create_task_instance_entry(), paosl_dvs_killer_thread(),paosl_dvs_killer_thread_alarm_handler(), paosl_dpm_register_device(), paosl_dpm_deamon()
PA-HAL	pahal_dvs_initialize_processor_pm(), pahal_dvs_get_frequency_levels_info() pahal_dvs_get_current_frequency(), pahal_dvs_set_frequency_and_voltage() pahal_dvs_pre_set_frequency_and_voltage(), pahal_dvs_post_set_frequency_and_voltage() pahal_dvs_get_lowpower_states_info(), pahal_dvs_set_lowpower_state() pahal_dpm_device_check_activity(), pahal_dpm_device_pre_switch_state() pahal_dpm_device_switch_state(), pahal_dpm_device_post_switch_state() pahal_dpm_device_get_info(), pahal_dpm_device_get_curr_state() pahal_battery_get_info()

The piece of code that follows shows an example on how the PA-API functions are used in a MPEG decoder source code when creating threads using PA-API functions. A thread is created specifying that the deadline and period are 100 and the worst case execution time is 30 (assuming it was profiled and therefore known ahead of time. The thread is instantiated and access to the power-aware functionality contracts is enabled and terminated by the functions paapi_app_started() and paapi_app_done() respectively. These functions delimit the work done by the threads which is encapsulated in one single function in this example.

```
void main() {
  mpeg_decoding_t =
    paapi_create_thread_type(100,30,100);
  paapi_create_thread_instance(mpeg_decoding_t,
mpeg_decode_thread); }
```

```
void mpeg_decode_thread() {
  for (;;) {
    paapi_app_started();
    /* original code */
    mpeg_frame_decode()
    paapi_app_done();   }}
```

This provides a generic dynamic power management (DPM) API sufficient to support different devices and DPM policies by using a common set of functions. The API also provides a common framework for implementing new (DPM) policies. For DPM purposes, each device is registered with the power manager and with each device we attach enough information to execute whichever policy the device was registered to be managed with. Often device DPM techniques switch devices to low-power modes or states based on how long the device has been idle. For instance, threshold values are defined for each device so that the longer the device is idle, the deeper the sleep mode it is switched to. A common set of functions and data structures have to be defined in order to manage such devices. These furnish the implementation of the DPM techniques and provide the guidelines for implementing new ones. Some of the functions defied for this purpose are listed in the previous table and are shortly described below:

- `dpm_device_check_activity()` - This function finds out whether the device was activated or has been idle since the last time it was queried. To do that, a device activity structure has to be kept and has to be compared against a new activity information every time the device is queried (on the embedded Linux platform, for instance, this information comes from the /proc virtual file system interface. For other operating systems without such interface this information has to be kept by the API in form of tables in order to track the device activity). If the amount of activity is the same it means that nothing has happened since the last time it was checked. Otherwise some activity has happened and the stored information is updated. This function makes sense only if the policy used is based on the activity information (kept in a device status table).
- `dpm_device_switch_state()` - This function will switch the state of the device from origin state to destination state.
- `dpm_device_get_info()` - It gets information about the device.
- `dpm_device_get_state()` - It tells in which state the device is in currently.
- `dpm_device_register()` - It registers the device along with the appropriate functions and power management policy to handle it. This information is kept in the device info table.

Structures containing the possible states each device can be at, as well as which policies, are attached to each device and the information needed to implement such policies are defined and kept within the kernel. The kernel DPM entity consists of tasks associated with each device and implementing a specific policy for managing the device. If all devices use the same policy then multiple instances of this policy are created and they manage each device individually.

