ELSEVIER

Microprocessors and Microsystems 26 (2002) 463–474

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.com/locate/micpro

Component-based development of DSP software for mobile communication terminals

Kari Jyrkkä*, Olli Silven, Olli Ali-Yrkkö, Ryan Heidari, Heikki Berg

Nokia Corporation, P.O. Box 50 (Elektroniikatie3), Oulu FIN-90571, Finland

Received 27 March 2002; revised 15 August 2002; accepted 28 August 2002

Abstract

DSP software development has been tied down by extreme computational requirements. Furthermore, the DSP development tools available today are less advanced than in other embedded software design. This has lead to DSP software architectures that have not taken into account future expansion needs. Therefore, DSP software architectures have been inherently closed. Now, as system complexity increases, this design methodology becomes more of a burden, since it does not support component-based DSP software development that requires open interfaces.

In this paper, mobile-communications DSP software architectures are studied as cases, and key areas for improvements towards more open DSP software development are identified. Proposed solutions are judged against the limited resources of mobile communication terminals and the characteristics of communication DSPs.

Keywords: Mobile communications; DSP; Architecture; Component software; Open software

1. Introduction

DSP software architectures built for mobile communications processors are prime examples of closed designs that are intended to carry out specific tasks and nothing more. However, the designers of these systems typically target the creation of common platforms for product families, in other words using essentially the same architecture and software over a range of products. It requires an extreme effort to meet the contradicting requirements of efficiency and versatility using the same architecture with the limited resources offered by mobile processors.

Mobile processors have become very high volume products, where cost and low power have been the main design criterion [11]. The products containing these processors typically feature limited programmability and only a few operational modes. These include different channel coding schemes for various channel bit rates; a high I/O to computations ratio as in GMSK demodulation; and stream data processing such as 'when buffer ready, do computations before next buffer ready'. The role of the DSP software architecture is to match the characteristics of mobile communications DSP software with the underlying limited processor capabilities.

In a typical mobile communications DSP architecture as depicted in Fig. 1, the software consists of a loop where different operational modes can be selected as alternative branches. Each branch is responsible for handling all the input/output channels that are active in the respective mode. Processing through a branch consists of a sequence of function calls to data processing algorithms; an 'input buffer ready' or 'sample ready' interrupt initiates the sequence.

This approach uses the DSP processor resources very efficiently and predictably. Its design methodology is fairly well understood [3], and it results in virtually no task control overhead. On the other hand, the introduction of any new functionality is difficult with this architecture. For example, in mobile phone communication, the voice user interface may be required to operate in every mode of the DSP loop. Such a function needs to be integrated and tested with all branches.

Fig. 2 shows a system level mobile architecture, where the role of the DSP processor is to take care of data-intensive processing from input port to output port, with the assistance of the hardware accelerator on application specific integrated

* Corresponding author. Tel.: +358-10-505-8539; fax: +358-10-505-7222.

E-mail address: kari.jyrkka@nokia.com (K. Jyrkkä).

0141-933/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0141-9331(02)00077-7

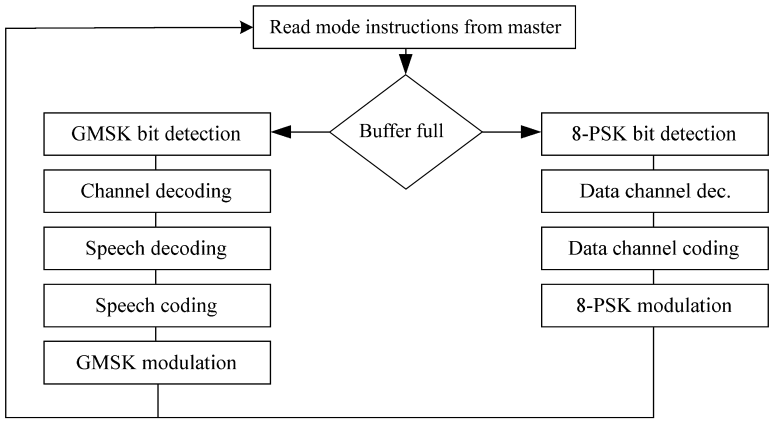

Fig. 1. A loop-type communications DSP software architecture.

circuits (ASICs) [14]. The microcontroller unit (MCU) is in charge of higher-level operations and system control. The hardware/software partitioning of the DSP system is subject to extensive changes between product generations. For instance, in older mobile phones the Viterbi algorithm resided in hardware, advances in mobile processors brought it into software, but increased data communications needs have pushed it back into hardware again. With the loop type DSP software architecture, these kinds of changes affect the testing of every operational mode.

The traditional loop-type DSP software architecture is a burden for development when enforced at system level, and many of the technical limitations that lead to its adoption are no longer considered to be critical. The increased memory spaces of the processors enable the use of low-overhead, real-time operating system kernels, and support higher-level languages. Multimedia applications are coming, and the complexity of DSP system software alone requires more advanced methods than assembly-level coding and manual tuning of memory usage and scheduling. Clearly, mobile communications devices need more open and easier DSP software development framework to cope with increasing complexity, changing hardware and time to market requirements. Ideally, DSP services (even resources) should be made available to third party software developers.

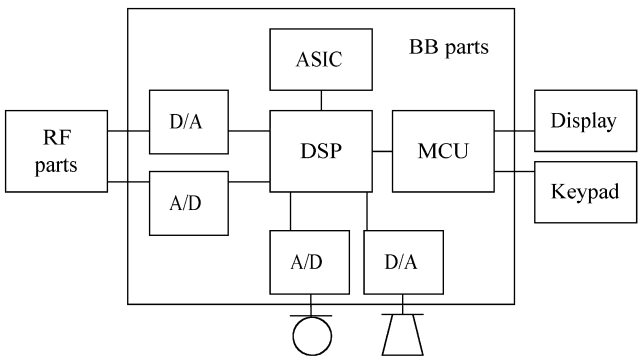

Fig. 2. A typical mobile communications device architecture.

