Efficient Power Management in Real-Time Embedded Systems

Ana Luiza de A.P. Zuquim*, Luiz Filipe M. Vieira*, Marcos A. Vieira*, Alex B. Vieira*,
Hervaldo S. Carvalho*, José A. Nacif*, Claudionor N. Coelho Jr.*,
Diógenes C. da Silva Jr.†, Antonio O. Fernandes*, Antonio A.F. Loureiro*
*Department of Computer Science, Federal University of Minas Gerais, Brazil
Email: {ana,lfvieira,mmvieira,borges,hervaldo,jnacif,coelho, otavio,loureiro}@dcc.ufmg.br
†Department of Electrical Engineering, Federal University of Minas Gerais, Brazil
Email: diogenes@cpdee.ufmg.br

Abstract— **Power consumption became a crucial problem in the development of mobile devices, especially those that are communication intensive. In these devices, it is imperative to reduce the power consumption devoted to maintaining a communication link during data transmission/reception. The application of dynamic power management methodologies has contributed to the reduction of power consumption in general purpose computer systems. However, to further reduce power consumption in communication intensive real-time embedded devices we have to consider the state of the computation and external events in addition to power management policies. In this paper we propose a model of an Extended Power State Machine (EPSM), where we adapt a Power State Machine to include the state of an embedded program in the power state machine formulation. This EPSM model is used to adapt the Quality of Service (QoS) in communication intensive devices to ensure low power consumption. In such development, a middleware layer fits in the system's architecture, being responsible for intercepting the data communication and implementing the EPSM. Also, a software tool was developed, allowing the Middleware Code to be generated based on the State Machine. A case study demonstrates the application of the proposed model to a real situation.**

I. Introduction

Due to the evolution in the component miniaturization process and the development of high-speed wireless network technologies, there was an increase in use of mobile devices in the last few years. The development of high-speed wireless network technologies enabled an increased demand for mobility and changed the focus of computing systems. These systems are now communication intensive, which means communication tasks are the system's main goal and they must be executed under certain constraints. Sensor networks and wearable computer applications are examples of this shift in paradigm. The increasing use of mobile devices lead to the development of several applications and to an increase in these systems functionalities. However, the power consumption problem was accentuated once these devices are battery operated and we know that batteries are not inexhaustive. Treating the power consumption problem on PCs is quite different, once we suppose we have a continuous power supply. On laptops, otherwise, we do not have a continuous power supply, but their size allow the use of bigger batteries, providing consequently more autonomy. The power consumption problem arises since it is desirable to have mobile devices as small as possible within longer periods of time, and battery's capacity grow with battery's size.

In this work we will consider, more specifically, real-time embedded systems used in a communication intensive environment, which have some peculiarities that may be considered when we are developing new applications. For real-time constraints we mean that a real-time system must satisfy explicit (bounded) response-time constraints or its correctness may be compromised, risking severe consequences, including failure [1]. In these systems, the response time is as important as the correctness of the outputs. A real-time system does not have necessarily to be fast; it must simply produce correct responses within a definite time limit. A real-time embedded system usually monitors on the environment where the embedded system is installed, and if it does not respond in time to a request, the result can be disastrous. Examples of real-time embedded systems are aircraft engine control systems, nuclear monitoring systems and medical monitoring equipment. The need for real-time responsiveness associated with the communication intensiveness introduce extra constraints to a power management policy.

The embedded systems design considers the systems characteristics and restrictions that are fundamental for an efficient system function. As a result, low power design of communication intensive real-time embedded systems must consider the environment and application constraints to optimize the system's design [2], such as real-time responsiveness and intensive execution of communication tasks.

The interaction between the system and the environment may be represented by external events, which must be considered when reducing the power consumption. It is also important to consider the state of computation when the system turns on/off components to reduce power. The state of the computation in each period of time represents the state of the application and its restrictions in an instant of time, which can have a direct influence on the decisions made by a power manager.

Although energy minimization of embedded and mobile computing is of great importance, energy consumption must be carefully balanced against the need for real-time responsiveness. And more important than the power, delay, or even energy is the relation between energy and delay (E x d), once it represents the energy that is spent executing a task.

A power manager task can act in a system by turning off components or adjusting the QoS parameters according to the state of the computation, without harming its timeliness. As an example we can mention an application for remote cardiac monitoring system. In this case it is very important not to interrupt the monitoring function when an abnormal condition is detected, even if the battery power level is critically low.

In this work we proposed a dynamic power management model for a communication intensive real-time embedded system based on systems' quality of service (QoS). This model considers the application's constraints and the environment for the implementation of a dynamic power manager. QoS is defined as a function of the battery power level and the need for system responsiveness (here captured by the computation state) at an instant of time. We propose a model of an Extended Power State Machine (EPSM), where we adapt a Power State Machine to include the state of an embedded program in the power state machine formulation. This EPSM model is used to adapt the Quality of Service (QoS) in communication intensive devices to ensure low power consumption. The model implementation was done through the development of a middleware layer that incorporates a power state machine and executes as a thread independent of the application. The middleware layer hides the difficulties deriving from the combination of diverse applications implementing the same power management model, once it may control the hardware accesses from all applications. We have also developed a graphical tool where the Extended Power State Machine is defined and the code for the middleware layer is generated. A real application where the model was applied is also presented in this paper.

This paper is organized as follows. In Section II we talk briefly about our motivating example, a wearable computer based physiological signal monitoring, which is presented in detail as a case study in Section IV. In Section III we first define some basic concepts and present a new model proposition based on Power State Machines, and show the architecture of an implementation through a middleware layer supporting the EPSM, presenting also a tool created to generate the middleware code for both client and server sides of the application. In Section V we present a case study of power consumption reduction in a wearable computer. Finally we present our concluding remarks in Section VI.