The piece of code below shows a threshold based dynamic power management scheme. For each device state there is an associated threshold which defines when to switch to that state as described in the lower envelope algorithm.

```
void threshold_policy_deamon(device_info_t dev){
  unsigned idleness;
  for (;;) {
    /* check for how long the device has been idle */
    idleness = dev->check_activity(dev);
    /* if idle for longer than the threshold
       switch to   next state */
    if ( idleness > dev->check_state()->threshold ) {
      dev->check_state()->switch_state(dev,
         dev->check_state, dev->check_state()->next); }
    /* sleep until next period for checking idleness */
    sleep(dev->policy_info->th_policy->period); }}
```

When mixing DPM and DVS algorithms in the same platform there is a tradeoff on whether to slowdown as much as possible or to execute some tasks faster than the minimum possible frequency and rearrange the idle times in order to get better changes to shutdown some of the system components. In [IraniShuklaGupta03], we have devised an algorithm to optimize this tradeoff on a system. The algorithm, named as *Procrastinator*, adjusts (or procrastinates) start times and deadlines of some of the tasks in order to create longer idle times and obtain more chances to wisely bring the device to a low power mode. The PASA architecture and its API make it possible to utilize the combined DVS and DPM opportunities and improve their effectiveness with additional information available from the application itself.

4. Middleware Power/Performance Brokerage Service in ServiceFORGE

Node level dynamic power management described in the previous section is only a first step in achieving an application-level control of the system power/performance tradeoffs. We approach the distributed power/performance optimization problem as one of middleware services that interact with the node level services to make the right tradeoffs in the context of application behavior and its needs [Mohapatra et al 03].

To explain this, let us use the example of a multimedia streaming from a server to a set of mobile handheld devices. The system architecture shown in **Fig. 3** below consists of a multimedia server, a proxy server (that adapts the video stream to client capabilities), a wireless access point and the clients (low-power wireless devices). The multimedia server streams videos to clients on request from users. All communication between the servers and clients are routed through the proxy server, which can transcode the video stream in real time.

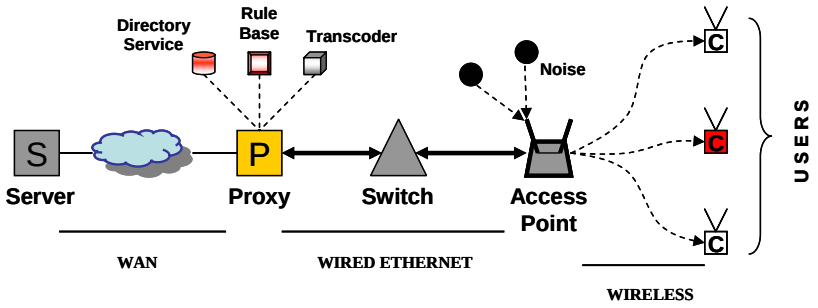

Fig. 3. Streaming Multimedia Example

Middleware level components (services) execute on both the handheld device and the proxy, performing two important functions: On the device, it sends residual energy availability information the proxy and relates video stream parameters and network control information to lower abstraction layers. This information is conveyed using the PASA API for the HAL/OSL layers. On the proxy, it performs a feedback based power aware admission control and real time transcoding of the video stream based on the feedback from the device. It also controls the video transmission over the network based on the load on the network and video stream quality level. To illustrate the application control of the client power and fairness of the service to multiple clients, we use quality level of the video (specified as PSNR).

Fig. 4 shows the overall ServiceFORGE architecture. To implement the desired level of power/performance control, we assume that the following levels of abstraction apply to both the proxy and the clients: architecture, operating system, middleware, application. Each of these levels has components/services interacting with corresponding services on the same level or with components at a different level of abstraction. The architecture level includes most of hardware components: CPU with memory, display, network card, etc. OS level provides the scheduler, DVS control, power-aware APIs and other OS level services. The middleware level provides a series of services, like network management, transcoding, admission control, mobility information, etc. The video application runs at the application level, with the other tasks running on the device.