Component-based DSP software development framework is defined here as a platform for third party DSP software component development or, at the minimum, as an environment for internal component-based software design, which provides also an encapsulation and execution environment for legacy software. From a software point of view, this means opening selected application programming interfaces (APIs) and providing the necessary development tools, such as system simulators, test cases, and monitoring software. From a hardware point of view, the key issue with component-based software architecture is the efficient utilization of the mobile-communication-specific resources in data-intensive applications.

The special characteristics and efficiency requirements of DSP software are still valid, despite advances in processor technology. They must consequently be considered under the light of a component-based DSP software definition. Highly efficient and predictable loop-type, periodic DSP task-scheduling must be available, while the architecture must provide mechanisms to enable flexible loop configuration for software developers, and the addition of new functionality even during a runtime operation. Even the operating system utility in the DSP software of mobile communication devices needs to be evaluated from this point of view.

In this paper, mobile communications DSP software is studied as a case for component-based software development. Key areas for improvement towards openness are identified, and some principles are proposed for using and providing DSP services with minimal losses to performance. The paper is structured as follows. Section 2 discusses the open platform concept and mobile communications software development. Section 3 proposes architecture for mobile communication DSP software, and Section 4 analyses its properties with respect to typical mobile multimedia terminal DSP design challenges. Finally, Section 5 contains discussion and Section 6 presents conclusions.

2. Open platform concepts and DSP software

The complexity and functionality of mobile communication devices has increased dramatically during recent years. As a result, there is an endless demand for powerful tools and methodologies to enable the completion of development within the given time frames. In order to expand the number of developers, 'standard interfaces' and 'openness' have become key issues in the mobile communications field. In this chapter, proposed open-platform concepts are analysed from DSP software point-of-view.

2.1. Symbian operating system

Symbian OS is an open operating-system platform and library that contains a huge number of services; for instance, TCP/IP protocol stacks and graphics display support [15]. Symbian OS is clearly targeted at the developers of general-purpose applications. Due to the size and rather long context-switch latencies of its current implementations, it is not suitable for very small memory capacities and purposes that require a real-time response. As such, it is not a candidate for DSP. A small footprint is among the prime requirements for a mobile terminal DSP software platform.

2.2. Binary runtime environment for wireless

'Binary runtime environment for wireless' (BREW) is a technology that allows users to download and run software on mobile phones [1]. Third-party application developers use the BREW software developer kit to develop new value-added services. Middleware servers are provided to enable the authentication of certified applications, the management of end-user download purchases, and so on.

BREW application execution environment (AEE) is an object-oriented application development and execution environment, which sits on top of Qualcomm's mobile station modem ASIC. BREW is expected to enable third party application development for low cost mass-market devices, thus efficiency and RAM consumption have been key design criteria. There are plans to support audio, video, email, and location-based services over BREW-enabled architecture. Many of these clearly require DSP-type computation. However, from DSP software development point of view BREW is a closed environment.

2.3. An open DSP software architecture for audio decoder

An open DSP software architecture proposed for audio decoders [4] includes an idea to allow third party software modules to co-exist as value added post decoder plug-in functions (Fig. 3). This approach allows users to take full advantage of DSP processors signal processing capabilities, but the application repertoire and API are very limited. In essence, the idea is to reserve memory and time-slots from

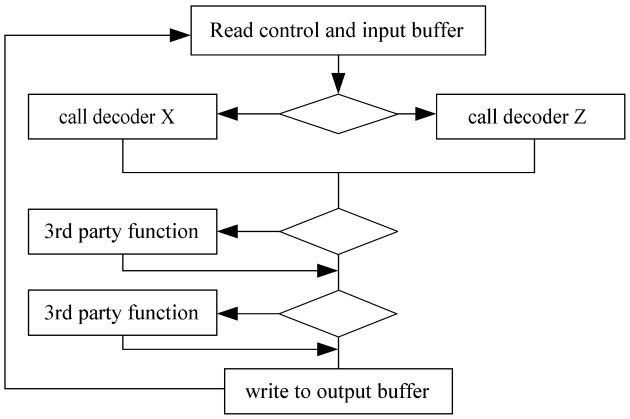

Fig. 3. Open audio decoder DSP software architecture solution.

loop-type DSP software architecture, for use in third party modules.

This approach is an example of providing internal openness for loop-type software architecture. The manufacturer of the platform system must carry out the integration testing of third-party modules. The approach is attractive in its simplicity, but the degree of openness provided is limited, as the API services are limited to data input and output. Nevertheless, it is intended for DSP use and takes into account its real-time requirements by providing a fixed timeslot for third party algorithms.

2.4. Open multimedia application platform

'Open multimedia applications platform' (OMAP) combines the functionality and characteristics of a digital signal processor and MCU to support wireless, multimedia applications [2]. The OMAP hardware architecture is based on a combination of DSP (TMS320C55x) and MCU (ARM925T) cores. Both have a memory management unit (MMU) for virtual-to-physical memory translation and task-to-task memory protection; however, this is limited in DSP to external memory access. In addition, the OMAP hardware module contains a multiport DMA controller, along with interfaces for connecting to external peripherals (Fig. 4).