II. Motivating Example

The Wearable Project has been developed at the Computer Engineer Laboratory at the Computer Science Department of Universidade Federal de Minas Gerais and it is our motivating example. The prototype consists of a wearable computer working as a remote physiological signal monitoring system connected to a personal computer (PC) through a wireless network link.

A wearable computer is a low size mobile computer that is subsumed into the personal space of the user, controlled by the user and always on and accessible [17]–[21]. Wearable computers are proactive, which means they send information to its user/another system even when not requested. They usually do not have displays or keyboards and communicate with other computers through a wireless network interface.

We may define a wearable computer as a communication intensive embedded system, responsible, most of times, for the execution of real-time tasks. Wearable computers have been applied in diverse areas of medicine like artificial organs, monitoring, simulations, and as accessory sensors. Wearable computers have been proposed a long time ago [18]; however, its applicability has faced problems related to the dimensions of the computer and its components, and weight, especially when we consider battery dimensions [13]. Applications involving wearable computers are limited by problems related to power consumption, such as battery size, weight and lifetime. Many authors have addressed this problem from a hardware point of view, trying to minimize power consumption of individual components. We can also find in the literature some works related to battery technologies, which try to reduce the battery size and weight while increasing its capacity. The use of batteries as a source of energy in wearable computers should be compatible with human usability and mobility; and it should avoid health problems, such as problems related to battery weight or heat. These improvements are important since we have a trade-off between processing and battery lifetime.

In the Wearable Project, more specifically, we faced some problems related to a high power consumption implying the use of big and heavy batteries. These batteries had, sometimes, the same size of the whole system. Special clothes for carrying the monitoring system were developed and the necessity of power consumption optimization was clear. In this sense, some aspects should be observed, such as the real-time constraints imposed by the application and an intensive communication between the wearable computer and the fixed unit (PC). We noticed also that the transmission rate and the maximum transmission time available for a transmission in each instant of time vary according to the physiological signal that is being transmitted; and all this information can be used to optimize the power consumption.

III. A new approach to the real-time embedded systems power management problem

Reducing the power consumption of mobile systems is one of the greatest challenges for researchers. The increasing use of mobile computers accentuated problems related to power consumption, such as battery size, weight and lifetime. And, as shown in the previous sections, several researches have been developed on low power hardware and software, and power management policies were also proposed.

However, when we are dealing with communication intensive real-time embedded systems, the real-time constraints are not considered in the propositions found in the literature, although they cannot be neglected. When we add low power constraints to real-time embedded systems, it is necessary to consider the state of the computation when the system turns on/off components to reduce power. The state of the computation represents the application state in an instant of time, which can be mapped as the quality of the service provided by the application in each instant of time. Even more, the state of the computation in each period of time may influence the decisions made by the power manager, once the decision to turn on/off a component must be taken according to the application needs.

It is necessary to have the application and the power manager in tune in order to optimize the whole system. Although energy minimization for embedded and mobile computing is of great importance, energy consumption must be carefully balanced against the need for real-time responsiveness.

As an example we can mention our application for remote cardiac monitoring. In this case it is very important not to interrupt the monitoring function when an abnormal condition is detected, even if the battery power level is critically low.

None of the approaches described previously addresses the problem of adjusting the QoS for real-time embedded system according to the state of the computation, once it should be considered in such systems. In real-time embedded systems, it may be more important, for example, to maintain a consistent communication link (even if for a shorter period of time) than reducing the power consumption to save energy. A power manager task can act in a system by turning off components or adjusting the QoS parameters according to the state of the computation, without harming its timeliness.

This is the main idea of the concept developed in this work, and will be fully explained in the following.

Communication intensive real-time embedded systems are systems that depend on an efficient communication link for the accomplishment of the system's main tasks, which must be done under real-time constraints. Although energy minimization for embedded and mobile computing is of great importance, energy consumption must be carefully balanced against the need for real-time responsiveness. In this scenario, one can notice that the indiscriminate use of the power management policies already proposed can take to undesirable results.

This work considers a power management policy that turns on/off components of an embedded system in order to reduce the power consumption. The decision of when components should be turned on and off is taken first according to the state of the computation, and then according to the battery charge state. When we are dealing with real-time embedded systems, the state of the computation in each period of time can have a direct influence on the decisions made by the power manager.

A. Power State Machine

According to the work presented in [4], a *Power State Machine* (PSM) is the representation of a power manageable component (PMC) by a finite state machine. We can model a system that implements dynamic power management as a set of manageable interactive components controlled by a power manager (PM). A manageable component can be described as an indivisible block of the system and the granularity of this definition is arbitrary, i.e., it can be as simple as a micro-controller functional unit or as complex as a circuit board. A fundamental characteristic of a manageable component is the availability of multiple modes of operation associated with different performance and power consumption.

Each one of the PSM states is related to a manageable component mode of operation, representing an instance that spans the power-performance trade-off. It means that we can reduce the power consumed by a component by going to a low power operating mode (low power state) whenever possible, what affects the system's performance. State transitions have a power and a delay cost and they are triggered according to a pre-defined power management policy. In general, low-power states have lower performance and larger transition latency when compared to higher power states. By this way, the process of shifting between modes must be coordinated, once the power saved in the low power state must compensate the power and time spent in the transition. This abstract model can be applied to processors and memories, as well as for devices such as disks drives, wireless network interfaces, display and others.