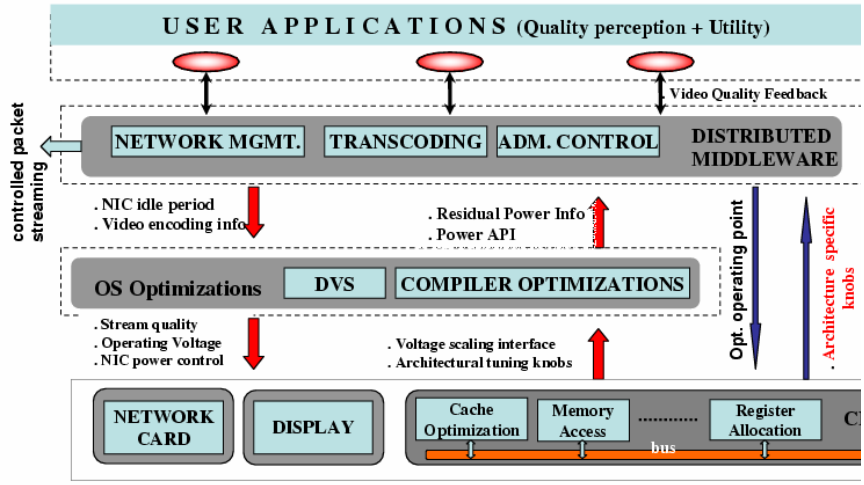

Fig. 4. Architecture of ServiceFORGE

Components and services at different level of abstraction interact together with a final goal of improving the overall system performance, including power, deadlines and quality of service. There are various control knobs available at the device: CPU voltage and frequency scaling, memory system configuration, network card access pattern. The video stream can be controlled through its encoding parameters: frame rate, bitrate, and frame rate. Each of these parameters are controlled by the middleware services and adapted as required by the runtime conditions in the system. For example, if the residual energy available at a mobile device drops, the control decreases the quality of the video stream by lowering its frame rate or one of the other parameters (frame size, bitrate). Similarly, when a new user joins the system, resources need to be freed in order to accommodate the new node in the network (allocate network bandwidth, transcoder CPU time, etc.).

The middleware services also control network transmission. To save power, video stream data is grouped into short burst transmissions and sent periodically over the network. This allows the network card at the device to go into longer periods of low-power sleep mode [Shenoy03].

Each component in the abstraction hierarchy provides services to the other components on the same node or on other nodes in the network. During runtime, there is a continuous exchange of information and control between nodes to ensure that the constraints imposed on the system are met and quality of service is preserved for all the clients. If for some reasons these conditions cannot be satisfied the admission control component may decide to renegotiate video quality levels with all users in the system.

Experiments Using Video Server Example

We performed several experiments to evaluate power savings and performance improvements at different levels of abstraction as well as globally for the entire system. At the architecture level, we selected cache parameters as optimizing knobs and profiled video clips for a large space of cache configuration points. **Fig. 5** shows results from configuring the data cache to meet the requirements of particular video streams. These changes alone can yield 10-15% in power savings. Combining frequency and voltage scaling with cache reconfiguration increase the opportunities for power savings, as the processor can be run at a lower voltage and frequency when decoding less complex frames. This combined approached yields up to 60% in energy savings as compared with the initial architecture.

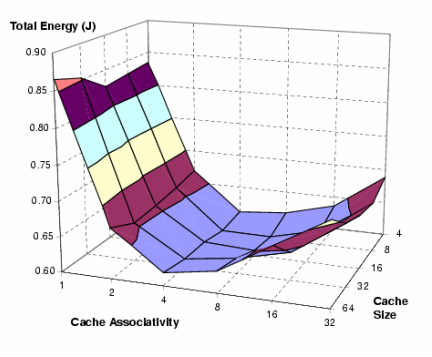

Fig. 5. Cache Optimization Search Space

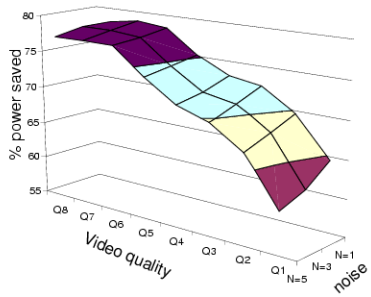

Fig. 6. Power Savings in the Network Card

Fig. 6 shows that at the network level we obtained up to 70% power savings by sending optimized bursts of video and turning the network interface off (sleep mode) between consecutive bursts. The ideal burst time was computed for each quality level and for different network load - users in the network are modeled as noise. Finally, we evaluate the performance of the integrated framework. Our goal is to provide an optimal user experience and maintain an acceptable utility factor for the system. We define an "*acceptable utility factor*" to be obtained when the system can stream the highest possible quality of video to the user such that time, acceptable quality and power constraints are satisfied (i.e the video clip runs to completion, at a quality level above or equal to the one the user specified, the difference between the two defining the final utility factor). To accomplish this it is important to understand the notion of video quality for a handheld device and its implications on power consumption. **Fig. 7** shows how adaptive middleware provided by ServiceFORGE can improve the utility factor for the integrated framework.