The OMAP software architecture (DSPBridge) is a combination of software for the MCU and DSP real-time

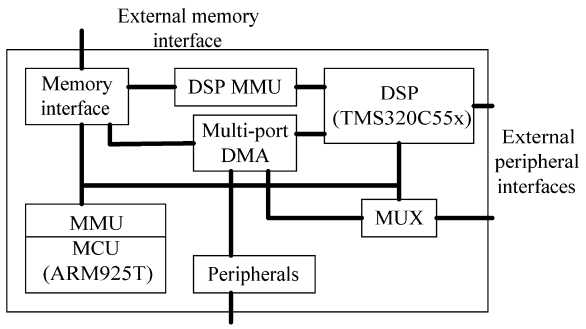

Fig. 4. The OMAP chip architecture.

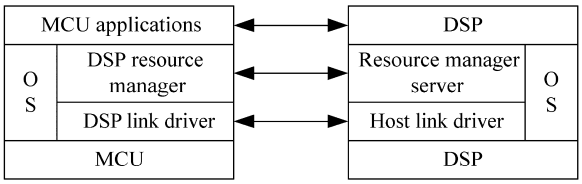

Fig. 5. The OMAP software architecture.

operating systems. It links these operating system environments together. The DSPBridge consists of a set of API and device drivers that are used for handling inter-processor communication and task control over the MCU–DSP interface. (Fig. 5).

A fundamental concept in OMAP MCU–DSP co-operation is that DSP functions as an autonomous slave to MCU, and as such can be considered as a multifunction hardware accelerator for applications on MCU. MCU keeps track of DSP resources in use on the DSP side (memory and processing capacity), and controls both the tasks and communication between processors. This is achieved by providing specified driver-level calls on the MCU side for different DSP tasks. Based on these driver calls, MCU makes task requests for DSP, which dispatches them to the local real time operating system (RTOS). On the MCU side, only these interfaces are visible to applications. As necessary, this concept actually allows the addition of several signal processors into same signal processing accelerator pool without changing the interface for user level applications. In the OMAP architecture the control of the peripherals of the DSP core processor has not been defined.

2.5. Mobile terminal software framework for microcontroller

The Nokia mobile-terminal MCU software framework is a messaging-based client/server software architecture where clients (applications) and servers communicate through a connectivity layer. A proprietary solution, a brief introduction to framework, can be found in Ref. [7]. A message protocol with a connectivity layer enables communication from one framework entity to another, regardless of whether they are within the same task, the same processor or even within different processors (Fig. 6). Essentially, the MCU software framework is a broker pattern architecture intended for enabling component software development for mobile communications. It is fairly similar to CORBA [16].

Servers control mobile terminal resources. A server retains the resource it controls in a server interface, and

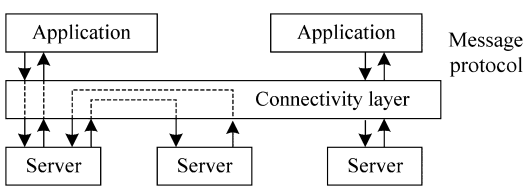

Fig. 6. A mobile terminal MCU software framework.

makes that interface available through the connectivity layer. The features of a phone are available through applications that use services offered by servers. Applications can use the services of all servers, and servers can use services belonging to other servers in order to provide the requested functionality.

The connectivity layer provides both point-to-point (request–response) and multicast (event indication) messaging services. Server developers and application developers can order events from servers to implement control in event oriented processing, while the actual event distribution mechanisms are hidden in the connectivity layer. An apparent shortcoming of the architecture is the lack of UNIX like communication pipes between servers and applications. That is an attractive solution to stream type data communications, and has been used with a low-overhead DSP operating systems such as SPOX [10], which is currently Texas Instruments DSP/BIOS II [5].

The framework considers information hiding at a higher level than hardware abstraction. Although it is a closed environment for the time being, it provides a rich environment for application development. It is also a viable solution for DSP software architecture, because it provides flexibility for hardware/software partitioning and software allocation decisions.

2.6. Summary

To wrap up: a hardware abstraction layer, such as that provided by Symbian OS, minimizes hardware-change-related software modifications and as such is a worthwhile goal; BREW [1] is an example of lightweight execution environment; the open audio decoder [4] approach is very DSP specific and efficient; the Nokia MCU software framework [7] provides advanced encapsulation. However, in a mobile DSP software architecture, all these features should somehow be combined. As a system platform, OMAP provides mechanisms for efficient communication between DSP and microcontroller resident tasks. This is necessary due to the data-intensive nature of multimedia applications; it represents the natural evolution towards hiding increasing complexity via a hierarchical, layered architecture and encapsulation.

A fundamental issue in DSP software is its ‘granularity’. From the software development point-of-view, it would be advantageous to develop small independent software entities, however at the same time DSP computing resources should be utilized with low inter-entity overhead. The key to achieving this compromise is the DSP software architecture.

3. Open mobile DSP software architecture

Mobile communications system architecture has to be flexible with respect to changes in hardware/software partitioning. It should support software portability between

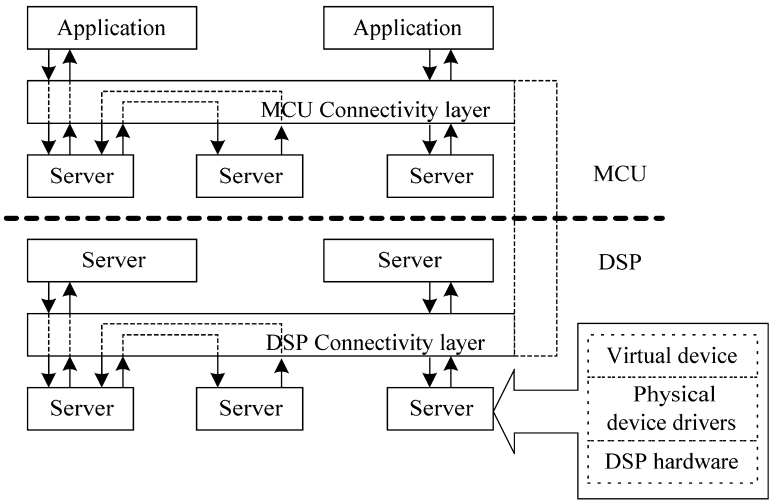

Fig. 7. The MCU framework extended to the DSP side.