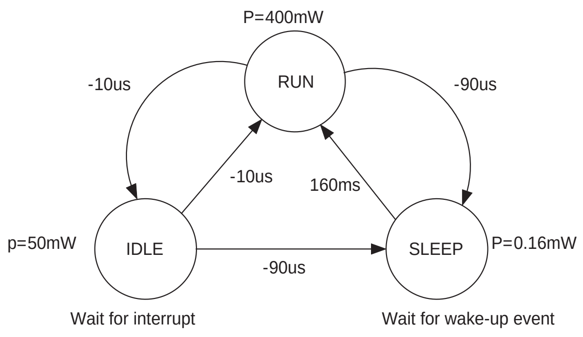

Fig. 1. Power State Machine for the StrongARM AS-1100 processor

Taken from [4], Figure 1 is the PSM model of the StrongARM AS-1100. Each state in the machine is marked with power dissipation and performance values, and edges are marked with transition times. The power consumed in the transitions is approximately equal to that in RUN mode. Notice that both IDLE and SLEEP states have null performance, but the time spent to exit the SLEEP state is much higher than to exit the IDLE state. However, the power consumed by the chip in SLEEP state is much smaller than in IDLE.

Considering the power machine shown above, one can notice that choosing a management policy is an important task, since it has to foresee the period of time that will be spent at the SLEEP state and the power that will be saved in each situation, which must compensate the time spent and power consumed in the transition between the RUN and SLEEP states. The decision to enter a low power state must be well balanced, and can help reducing expressively the power consumed by the device.

B. Extended Power State Machine concept

Although it is correct to map a power management implementation through a power state machine (PSM) in general purpose devices, such as notebooks and palmtops, this PSM model is inappropriate or incomplete when it is necessary to consider the state of the computation in each period of time. In real-time systems, for example, the process of choosing the power management policy to be applied by a power manager can be useless if it disturbs the timeliness imposed by the real-time application. In these types of systems, the power management policy used must consider the computation state, which is defined through changes in the system's internal state caused by external events. The decision of turning a component on or off must consider how important the task that is being executed is. So, the representation using a PSM is incomplete when applied to real-time systems, once transitions between states may consider not only on the battery level, but also on the computation state and external events.

Thus, it is necessary to extend the definition of a PSM, which we will denominate Extended Power State Machine (EPSM), which considers the state of the computation in its power save decisions. An EPSM differs from a PSM in the following aspects:

1) We call events changes in the state of the computation in a period of time or in battery capacity;
2) The previous knowledge of the states of the computation that cause a transition, in conjunction with the power consumption in specific situations determines the states and transitions of the state machine;
3) We will be considering communication intensive devices, mainly those where information exchange with environment is performed with real-time constraints. As a consequence, we will define, for each system state, a quality of service based on the information exchange.

Definition 1 (EPSM): Let M = (Q, E, q_o, δ, QoS) be an Extended Power State Machine, where:
Q is the set of states associated with a previously defined QoS level that is required for a given system condition, i.e. the system's computation state; **E** is the alphabet of events. It can be battery events or even external events that change the states of the computation; $\mathbf{q}_o$ is the starting state; and δ is the set of transitions defined as $\delta \subseteq Q \times 2^{2^E} \rightarrow Q$, where the expression 2^{2^E} represents logic expressions on power and system's events.

QoS is the degree of quality associated to the volume of information that is being transmitted or received for each variable that is communicated by the device.

The quality of service is mapped to the quantity of transmitted/received information, defined as a function of the time available for the transmission and the transmission rate. This mapping can be represented by: QoS $\rightarrow (\Re \times \Re)^m$, where m variables should be transmitted at an adaptive rate ($\Re$) inside the maximum period of time ($\Re$).

Thus, it is possible to turn off components when the transmission ends before the end of the period of time, or to distribute the transmission during the available period of time even though it is made in a lower rate.

It is important to note that changes in the states of the computation depends on the application and, as a consequence, the events generated are defined according to it. We must point out that applying power management techniques to hard real-time systems may be, in some cases, impracticable, once it may introduce an unacceptable delay or condition for the system.

A graphical representation for an example of the EPSM proposed above is shown in Figure 2 that defines states Q_1, Q_2, Q_3 and Q_4, which represent common situations that map states of the system and their respective QoS defined according to the communication needs. For example we may have

Q = { (Q_1 = No activity,QoS_1), (Q_2 = Low activity,QoS_2), (Q_3 = Medium activity,QoS_3), (Q_4 = High activity,QoS_4)}

QoS = {QoS_1= no transmission, QoS_2 = partial transmission of data (few data), QoS_3 = partial transmission of data (much data), QoS_4 = full transmission}

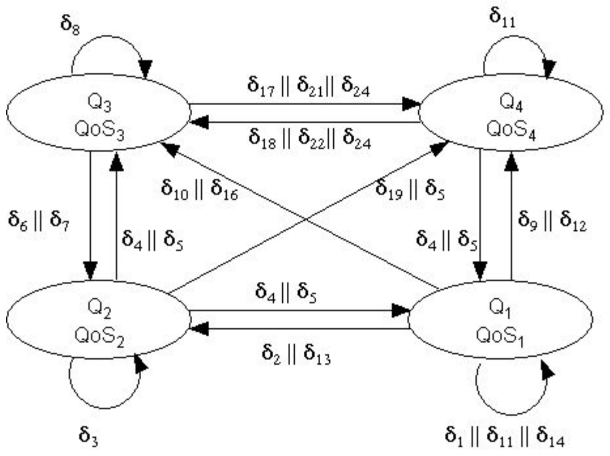

Fig. 2. Example of an EPSM of a military equipment

For each state the system will have a different configuration, and changes in the system variables will influence each state differently, i.e., the transitions from one state to the other are triggered by different events, E_n, that lead to different states. Transitions are represented in our example by δ_n.

δ $=\{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, ...\}$ where

δ_1 $=Q_1 \times (E_1 \ \&\& \ E_5) \rightarrow Q_1$

δ_2 $=Q_1 \times (E_2 \ \&\& \ E_5) \rightarrow Q_2$ and so on.