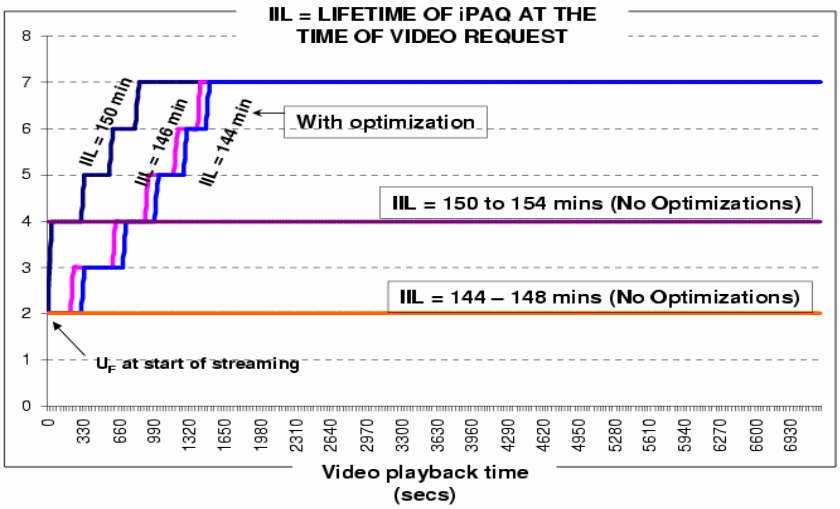

Fig. 7. Utility Factor over Time.

5. Summary and Conclusions

Ensuring best system performance in presence of very real resource constraints in distributed embedded systems is a difficult problem. Solving this problem requires analysis of available system resources, application needs in presence of dynamically changing operating conditions. Instead of seeking optimization techniques, our approach is to enable application participation with the runtime systems in setting the appropriate resource utilization policies. Towards that end, we have built ServiceFORGE to provide two basic capabilities: capability for the middleware to carry out a dialogue with the application in determining its needs and conveying these through a structured interface to individual nodes; and the capability for the individual nodes to change performance/power usage knobs based on the middleware directives. Early experiments suggest that this architecture can be useful in achieving better quality of results for the same power budgets in the case of streaming video. Additional experimentation across various application domains is necessary to understand how application programming can be structured to take advantage of the new services in the system software.

References

[ACEORB] Center for Distributed Object Computing, "The ACE ORB (TAO)" www.cs.wustl.edu/_schmidt/TAO.html Washington University.

[aspectGAMMA02] M. Mousavi, G. Russello, M. Chaudron, M.A. Reniers, T. Basten, A. Corsaro, S. Shukla, R. Gupta, D. Schmidt, "Aspects+GAMMA=AspectGAMMA: A Formal Framework for Aspect-Oriented Specification", presented at the Early Aspects Workshop, Twente, Netherlands, April 2002.

[Balboa] Balboa Project. Component Composition Enviornement Home page: http://www.cecs.uci.edu/_balboa.

[Banatre93] Jean-Pierre Banatre and Daniel Le Metayer, Programming by multiset transformation, Communications of the ACM (CACM), 36(1):98--111, January 1993.

[Bapty et al, 2000] Bapty T., Neema S., Scott J., Sztipanovits J., Asaad S, "Model-Integrated Tools for the Design of Dynamically Reconfigurable Systems", VLSI Design, 10, 3, pp. 281-306, 2000.

[Birkhoff1933] G. Birkhoff. On the Combination of Subalgebras. Proceedings of Cambridge Philosophical Society, 1933.