DSP and MCU environments of the system, stretch according to available computational resources, provide a preferably uniform view to software architecture, and enable opening interfaces for third party software development. Fig. 7 proposes a solution based on the Nokia mobile terminal MCU software framework that fulfils these requirements. In practice, the MCU framework has been extended to the DSP side of the system.

Furthermore, the framework effectively hides the operating system from the software designer by providing its own abstractions (or mechanisms) and interfaces. This protects the legacy investment, as the framework provides standard patterns for communications that are the same on both MCU and DSP. A uniform system interface also simplifies the implementation of heterogeneous prototypes, in which different parts of the system are at different levels of maturity. In the architecture, hardware/software-partitioning changes are hidden by means of virtual devices. This is a major revision to the earlier practice, where hardware was accessed directly, that is, without standard interfaces. Consequently, changing the loop-type DSP architecture requires that hardware service interfaces are carefully designed. The benefits of legacy software adaptation come from simplified testing and more controlled software modifications with a view to future DSP hardware/software partitioning changes.

The architecture is designed to enable the separation of mobile terminal hardware and software development, in particular on the DSP side. Ideally, a hardware platform release will include a new hardware design and new physical device drivers, which have been tested with legacy software using existing virtual devices. Similarly, software designers could implement new data-channel functionality using existing product hardware, although the final product might need a new hardware accelerator to meet overall performance requirements.

Structuring of DSP software in servers with clear interface services provides encapsulation and a uniform view to the overall software design. The DSP connectivity layer acts as the glue between servers, defining standard communication mechanisms such as message passing and data pipes, which can be optimized for each hardware platform. DSP software developers regard the ‘standard’ server/DSP connectivity layer platform easy to learn, wherein the scalability and distribution properties of DSP software are inherently increased.

3.1. Characteristics of the DSP connectivity layer

As in any software architecture, the mechanisms of the framework connectivity layer can be designed to optimize message-handling capacity, data throughput, and communications costs. Communications costs can be categorized as memory consumption, latency, and CPU cycle overhead. On the MCU side, message-handling capacity appears to be important due to event-oriented processing, while DSP must be optimized with respect to stream type data communications. Consequently, implementations of the connectivity layer differ somewhat between MCU and DSP.

As an example, Table 1 shows the communications costs of two commercial DSP RTOS. The TMS320C55x CPU cycle penalty with OSE is measured from a low-priority task *send message* call to a high-priority *receive message* return. The context switch from a low to high priority task is thus included. DSP/BIOS II pipe communication cost with TMS320C54x core is a sum of *pipe put* and *post semaphore* calls [6]. As both RTOS use pointer passing instead of

Table 1
The communications costs of the OSE and DSP/BIOS II DSP real time operating systems

	Send to receive CPU cycles	Interrupt latency CPU cycles
OSE	298	210
DSP/BIOS II	399	45

message copy, the size of the message does not affect the CPU cycle count.

Regardless of differences in communications mechanisms, i.e. message passing versus pipes, costs are closely the same. However, both mechanisms—packet switched, like message passing and circuit switched, like pipes—are needed in the DSP framework connectivity layer, to provide tools for DSP developers to build efficient and configurable DSP software systems. In communications applications the selection between the operating systems shown in Table 1 depends on factors such as packet size, packet arrival frequency, and the software architecture.

3.2. Use of DSP connectivity layer instead of loop type model

Fig. 8 shows how efficient data channel DSP software can be configured from servers with DSP framework communication mechanisms. The servers use both message passing and pipe communication via the DSP framework. Standard message passing is used to configure the server mode and pipe configuration, whereas pipes are used to enable efficient runtime data exchange between servers. The correlation with the loop-type model presented in Fig. 1 is evident. The benefits of this approach have been comparatively analysed by Purhonen [13].

Fig. 9 shows the structure of the connectivity layer and message-passing style communication paths, with different MCU-framework server allocations for the system.

In the MCU framework, the communication manager provides a uniform messaging interface in a distributed system. It does so by handling the task mailbox and

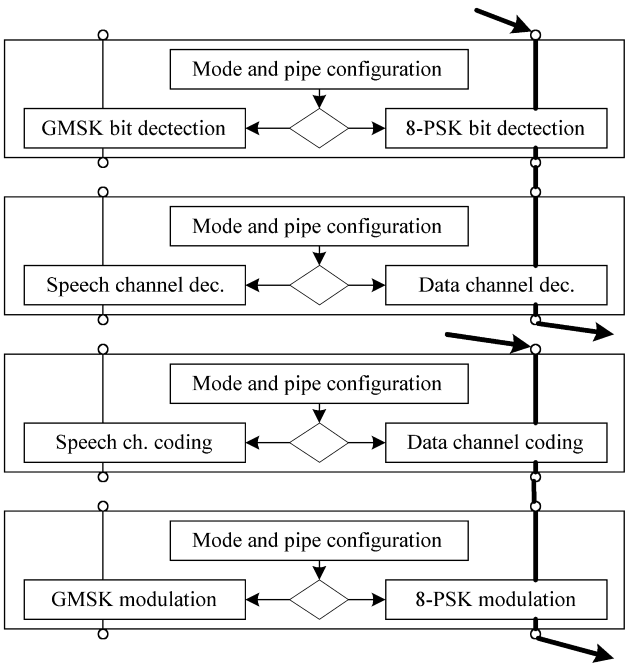

Fig. 8. The server structure and DSP framework communication mechanisms.