Let us contrast the EPSM presented above with the PSM of Figure 1. We can notice that in a PSM, the states represent an operation mode of the system, which is associated to system's hardware configurations. In an EPSM, however, we have QoS levels mapped directly by states, where different hardware configurations are loaded according to the task that is being executed. While transitions in a PSM are triggered by a power management policy that is usually pre-defined, in an EPSM, the decision of when changing from a state to another is a combination of the battery power level and the system computation state, which are both dynamic characteristics of the system.

One scenario where this model applies consists of a portable equipment worn by a soldier in a war, receiving data from various sensors placed in the war environment and sending this data to a control station. The application remains collecting data from the sensors from time to time, and shutting down idle subsystems such as the transmitter, can save a significant amount of power in the system. However, we must remember that the computation needs can also vary according to external events, as for example, it is necessary to collect more data in more dangerous situations, or even stop receiving signals when the soldier is at the encampment. Thus, we define each state by a soldier condition, which implies in less or more data being collected from the environment. A soldier in combat, for example, implies on collecting data with the best quality possible. The detection of a bomb, for example, is a critical task and should be done even though the battery level is low. Battery events are also considered in the decision of which data should be collected. For this example we would have:

- $E = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$ where

E_1 = resting	E_4 = in combat	E_7 = 25% battery left
E_2 = walking	E_5 = 75% battery left	E_8 = 5% battery left
E_3 = running	E_6 = 50% battery left	

- $QoS = \{QoS_1, QoS_2, QoS_3, QoS_4\}$ where

QoS_1 = {1 data collection/0.5 h, moving sensors turned on}
QoS_2 = {1 data collection/7 min, 1/3 of the sensors turned on}
QoS_3 = {1 data collection/minute, 2/3 of the sensors turned on}
QoS_4 = {1 data collection/5 secs, all sensors turned on}

In the motivating example presented in Section 3, we may have the wearable computer collecting physiologic signals such as the electrocardiogram, blood pressure and blood saturation of oxygen in a patient during his daily activities. In this sense, the wearable computer works as a communication intensive real-time system, where the transmission rate and the maximum transmission time available for a transmission in each instance of time vary according to the physiological signal that is being transmitted. For each signal, it is possible to transmit an amount of data in the beginning of the period available for transmission and turn the communication interface off when data is not being transmitted. This optimization allows a reduction in the power consumption and can also be implemented by an EPSM.

IV. Model implementation through a middleware layer

Considering a communication intensive application that consists of a server and a client exchanging data through a wireless link, a middleware layer was developed implementing a power management policy while controlling the applications access to hardware and software. The middleware intercepts the data exchanged between the application and the communication interface and adapts it according to a power management policy. The middleware layer is incorporated in both client and server, once the data transmission is processed according to the power management policy. This approach is interesting since the presence of a power manager in the system is transparent to the user, although it demands an adaptation from the application side.

The development of a middleware layer simplifies the application's implementation, since it does not have to take care of the power management tasks nor worry about other applications that could be accessing a hardware that it is trying to turn off. Figure 3 shows how a middleware layer fits in the system architecture. The dashed line represents the use of a wireless communication interface in the communication process between the mobile unit and a fixed unit. The use of a mobile unit and a fixed unit is just an example and might be extended to other combinations of mobile and fixed units. Our choice for a mobile and a fixed unit was based on our motivation example and it expresses clearly the power management problem present in mobile units of all kinds.

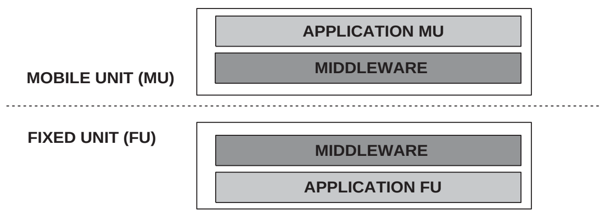

Fig. 3. The insertion of a middleware layer in the system's architecture

As just explained, the middleware layer is responsible for intercepting the data communication and implementing the Extended Power State Machine. Although all of the control of the state machine remains in the middleware, the application is responsible for defining the EPSM structure to the middleware according to the tasks executed at each application condition or state, including the states and transitions.

A. System Architecture

The application defines the states of the state machine and their respective QoS attributes and this state machine is the core of the middleware. The application defines also the transitions, indicating, for each transition, a condition test function and the next state it takes in the state machine if this condition is satisfied. Thus, the middleware is responsible for controlling the transitions between states and sending/receiving data to/from the communication interface according to some state QoS.

Some concepts were defined for a better understanding of the system's architecture and they are just presented:

- **Data buffer**: a data buffer is a shared data structure used for the data exchange process. The application writes data to data buffer while the middleware reads data from the data buffer. This read/write access is controlled through the use of mutexes.
- **Data block**: a data block consists of a set of data grouped according to its transmission rate and maximum transmission time characteristics, on which will be applied the same procedures for data transmission and storage. For each data block we have one data buffer associated, and all the data for that data block is transmitted using this

data buffer. Also associated with a data block we have data map functions, which are application-defined functions that provide an adjusting mechanism for the transmitting rates. For example, an application may be filling the data buffer in a higher rate than the middleware is capable to send data to the communication interface. In this case, an adjustment is necessary, such as a medium, an intermediary value, or other, according to the application needs. The data map function is responsible for applying this adjustment. These functions could also do some kind of backup, like saving not transmitted data in a file for posterior analysis.

• **State**: a state of the state machine can be defined as a group of data blocks with different transmission times and transmission rates that indicate how the data in their respective data buffers must be sent to the communication interface. We have also associated to a state a list of transitions to be tested and executed.

• **Transition**: a transition consists of a pointer to the next state and a transition test function, which is defined by the application as a condition test function (callback routine).