[Blair et al 98] Gordon S. Blair, G. Coulson, P. Robin, and M. Papathomas, "An architecture for next generation middleware," in Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer-Verlag, London, 1998.

[Boehm80] Boehm, B. Software Engineering Economics, Prentice Hall, 1980.

[Bol00] Bollella, G., Gosling, J. "The Real-Time Specification for Java," Computer, June 2000.

[Booch98] Grady Booch, Ivar Jacobson, James Rumbaugh, Jim Rumbaugh "The Unified Modeling Language User Guide", The Addison-Wesley Object Technology Series, 1998.

[BroyKrueger98] M. Broy, I. Krüger: Interaction Interfaces - Towards a scientific foundation of a methodological usage of Message Sequence Charts, in: J. Staples, M. G. Hinchey, Shaoying Liu (eds.): Formal Engineering Methods (ICFEM'98), IEEE Computer Society, 1998

[BroyStoelen01] M. Broy, K. Stølen: Specification and Development of Interactive Systems. Focus on Streams, Interfaces, and Refinement. Springer, 2001

[ChandraVahdat02] S. Chandra and A. Vahdat. "Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats". In Usenix Annual Technical Conference, June 2002.

[Chaudron98] Chaudron, M. R. V, "Separating Computation and Co-ordination in the Design of Parallel and Distributed Systems", Ph.D thesis, Leiden University, 1998.

[Chaudron94] Michel R.V. Chaudron, Schedules for Multiset Transformer Programs, Technical Report no 94-36, Department of Computer Science, Leiden University, December 1994.

[Choi02] K. Choi, K. Dantu, W.-C. Chen, and M. Pedram. "Frame-Based Dynamic Voltage and Frequency Scaling for a MPEG Decoder". In ICCAD 2002.

[Chou94] P. Chou, and G. Boriello, " Software Scheduling in the co-synthesis of Reactive Real-Time Systems", in Proceedings of the 31st Design Automation Conference, 1994.

[Corba 2000] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.4 ed.", Oct. 2000.

[Cornea et al 03] R. Cornea, N. Dutt, R. Gupta, I. Krueger, A. Nicolau, D. Schmidt, S. Shukla, "FORGE: A Framework for Optimization of Distributed Embedded Systems Software", IPDPS 03.

[Culler et al 01] David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices",in DARPA workshop on Embedded Software, 2001.

[Donahue et al 2001] S. M. Donahue, M.P. Hampton, M. Deters, J. M. Nye, R.K. Cytron, and K. M. Kavi, "Storage allocation for real-time, embedded systems," in Embedded Software: Proceedings of the First International Workshop (T.A. Henzinger and C.M. Kirsch, eds.), pp.131-147.

[Donahue et al 2002] S. Donahue, M. Hampton, R. Cytron, M. Franklin, and K. kavi, "Hardware support for fast and bounded-time storage allocation," Second Annual Workshop on Memory Performance Issues (WMPI 2002), 2002

[Doucet-date02] F. Doucet, R. Gupta, M. Otsuka, S. Shukla, "An Environment for Dynamic Component Composition for Efficient Co-Design", Accepted for presentation at the Design Automation and Test Conference (DATE 2002), Match 2002.

[Eme90] E. Allen Emerson. Temporal and Modal Logic. In Jan van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, pages 995--1072. Elsevier, 1990.

[Esterel Tech] Esterel Technologies Web Page, http://www.esterel-technologies.com/

[FengSechrest96] W.chi Feng and S. Sechrest. "Improving data caching for software mpeg video decompression". In IS&T/SPIE Digital Video Compression: Algorithms and Technologies, 1996.

[FinkbeinerKrueger01] B. Finkbeiner, I. Krüger: Using Message Sequence Charts for Component-Based Formal Verification. Specification and Verification of Component-Based Systems (SAVCBS). Workshop at OOPSLA 2001.

[Frappier 2000] Marc Frappier, Henri Habrias , “Software Specification Methods : An Overview Using a Case Study (Formal Approaches to Computing and Information Technology”, Springer Verlag, November 2000.