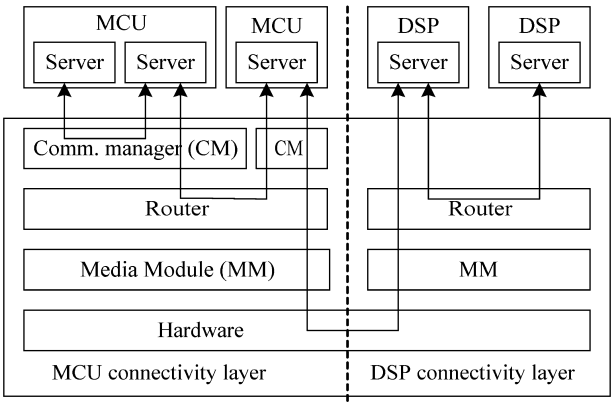

Fig. 9. The connectivity layer structure and its communication paths with message passing.

internal message queues, and by managing event delivery. The router level relays messages to and from communication managers, or to and from external devices through media modules. In addition, the router takes care of indication distribution to other communication managers. Media modules are tasks that implement link layers and they have standard interfaces.

In message passing, the DSP framework connectivity layer on the DSP side uses resources more economically than the MCU framework, at the same time offering essentially the same communication mechanisms. Both sides of the system can be optimized, based on their respective computation and communication profiles. One difference between the DSP and MCU sides is that each DSP task contains only one DSP framework server: in other words, communication managers can have simple implementation, or even collapsed to the router. This is a natural consequence given that DSP servers use message-oriented communications mostly with MCU servers and applications, rather than with other DSP servers. Loop-type DSP legacy architecture can initially be implemented as a server for the DSP framework, only requiring the communication interface to be changed.

3.3. DSP connectivity layer API

Fig. 10 depicts the connectivity-layer interface services ‘APIs’ provided for MCU framework servers. Dashed lines indicate those features not included in the DSP framework. Interfaces for sending and receiving messages form the basis

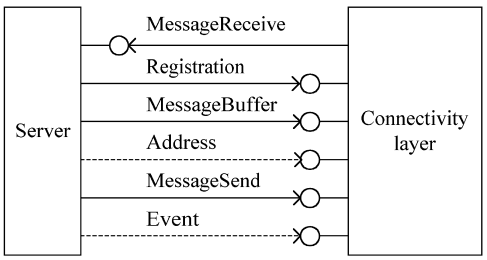

Fig. 10. The message passing APIs of the connectivity layer.

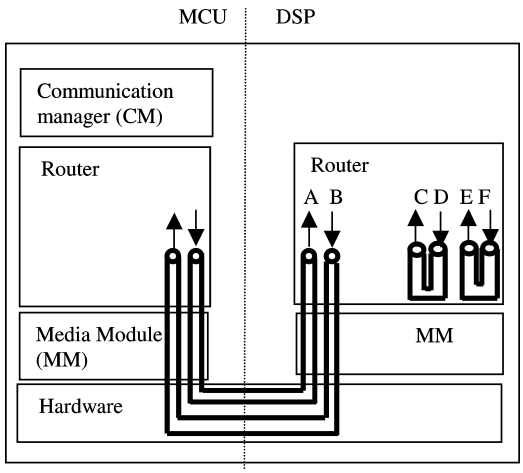

Fig. 11. The DSP framework connectivity layer communication pipes for stream data transmission.

for the DSP framework connectivity layer, and the message buffer interface offers the necessary dynamic-memory allocation and protocol message-handling functionality. The registration mechanism can be exploited by building dynamically re-configurable systems, and its usage is not limited to the booting phase. In mobile communications, this feature is of practical use for multistandard devices.

The address interface hides the location of the server. It uses methods for managing dynamic proxy objects that represent remote servers. Thus, the MCU framework application or server developer does not need to know the actual location of the communicating server. This feature is relatively unimportant for DSP developers, who know the few DSP servers and the main communicating partners on the MCU side. On the other hand for third party development it is necessary to provide a clean interface. The event interface (providing methods for sending and subscribing to message-protocol event messages) has been omitted from the DSP side. The complexity of the communication manager does not justify using this mechanism for controlling stream-type DSP data processing. Instead, a pipe mechanism is used to connect the DSP application and server input/output data streams.

Fig. 11 depicts those data pipes in the DSP framework architecture that are unidirectional and can be used to pass blocks of data in inter-/intra-task communications. Both polling and operating system scheduling mechanisms are used for data transfer synchronization. A programmer will see the pipes as a limited set of named input/output ports.

Many of the DSP servers encapsulate hardware operations as virtual devices, making the architecture more understandable for software developers. From the system development point-of-view, using a single standard and highly optimized communications mechanisms saves DSP resources. Furthermore, in the DSP framework, abstraction of the communication mechanism hides its actual implementation from DSP developers.

3.4. The hardware abstraction layer

The task of the hardware abstraction layer is to provide a view to the virtual hardware. If the underlying hardware changes, the virtual device—that is, its abstraction—stays the same. Commonly used hardware abstractions include the virtual radio and virtual audio codecs. Other, similar abstractions can also be added. In practice, many DSP framework servers encapsulate a virtual device.