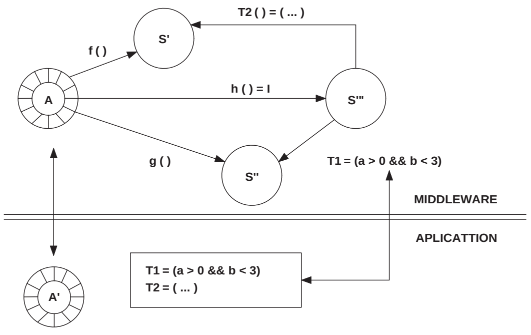

Fig. 4. Schematic of the interaction between an application and the middleware

A schematic of the interaction between the application and the middleware - that is present in both client and server sides – and the application is shown in Figure 4 and can be described as follows:

1) The application requests the creation of data buffers for shared access. These buffers will be the place where the application will write the data instead of sending it directly to a communication interface. They are represented in Figure 4 by **A**. Here, it is important to highlight the fact that it is the middleware who defines the data structure, of course, according to the application's specification (buffer size, etc).
2) The application creates a state informing its transmission rate and maximum transmission time. S', S'' and S''' represent the states in Figure 4. Recall that the QoS is mapped to the quantity of transmitted/received information, defined as a function of these two parameters applied to m variables. For each state we can have different blocks of data, each one with a distinct QoS requirement and associated with a different data buffer.
3) The application may inform data mapping functions for each data block, which will be responsible for adjusting the data read from the data buffer to the data sent to the communication interface, providing always consistent data. These functions are represented in Figure 4 by **f ( )**, **g ( )** and **h ( ).** As an example we can mention the situation where an application fills the data buffer in a rate that is different from the transfer rate. In this case, a data map function could be a function that reads X entries of the buffer and returns the medium value of these entries. The decision about what will be transmitted or what to do with data that is not transmitted is taken by the data map functions. In Figure 4 we have h ( ) = I, which represents the standard situation, where all data put in the buffer by the application will be sent to the communication interface.
4) Finally the application may define the transitions between states, indicating the next state to transition to if a condition test function is satisfied. The result of a condition test function is determined dynamically by the application through the execution of a callback routine, which will inform the middleware about its occurrence.

The transition function was implemented as a callback function once it is defined and executed by the application, which identifies the occurrence of events that trigger a transition and notifies the middleware about the occurrence of an event. Once the state machine is defined, the application and the middleware start exchanging data through the data buffer, and the middleware also reads/writes data from/to the communication interface.

B. System Modeling and Implementation

In the implementation of the whole system, which comprehends the middleware and the application explained previously, some important concepts such as multi-threading and object orientation were applied, allowing the development of a modular middleware.

The multi-threading concept allows multiple programs to be loaded into memory and to be executed concurrently, and this requires a firmer control on them. A thread shares with peer threads its code section, data section and operating system resources, but like any parallel processing environment, multithreading a process may introduce concurrency control problems that require the use of critical sections or locks.

The necessity of having a shared data buffer for the middleware and the application, with different program counters, register sets and stack spaces, implied in the use of multi-threading. We may highlight that we could use other approaches, like pipes or files, which would be inefficient in our case. So, we defined two threads accessing the same address space, and using semaphores to deal with the concurrent access in the data buffer. When the application thread is accessing the data buffer, it blocks it, preventing a middleware read/write access. Meanwhile, the middleware thread can process other tasks if it does not need to access the data buffer at exactly the same time. We defined also semaphores indicating if the buffer is full or empty, which are shared between the two threads. Thus, the implementation of the application and the middleware with multi-threads was the best solution for the multiple accesses in the data buffer, implying in the implementation of a lock policy to control this access.

The object-oriented concept was also adopted in the development of the middleware layer. The development process of real-time and embedded systems [21] may be guided by a development methodology as happens with the development of common software. In this sense, we explored the advantages provided by an object-oriented methodology, such as reuse and scalability. The UML (Unified Modeling Language) is particularly suited for real-time embedded systems project and development and its use provided a clearer vision of the whole system.

Representing the system's functionalities through a use case diagram allowed a better definition of the system's boundary and the interaction process between the application and the middleware, simplifying the specification and implementation of the whole system. From the definition of use cases, we produced the class diagram, which represents the structure of the system in terms of classes and objects, including how the classes and objects relate with each other. With this facility in hands it was possible to develop a code generator for the middleware layer, which creates the skeleton of code for the middleware from the state machine specification.

C. *Software Developed – Middleware Code Generator*

Once the middleware layer was developed using object-orientation concepts, the use of a code generator is easier and brings a large number of advantages such as an increase in the productivity in the development of applications that implement the power state machine proposed.

The middleware code implements the EPSM defined by the application, and a tool to generate this middleware code is useful and helps increasing the efficiency in development of such applications.

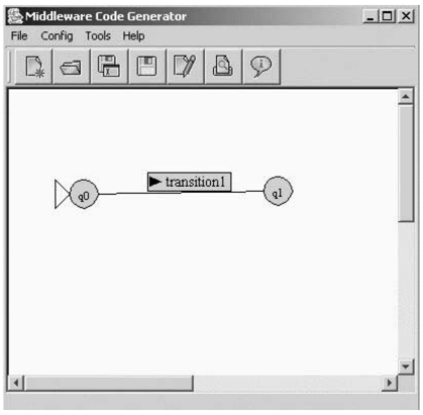

Fig. 5. Middleware Code Generator Tool

The developed tool allows the drawing of a State Machine, including the definition of states and transitions, and generates the code for both the client and the server. As highlighted before, the state machine must be implemented in both client and server sides, once the data is processed in one side (by a data map function) and must be understood by the other side. It is possible to define state names and transition times and conditions, representing the transition functions defined by an EPSM. The tool allows the configuration of every parameter needed in the code for the client and server.