[Gal01] Andreas Gal, Wolfgang Schroder-Preikschat, and Olaf Spinczyk, “On Aspect Orientation in Distributed Real-Time Dependable Systems”, "On Aspect-Orientation in Distributed Real-time Dependable Systems", Accepted at the Seventh IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2002) , San Diego,CA, January 7-9, 2002

[GarlanAllenOckerbloom95] D. Garlan, R. Allen, and J. Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. IEEE Software, November 1995.

[Genssler et al 2002 T. Genssler, O. Nierstraszand B. Schoenhage. Componenets for embedded software: The pecos approach. In Proc. Int. Conf. On Compilers, Architecture, and Systhesis for Embedded Systems, 2002.

[Gill et al 2001] Chris Gill, David Levine, and Douglas C. Schmidt, “The Design and Performance of a Real-Time CORBA Scheduling Service,” The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, guest editor Wei Zhao, Volume 20, Number 2, March 2001.

[GorlickRazouk91] M. M. Gorlick and A.R.R. Razouk. Using weaves for software construction and analysis. In Proc. Int. Conf. On Software Engineering, 1991.

[Grundy99] Jim Grundy, “Aspect Oriented Requirements Engineering for Component Based Software Systems”, In the Proceedings of Requirements Engineering (RE’99), June, 1999, Limerick, Ireland, IEEE Press.

[GunterMuschollPeled01] E. Gunter, A. Muscholl, and D. Peled. Compositional Message Sequence Charts. In Proc. of TACAS’01, volume 2031 of Lecture Notes in Computer Science, pages 496–511. Springer, 2001.

[Harrison et al 97] Tim Harrison and David Levine and Douglas C. Schmidt, “The Design and Performance of a Real-time CORBA Event Service,” Proceedings of OOPSLA ‘97, ACM, Atlanta, GA, October 1997.

[Henzinger98] Thomas A. Henzinger, It's about time: Real-time logics reviewed., In Davide Sangiorgi and Robert de Simone, editors, Proceedings of Ninth International Conference on Concurrency, volume 1466 of LNCS, pages 439–454. Springer-Verlag, Nice, France, 1998.

[Henzinger01] Thomas A. Henzinger, Ben. Horowitz, and Christoph M. Kirsch, “Giotto: A Time Triggered Language for Embedded Programming”, In the Proceedings of the First International Workshop on Embedded Software (EMSOFT’01), Lake Tahoe, CA, USA, October 2001.

[Hoare85] Communicating Sequential Processes, Prentice Hall, 1985.

[Huang et al 97] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhaonad R. Bettati, “RT-ARM: A Real-Time Adaptive Resource Management system for Distributed Mission-Critical Applications,” in Workshop on Middleware for Distrbuted Real-Time Systems, RTSS-97, (San Francisco, CA), IEEE, 1997.

[IDL/OMG] OMG Website, http://www.omg.org.

[IraniShuklaGupta03] Sandy Irani, Sandeep Shukla and Rajesh Gupta. “Algorithms for Power Savings, SODA 2003.

[JainSchmidt97] P. Jain and D. C. Schmidt, “Service Configurator: A Pattern for Dynamic Configuration of Services: in Proceedings of the 3rd Conference

on Object-Oriented Technologies and and Systems, USENIX, June 1997.

[Kiczales 97] G. Kiczales, "Aspect-Oriented Programming," in Proceedings of the 11th European Conference on Object-oriented Programming, June, 1997

[Koskimies et al. 98] Kai Koskimies, Tarja SystÄa, Jyrki Tuomi, and Tatu Männistö. Automated Support for Modeling OO Software. IEEE Software, pp. 87–94, January—February 1998.

[Krueger00] I. H. Krüger: Distributed System Design with Message Sequence Charts, Dissertation, Technical University of Munich, 2000, available at: http://tumb1.biblio.tu-muenchen.de/publ/diss/in/2000/krueger.html

[Lamsweerde00] Axel v. Lamsweerde, "Formal specification: a roadmap", in Anthony Frankelstein ed., The Future of Software Engineering, ACM Press, 2000.