There are relatively few servers, thus simplifying hardware/software partitioning and testing. This is important because of the changing hardware/software boundaries over different product generations. In practice, the hardware/software boundary changes are invisible from outside the servers, and the control solutions in the servers can be kept proprietary.

Each virtual device interface fulfils the following criteria:

- A virtual device interface should hide the hardware-related DSP control software. This includes the algorithms for hardware tuning, such as automatic gain and frequency control of virtual radio. It simplifies the early, independent testing of virtual devices in heterogeneous system prototypes.
- A virtual device interface should make it possible to use legacy DSP software functionality—such as GSM bit-detection and synchronization algorithms—via encapsulation.
- A virtual device interface should minimize event-based communication between clients and servers, leading to efficient data transfer in the system.
- A virtual device interface should facilitate the addition of new DSP features (such as new audio/channel codecs) which can use those services provided by virtual devices.

The virtual device concept and DSP framework hide differences between technology solutions over product generations. Interface standardisation enables third-party software development via the release of documentation and platform-specific development tools (at least in principle). Nevertheless, the DSP environment requires a high level of expertise from the developers, due to the limited availability of computational resources.

3.5. Operating system issues

For satisfying timing constraints in mobile communications terminal software predictability is the most important issue. Pre-runtime scheduling is often the only practical means of providing predictability [12]. In this respect the mobile terminals are typical complex hard real-time systems.

Ideally, DSP software tasks should be kept small, in order to make DSP software architecture reusable and configurable. In practice, the cost of scheduling of

Table 2
A comparison of different real-time operating systems

	Multistack		Single-stack, non-pre-emptive
	Pre-emptive	Non-pre-emptive	
RAM	Less efficient	Efficient	Most efficient
Utilization	Less efficient	Efficient	Efficient
Analysis	Analysis easy	More effort needed	More effort needed
Testability	Less predictable	Predictable	Predictable

selected RTOS limits the number of tasks available. RTOS selection also affects CPU utilisation and internal RAM usage, which are both important issues for mobile-terminal DSP software architecture when aiming at efficient resource usage. Table 2 evaluates different types of RTOS with their respective resource usage efficiency and ease of software development.

A single-stack operating system uses a minimum amount of internal RAM for storing task data, as every task is executed until completion, and pre-emption is not allowed. Non-pre-emptive operating systems force the DSP designers to do a complete timing analysis and even task-slicing in order to meet all the timing requirements. This clearly requires a very high level of expertise and knowledge of the entire DSP software system. On the other hand, software written for non-pre-emptive operating systems is more reliable and easier to debug, which makes the effort needed in the initial design stages more tolerable.

Non-pre-emptive scheduling also leads to high CPU utilisation, as tasks are executed until completion and there are no unnecessary context switches. Power saving is made easier since each task is responsible for shutting down its hardware accelerators, and an idle task does not need detailed task status information in order to enter the power saving mode.

DSP software functionality is mostly periodic and the task-scheduling order can be solved in advance. It is therefore questionable if pre-emptive priority-based scheduling alone is the best solution for DSP terminal software. In practice, a clean scheduling solution does not seem to be feasible. Pre-runtime scheduling has problems with the obvious dynamic runtime scheduling needs of multimedia multimode phones. The DSP software architecture solutions presented in this paper suggest a fairly small number of tasks that loosely emulate the task division of loop-type DSP software architecture.

4. Mobile terminal software: case studies

Next, three case studies are presented on the use of the proposed architecture, to demonstrate design scalability, dynamic software reconfigurability, and hardware/software partitioning flexibility. The selected cases constitute typical design problems in multimedia terminal DSP development. They illustrate the key points of the DSP framework solution: the modularity and distribution properties of the framework, the suggested pipe communication mechanism, and hardware abstraction with virtual devices.

Single slot traffic allocation								Multislot traffic allocation							
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
R			T	M				R	R	R	R		T	M	

Fig. 12. An example of GSM single slot and multislot traffic allocations.

4.1. Design scalability: single slot/multislot data communications

Fig. 12 depicts a GSM product-family problem, where DSP performance must stretch between different multislot capabilities. In GSM, data is transmitted in frames, each frame containing eight slots each with 24.7 kbit/slot/s user data [9]. During its 4.615 ms operation period, a low-end terminal for traditional circuit switched calls need only handle single slot receive-and-transmit and neighbour cell monitoring. The computations required by this task alone consume approximately a quarter of the capacity of a 100 MIPS DSP. In a high-end terminal with multislot data communications capability, the DSP must process four receive slots (R) and one transmit slot (T) in 4.615 ms, in addition to neighbour cell monitoring slots (M).

As a consequence the high-end terminal needs more than one DSP power to cope with multislot requirements. For the low-end product, a multiple DSP solution is not cost-effective. However, from the product family point-of-view, modem, audio and video functionality must stretch either to single or to multiple DSPs.

Fig. 13 shows DSP software modem functionality mapped to one DSP, and audio/video functionality mapped to another DSP. In order to simplify the reuse of software from a single DSP environment, it has been allocated to servers with protocol interfaces. This simplifies the distribution of the DSP functionality. When scaling from

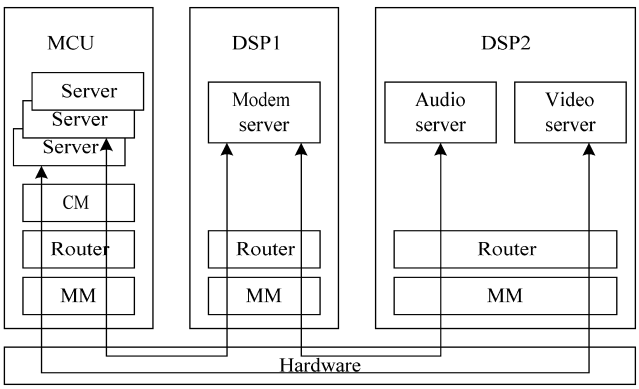

Fig. 13. An example of scaling software functionality to two processors using the proposed architecture.