The tool was developed with Java, maintaining its portability and oriented object paradigm, and a user-friendly graphical interface for a quick development. The code generated by the tool is in C++ and implements the same state machine for the client and server sides of the middleware. Figure 5 presents the tool interface, where an example state machine composed of two states q_0 and q_1 and a transition between these two states has being drawn.

The generated code allows a transparent implementation of a power manager for the application, and a high reuse of the middleware code.

D. *Tests and Results*

The model idea was first evaluated through the development of simple applications that were responsible for turning on and off the wireless communication interface of a wearable computer. For each operation we took the time and the power spent in the process. We also measure the power spent transmitting different volumes of data, and we could see that turning the interface off when idle and turning it back on after a pre-determined period of time would save energy (Figure 6).

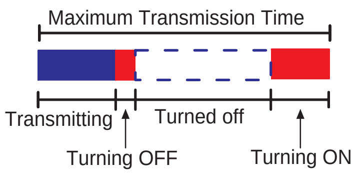

Fig. 6. Turning a wireless interface OFF to save energy

A pre-defined slot of time was used once we were dealing with small packets of data, and the transmitting process of small and large volumes of data (up to the maximum bandwidth) spend almost the same amount of power. This happens since the main power cost is concentrated in turning the interface ON, operation that is necessary for both small and large amounts of data. By this way, for a transfer rate of 600bytes per second, for example, we determined that we would send 6000bytes in 10 seconds and, as the transmission was concentrated in the first microseconds, we would turn the interface off in the rest of the period. For each state, the power saved is given as a function of the time the interface card stays turned off.

A second step consisted in the implementation of the middleware layer, which would be responsible for controlling the EPSM autonomously. The middleware layer implements the capabilities to turn the communication interface ON and OFF and controls also the data that is sent through it. With the middleware layer we could test effectively the concepts and ideas defined previously. Next, a case study is presented as an example of an implementation of the concepts explained in previous sections to a real application.

V. Case Study: Power Consumption Reduction in a Wearable Computer

The Wearable project has been created at the Computer Engineer Laboratory at the Computer Science Department of Federal University of Minas Gerais. The prototype consists of a remote physiological signal monitoring system operating on a wireless network. Wearable computers have been applied in diverse areas of medicine like artificial organs, monitoring, simulations and as accessory sensors. One of the greatest limitations of its application is the size of components and weight, especially when we consider battery dimensions [13].

The system is a battery powered mobile device with network communication capability. The equipment is composed by two hardware modules. The first one is an acquisition system that receives analog data from physiological sensors. The second module is a 486 DX 100 MHz microprocessor, running embedded Linux and a wireless Ethernet device based on 802.11b. This second module is shown in Figure 7. This processing module receives data from the sensors, classifies it and transmits the data to a central station that can be any computer connected to the network [19].

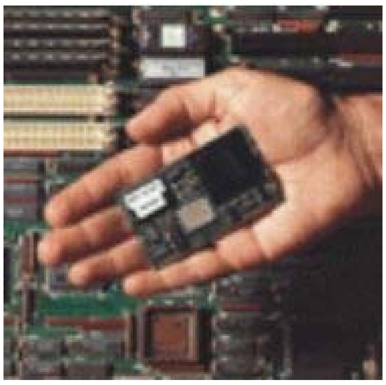
Fig. 7. Data processing module

Acquiring vital signals from a patient in movement is an operation that requires peak performance only during some periods of time. In this way, the system's components do not need to be active all the time. The capacity of turning on and off components or even adapting the performance to the user's requests is important functionalities for mobile applications, once we are tied to power constraints.

A critical factor that influences the system's performance is the power consumption. The CPU and the wireless network interface card are the great power consumers. Because of this, the application above is an interesting example where we can apply the dynamic power management concept based on EPSM formulation. It is possible to perform a dynamic reconfiguration of the system in order to provide the required service with the previously defined QoS. This adaptation of QoS is based on changes in the battery charge and in the state of the computation during a period of time. The implementation of the proposed model requires a previous evaluation of the system functionalities to determine the power states and the corresponding changes in QoS.

Table I shows an example of a set of sensors for the remote physiological signal monitoring system, along with the QoS acceptable limits. The wearable project implements a set of these functionalities: electrocardiogram (EKG) with up to three channels, heart rate, pulse oximeter and non-invasive blood pressure.

We defined the following order to turn the sensors off, although it can be changed in specific situations: 1. Body position; 2. Phonocardiogram; 3. Electromyogram; 4. Bio-chemistry markers; 5. Temperature; 6. Gas CO_2; 7. Gas O_2; 8. Respiratory Volume; 9. Respiratory rate; 10. Blood Pressure; 11. Pulse Oximeter; 12. Electrocardiogram; 13. Blood Flux. According to the clinical importance of the monitored signals, the following power saving procedure is used:

1) Progressively reduction of redundant sensors;
2) Progressively reduction of signals sampling rate until the minimum acceptable limit;
3) Progressively turn the sensors off. The power consumption saving procedure will decrease not only the processing work, but also the volume of data transmitted.

Note, however, that the specific power saving conditions are related to the computation state, which in this case is the patient condition evaluation. This evaluation must be performed either by an external event (the patient signals the system that he/she is not feeling well) or by carefully examining the patient data (either remotely or by the wearable system itself). Depending on the patient's condition, we can also decrease the sampling rate and number of sensors if the patient is in a healthy situation, and if there are no abnormal signals. In situations where an event occur or any abnormal signal is detected, the monitoring task should be performed with the maximum quality available necessary for the correct event identification by a physician, independently of the battery charge at that time.

Situations like these show how important it is to include the state of the computation in the model of dynamic power management for real-time embedded systems.

In the system described here, for an EKG monitoring device, we can identify three different monitoring states: (A) Minimum patient monitoring; (B) Partial patient monitoring; and (C) Maximum patient monitoring with the best possible signal quality.