[LeeXiong01] E . A. Lee and Y. Xiong. System-Level Types for Component-Based Design. In First International Workshop on Embedded Software, vol.2211 of Lecture Notes in Computer Science. Springer, October 2001.

[LiaoTjiangGupta97] S. Liao, S. Thiang, and R. Gupta. An Efficient Implementation of Reactivity in Modeling Hardware in the Scenic Synthesis and Simulation Environment. In Proc. IEEE/ACM Design Automation Conf., 1997.

[Linda 93] Bjornson Robert, "Linda on Distributed Memory Multiprocessors", PhD thesis, Yale University, 1993.

[Loyall et al 01] J. Loyall, J. Gossett, C. Gill, R. Schantz, J.Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi and D. Karr, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," in Proceedings of the 21st International conference on Distributed Computing systems (ICDCS-21), pp.625-634, IEEE, April 2001.

[Manna, Pnueli] Zohar Manna and Amir Pnueli, "The temporal Logic of reactive and concurrent systems", Springer Verlag, 1992

[Metropolis] Metropolis Project Web Page, http://www.gigascale.org/metropolis/infrastructure.html

[Microsoft Dnet 01] Microsoft, Microsoft(r) .NET My Services Specification, Microsoft Press, October 2001

[Misra, Chandy 88] J. Misra and K. M. Chandy, Parallel Program Design: A Foundation, Addison-Wesley, 1988.

[Mohapatra et al 03] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, N. Venkatasubramanian, "Integrated Power Management for Video Streaming to Mobile Handheld Devices, ACM Multimedia 2003.

[Moml 2000] Steve Neuendorffer, Ed Lee, "MoML: An XML Modeling Markup Language", http://buffy.eecs.berkeley.edu/IRO/Summary/00abstracts/neuendor.1.html

[Morse96] J. Morse, and S. Hargrave, "The increasing importance of Software", Electronic Design, vol.44(1), Jan. 1996.

[Mousavi01] Mousavi, M.R., Rusello G., and Chaudron M. R. V, " A Coordination Approach for the Design of Component Based Distributed Real-Time Systems", submitted.

[Mousavi et al 2002] M. Mousavi, M. Chaudron, G. Russello, M. Reniers, T. Basten, A. Corsaro, S. Shukla, R. Gupta and D. Schmidt. Using aspect –GAMMA in

Design and Verification of Embedded systems. In Proc. High level Design Validation and Test Workshop, 2002.

[O'Ryan et al 2000] C. O'Ryan, D. C. Schmidt, F. Kuhns, M. Spivak, J. Parsons, I. Pyarali, D. Levine, "Evaluating Policies and Mechanisms for Supporting Embedded, Real-Time Applications with CORBA 3.0" in Proceedings for the 6th IEEE Real-Time Technology and Applications Symposium, (Wash. D.C.), IEEE, May 2000.

[Omg99a] Object Management Group, Real-time CORBA Joint Revised Submission, OMG Documentorbos/99-02-12 ed., March 1999.

[Omg99b] Object Management Group, "Dynamic Scheduling, OMG document orbos/99-03-32 ed., March 1999.

[Omg2000] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.4 ed., October 2000.

[Omg01a] Object Management Group, "The Common Object Request Broker: Architecture and Specification Revision 2.5, OMG Technical Document formal/00-11-07", October 2001.

[Omg01b] Object Management Group, "The Common Object Request Broker: Architecture and Specification, 2.6 ed., December 2001.

[Paulin97] P. Paulin, C. Liem, M. Cornero, F. Nacabal, G. Goossens, "Embedded Software in real-time signal processing systems: Application and architectural Trends", Proceedings of IEEE, vol. 85(3), 1997.

[Pyarali+02] Irfan Pyarali, Douglas C. Schmidt, and Ron Cytron, "Techniques for Enhancing Real-time CORBA Quality of Service," Submitted to the IEEE Proceedings. Available at http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html. p. 419-435.