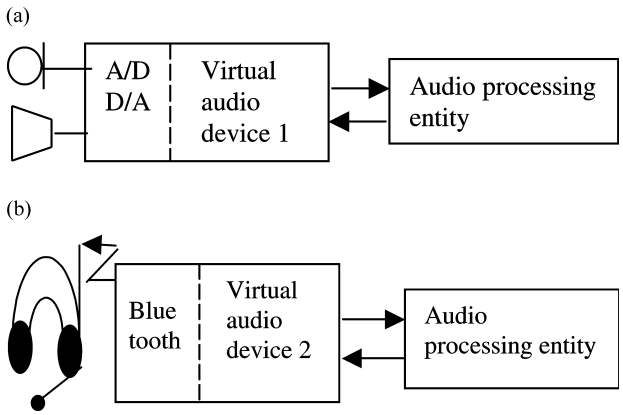

Fig. 14. (a) The audio processing chain with an internal terminal microphone and loudspeaker. (b) The audio processing chain with an external microphone and loudspeaker.

one to two DSPs, the DSP framework connectivity layer hides the server changes. By adding a low-level media module, the necessary communication mechanism modifications are performed. For the sake of presentational clarity, the inevitable pipe-based communications are not shown in the diagram.

4.2. Dynamic software reconfigurability: audio services

Mobile terminal audio can use an internal microphone and loudspeaker, or an external headset. In the standard case depicted in Fig. 14a, the audio processing entity communicates with the virtual audio device that hides the internal microphone and loudspeaker interfaces. If a conventional audio accessory—such as a handsfree set—is connected to the terminal, the virtual audio device remains unchanged. However, when a wireless 'Bluetooth' audio accessory is used, the loudspeaker and microphone interface undergoes essential changes. In order to use the Bluetooth audio device, the terminal replaces the initial virtual audio device with another virtual audio device, as shown in Fig. 14b. The communication mechanism allows the change to take place even during a call; the data streams are redirected using pipes.

The situation from a software architecture viewpoint, both before and after the switchover to a Bluetooth device, is shown in Fig. 15a and b, respectively. The virtual audio devices, audio processing entity, and Bluetooth are all implemented as servers. All stream data communication is piped. For the sake of clarity, the protocol messaging needed in order to set up pipe connections is omitted from Fig. 15a and b.

4.3. Hardware/software partitioning flexibility: GSM modem

The functional blocks and data paths of a GSM modem from a DSP perspective are shown in Fig. 16. The modem is the heart of a GSM terminal, and takes care of all the radio

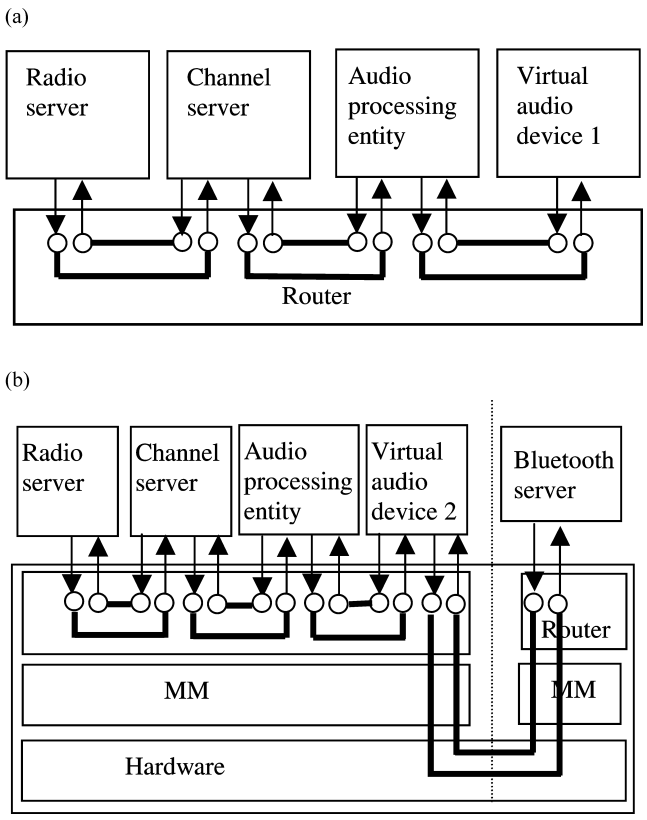

Fig. 15. (a) Audio data stream connections using an integrated microphone and loudspeaker. (b) Audio data stream connections using a wireless headset.