In Situation (A), the EKG sensors will capture the data using one channel and a sampling rate of 300Hz. According to a local evaluation of the condition, the EKG is transmitted with a lag of 30 seconds, with power consumption being traded against system responsiveness. Situation (B) requires a more responsiveness for data capture process. It uses three channels and a sampling rate of 300Hz. More channels are necessary for better characterizing an abnormal EKG wave. In this situation, a specialist is necessary most of the time to analyze the waves, and as a result, transmission lag time is reduced to 5 seconds (internally). Situation (C) is a high risk situation where the

TABLE I

SIGNALS CHARACTERISTICS, DATA FLUX AND TRANSMISSION POWER CONSUMPTION

Signal	Sample size (bits)	Frequency (Hz)	Frequency variation acceptable (Hz)	Number of Sensors	Data Flux (bits/min) based on normal sensors and frequency bounds	Transmission Power Consumption based on Data Flux (J/sec)
EKG Static	16	1000	300 to 1000	2 to 16	7,680,000 bits/6sec	10,437
EKG Dynamic	16	1000	300 to 1,000	2 to 10	9,600,000	13,046
BP Non-invasive	8	0.01	≤ 1	1	4,8	0.000006
BP Invasive	16	40	40 to 100	≥ 1	38,400	0.05
Pulse Oximetry	4,800	1	≥ 1	1	288,000	0.39
Blood Flux	16	40	≥ 40	≥ 1	38,400	0.05
Respiratory Rate	16	100	≥ 100	1 to 3	96,000	0.13
Respiratory Volume	16	100	≥ 100	1 to 3	96,000	0.13
Body Position	8	1	≤ 0.1	≥ 6	2,880	0.004
Gas O_2	8	0.1	≤ 0.1	≥ 1	48	0.00005
Gas CO_2	8	0.1	≤ 0.1	≥ 1	48	0.00005
Biochem Markers	8	0.1	≤ 0.1	≥ 1	48	0.00005
Phono-cardiogram	16	1000	500 to 1,000	≥ 1	960,000	1.3
Electro-myogram	16	500	300 to 500	≥ 3	1,440,000	1.957
Temperature	8	0.01	≤ 0.01	≥ 1	4.8	0.000006

patient's clinical situation needs almost all the signals with the best quality at real-time. In this case, it will be necessary to use more channels (12) and a sampling rate of 1000Hz (high resolution EKG). In this case will transmit the data at every second.

Note that when we discuss real-time responsiveness in this example, we are considering only the device's communication delays. In reality, Internet based devices suffer from all kinds of network traffic delays that are not considered in this paper. These delays are one of the reasons why the system requires enough intelligence to make local decisions when no response from a medical monitoring team is available in reasonable time.

The triggering events that will transition the system into its different monitoring states are the following: patient with no previous disease identified; patient with a low-risk disease; patient with a middle risk disease; and patient with a high-risk disease.

These events are based on the initial physician's evaluation of the patient's condition and they are modified according the patient's clinical evolution.

TABLE II

MONITORING QUALITY ACCORDING TO PATIENT'S CLINICAL STATE

Quality of Monitoring	Min (%)	Partial (%)	Max (%)	Comm. Power (mW)
Patient w/o disease (healthy user)	97	2.9	0.1	71.00
Patient w/ a low risk disease	95	4.75	0.25	78.70
Patient w/ a middle risk disease	40	59.5	0.5	248.45
Patient w/ a high risk disease	30	65	5	327.67

Considering each one of these classifications, we can associate the presence of events to the patient's necessity of monitoring. This association can be used to calculate the probability of an event occurrence and the probability of changing the monitoring parameters in a period of time.

In a period without disease, for example, the patient can be monitored 97% of the time using the minimum configuration and 0.1% using the maximum configuration. In a period where the patient is in a high risk situation, it will be necessary to use the minimum configuration in 30% of the time and maximum configuration in 5% of the time. Table II presents the correlation between the amount of time spent with each type of monitoring state, according to the patient's clinical condition and potential risk of disease.

As shown in Table II, it is possible to save power by changing dynamically the monitoring state of a patient. In this way, it is possible to change the QoS for the communication parameters and also turn off the network interface when it is not in use. These operations contribute to reduce the power consumed in communication and processing. The state of the computation indicates when changes in the monitoring configuration can be performed. Changes in the states of the computation are identified by a previous data processing executed over data sampling. The indication of the patient initial risk level is determined previously by a specialist. The patient condition may change and will be defined by a software tool based on an agent that analyses the patient data.

In Figure 8 we have the representation of the EPSM model for the medical device explained before. We can notice that the high power consumption can be avoided if we have a patient in good health conditions, which accounts for most of the time. It is important to emphasize that the values shown in Table II can be fine tuned in order to get better results according to the application that is being executed. The representation through an EPSM lets us identify the important events, based on the state of the computation that will really influence the transitions between states.

Considering patients with different health conditions, there will be different monitoring conditions that will allow lower and higher power savings. For example, considering a patient with no previous diagnosis of a disease, he can be monitored

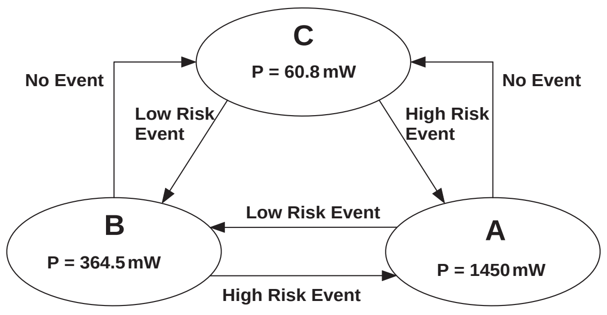

Fig. 8. EPSM for an electrocardiogram monitor

with a minimum data-collecting rate, at 97% of the time, and be partially monitored at 3% of the time. The graph in Figure 9 shows how is the power consumption for this monitoring condition. The peaks of power consumption occur when the communication interface is turned on, and the straight line represents the medium power consumption.