[Ramanathan TCAD2002] D. Ramanathan, S. Irani, R. Gupta, "An Analysis of System Level Power Management Algorithms and their effects on Latency", IEEE Transactions on Computer Aided Design, March 2002.

[RTCorba 2000] Object Management Group, "Dynamic Scheduling Real-Time CORBA Joint Revised Submission, OMG Document orbos/2000-08-12 ed.", August 2000.

[Saxena99] Saxena . A, Shukla. S, Weihmayer. R, Wu. P, "CORBA based Event Management System: A Case Study in Automatic Global Correlation", In the Proceedings of the International Conference on Parallel Processing Techniques and Applications (PDPTA'99), CRA Press, Las Vegas, June 1999.

[Schantz 2002] Richard E. Schantz and Douglas C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," Encyclopedia of Software Engineering, Wiley and Sons, 2002.

[Schmidt et al 2000] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Vol. 2. New York: Wiley & Sons, 2000.

[SchmidtKuhns2000] D. C. Schmidt and F. Kuhns "An Overview of the Real-time CORBA Specification" IEEE Computer Magazine, Special Issue on Object-oriented Real-time Computing, vol.33, June 2000.

[Schmidt et al 2001] D. C. Schmiddt, S. Mungee, S. Flores-Gaitan, and A. Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," Journal of Real-time Systesm, special issue on Real-time Computing in the Age of the Weg and the Internet, vol.21, no.2, 2001.

[Schmidt 2001] Douglas C. Schmidt, Sumedh Mungee, Sergio Flores-Gaitan, and Aniruddha Gokhale, "Software Architectures for Reducing Priority Inversion and Non-determinism in Real-time Object Request Brokers," Journal of Real-time Systems, Kluwer, Vol. 21, No. 2, 2001.

[SelicGulleksonWard94] Bran Selic, Garth Gullekson, and Paul T. Ward: Real-Time Object-Oriented Modeling, Wiley, 1994.

[ShaRajkumarLehoczky90] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-time Synchronization," IEEE Transactions on Computers, vol. 39, September 1990

[Shenoy03] P. Shenoy and P. Radkov. "Proxy-Assisted Power-Friendly Streaming to Mobile Devices". In MMCN, 2003.

[Shukla98] S. Shukla "Fault-Tolerance Patterns for Network Management Applications", Invited Presentation at the Dagstuhl Seminar on Self-Stabilization, Dagstuhl, Germany, August 1998.

[Stankovik87] John A. Stankovic and Krithi Ramamritham, Tutorial on Hard Real-Time Systems, IEEE Computer Society Press, 1987.

[Szyperski98] C. Szyperski. Component software: Beyond Object Oriented Programming. Addison-Wesley, 1998.

[Thoen-Cathhoor00] Filip Thoen, and Francky Catthoor, "Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems", Kluwer Academic Publishers, 2000.

[Udupa 99] Divakara K. Udupa "TMN: Telecommunications Management Network", McGraw-Hill Professional Publishing, January 1999.

[U2 Partners] Revised submission to OMG RFPs ad/00-09-01 and ad/00-09-02:Unified Modeling Language 2.0 Proposal. Version 0.671 (draft). available at http://www.u2-partners.org/artifacts.htm, 2002.

[Wang et al 01] Nanbor Wang, Douglas C. Schmidt, Kirthika Parameswaran, and Michael Kircher, "Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications," IEEE Distributed Systems Online special issue on Reflective Middleware, 2001.

[WhittleSchumann00] J. Whittle and J. Schumann. Generating Statechart Designs From Scenarios. In International Conference on Software Engineering (ICSE 2000), 2000.

[XiongLee2000] Y. Xiong and E. A. Lee. An Extensible Type System for Component-Based Design. In the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, vol. 1785 of Lecture Notes in Computer Science. Sringer, april 2000.

[Yuan et al 2003] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets. "Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems". In MMCN-03.

[ZinkyBakkenSchantz97] J. A. Zinky, D. E. Bakken and R. Schantz, "Architectural Support for Quality of Service for CORBA Objects," Theory and Practice of Objects Systems, vol. 3, no.1, pp.1-20, 1997.