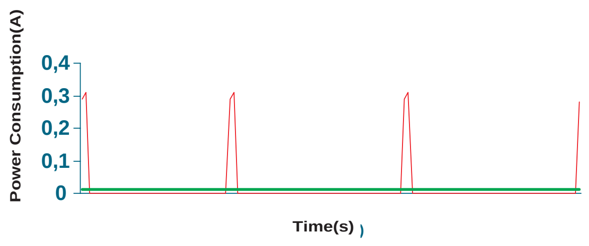

Fig. 9. Power consumption vs. Time for a patient monitoring

A 4AH battery was used for testing and it was verified that when not implementing any power management technique, the battery lifetime was about 8hs, while using the power manager as proposed, the battery lifetime almost duplicates (about 15 hs); in both situations transmitting one channel at 300Hz.

VI. Conclusions and Future Work

In this work we proposed a model to achieve low power consumption in real-time embedded systems based on dynamic power management, where the state of the computation is as important as the battery capacity. The formulation was called an Extended Power State Machine (EPSM), where the states of an EPSM depend on the embedded program's state, and the transitions depended on external events. Power consumption was reduced based on adapting the quality of service for the variables that were communicated between a mobile and a fixed unit (monitoring station). Each state of the system required a different QoS.

A case study was presented for a medical wearable device that considers the patient's state when deciding upon the quality of service and responsiveness for the sensors' data transmitted over a wireless network. We showed that significant power reduction in communication was achieved when this technique was applied. The EPSM was designed as a multi-thread application comprising the medical application and a middleware layer, which implements the EPSM. The solution adopted on designing a middleware layer facilitates the implementation of further applications.

Acknowledgment

This work has been partially supported by CNPq, Brazil, under process 55.2111/2002-3.

References

[1] P. A. Laplante, *Real-Time Systems Design and Analysis*, I. Press, Ed., 1997.
[2] W. Wolf, *Computers as Components - Principles of Embedded Computing Systems Design*. Morgan Kaufman Publishers.
[3] P. Havinga and G. Smit, "Design techniques for low power systems," 2000. [Online]. Available: citeseer.nj.nec.com/havinga00design.html
[4] L. Benini, A. Bogliolo, and G. D. Michelli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 0, no. 3, June 2000.
[5] T. Simunic, L. Benini, P. W. Glynn, and G. D. Micheli, "Dynamic power management for portable systems," in *Mobile Computing and Networking*, 2000, pp. 11–19. [Online]. Available: citeseer.nj.nec.com/simunic00dynamic.html
[6] L. Benini, G. Castelli, A. Maci, and R. Scarsi, "Battery-driven dynamic power management," *IEEE Design and Test of Computers*, vol. 18, no. 2, pp. 53–60, March/Abril 2001.
[7] S. Havinga, "A survey of energy saving techniques for mobile computers," 1997. [Online]. Available: citeseer.nj.nec.com/smit97survey.html
[8] G. F. Welch, "A survey of power management techniques in mobile computing operating systems," *Operating Systems Review*, vol. 29, no. 4, pp. 47–56, 1995. [Online]. Available: citeseer.nj.nec.com/42641.html
[9] J. R. Lorch and A. J. Smith, "Software strategies for portable computer energy management, Tech. Rep. CSD-97-949, 13, 1997. [Online]. Available: citeseer.nj.nec.com/lorch98software.html
[10] K. Li, R. Kumpf, P. Horton, and T. E. Anderson, "A quantitative analysis of disk drive power management in portable computers," in *USENIX Winter*, 1994, pp. 279–291. [Online]. Available: citeseer.nj.nec.com/li94quantitative.html
[11] C. H. Hwang and A. C. Wu, "A predictive system shutdown method for energy saving of event-driven computation," P. I. C. C.-A. Design, Ed. IEEE CS Press, 1997, pp. 28–32.
[12] J. Lorch, "A complete picture of the energy consumption of a portable computer," 1995. [Online]. Available: citeseer.nj.nec.com/lorch95complete.html
[13] M. Bhardwaj, R. Min, and A. Chandrakasan, "Power-aware systems," in *34th Asilomar Conference*, November 2000, invited Paper.
[14] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N. K. Jha, "Power analysis of embedded operating systems," in *ACM*, 2000, pp. 312–315.
[15] V. Swaminathan and K. Chakrabarty, "Real-time task scheduling for energy-aware embedded systems," 2000. [Online]. Available: citeseer.nj.nec.com/375160.html
[16] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *IEEE Annual Foundations of Computer Science*, 1995, pp. 374–382.
[17] R. W. Picard and J. Healey, "Affective wearables," The Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA, MIT Laboratory Perceptual Computing Section Technical Report 467, November 1997.
[18] S. Mann, "Definition of wearable computer," May 1998, university of Toronto.
[19] J. Conway, C. C. Jr., A. F. D.C. da Silva, L. Andrade, and H. Carvalho, "Wearable computer as a multi-parametric monitor for physiological signals," in *BIBE Biomedical Applications*, 2000, pp. 236–242.
[20] T. Staner, S. Mann, B. Rhodes, J. Healey, K. Russel, J. Levine, and A. Pentland, "Wearable computing and augmented reality," The Media Laboratory, Massachusetts Institute of Technology, Cambridge, MA, MIT Media Lab Vision and Modeling Group Technical Report 355, November 1995.
[21] B. P. Douglass, *Real-Time UML Second Edition - Developing Efficient Objects for Embedded Systems*, A. Wesley, Ed., 1999.