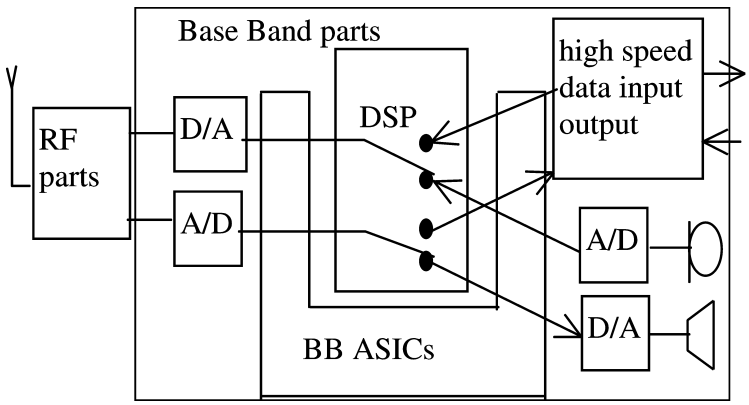

Fig. 16. GSM modem functions and data paths from a DSP standpoint.

frequency (RF) and base band (BB) processing necessary for transmitting and receiving the RF signal via the air. It performs bit detection and modulation, and channel decoding and encoding for GSM slots. The modem functionality must stretch from single-slot, voice-only terminals to multislot, high-speed data terminals in the same product family. This means different hardware/software partitioning decisions for BB ASICs and DSP software.

Fig. 17 depicts the GSM DSP modem software, as allocated to two servers. The radio server implements the virtual radio interface, the functionality of which is most likely to change according to different product category requirements, such as a multislot capability. The channel server implements logical channel control, channel encoding and decoding. These are not hard real-time tasks. The Viterbi decoder may be implemented either in hardware or software.

The most time-critical DSP software controls the hardware, and is contained in the radio server. While the software implementation of bit-detection algorithms in a single-slot terminal changes to a hardware implementation in a multislot terminal, the interface which transmits the detected soft decision bits to the channel server remains the same.

Pipes are used within servers to connect data streams together, thus enabling runtime hardware/software partitioning. For example, in a channel server experiencing high data rates, the decoder connects de-interleaved data streams to a 64- or 256-state hardware Viterbi algorithm. In the control channel mode, the pipe is connected to a 16-state software Viterbi algorithm implementation.

Abstractions of the virtual radio interface are presented in Fig. 18. The *measure* interface provides methods for measuring signal levels received over a particular band, and the given list of neighbours, so as to support cell selection. The radio server contains hardware dependent initializations, RF tuning, and the timing of BB and DSP algorithms that measure and control the gain of A/D converters during measurement. The responses are signal levels received in decibels.

In a cellular network, the *synchronise* interface provides methods for initial cell synchronisation and neighbour cell synchronisation. This interface hides the implementation of DSP synchronization algorithms, automatic frequency tuning, and the handling of frame timing information.

The *receive* and *transmit* interfaces separate channel encoding and decoding from time-critical, hardware-based bit detection and modulation. These interface services are used for new channel coding schemes, or for logical channel structure implementations over evolutions of the cellular standard.

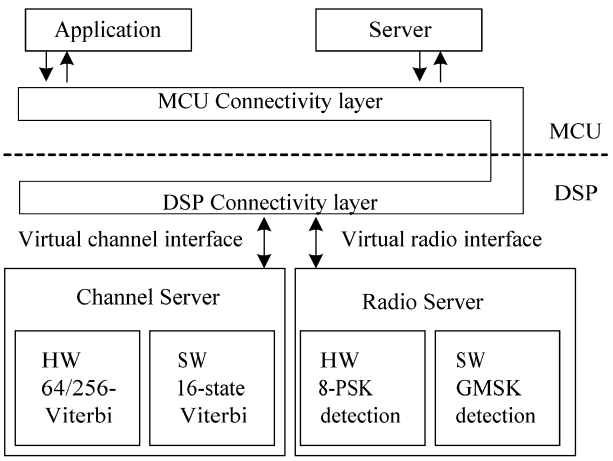

Fig. 17. GSM modem functionality and a hardware/software partition.

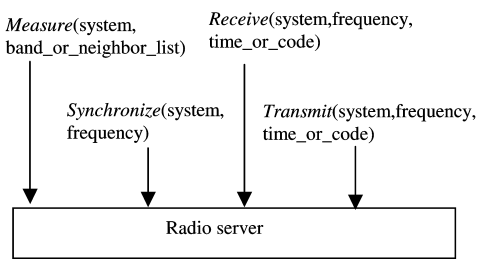

Fig. 18. A radio server and its virtual radio interface.

5. Discussion

In practice, there are no fundamental obstacles to exploiting advanced architectures and design methodologies for embedded DSP software. The principles of object-oriented design, in particular the concealment of information at a service level, provide a structured view of a system to application programmers. A stable base-system architecture is important for both third-party application development and in-house activities. Similarly, the abstract hardware—that is, the virtual devices—conceals hardware platform changes from server software designers.

However, communications DSP software should optimize the number of instructions used per received bit. Due to the increasing data rates and low power requirements much of the implementation is always in hardware, but the HW/SW partitioning is prone to change between product generations. This sets special requirements for the software architecture to ensure both efficiency and scalability.

The proposed architecture and communication mechanisms provide well-defined interfaces, which are mandatory for in-house/third-party, component-based development. In the case of third-party development, each component under the DSP framework connectivity layer is a server that connects to the servers provided by the environment. From the low power point of view the proposed software architecture is in line with the locality principle described by Havinga and Smit [8]. A justification for server approach is their observation that 'at system level locality can be applied'.

Naturally, development tools are needed for enabling software component testing in heterogeneous prototypes. In principle, the DSP framework allows the testing of new server and application software (for example, in a PC connected to the mobile terminal). Although this is not a complete solution to the system integration problems associated with third party development, it is a major step towards an open software platform.

The proposed DSP software architecture enables legacy software, such as a GSM modem, to be encapsulated as a DSP framework server. Simultaneously, it provides a gradual transition path for transforming legacy code for a more open software development environment. Among the first actions necessary is to export hardware interfaces into virtual devices.

The embedded software systems of mobile communications terminals have so far been closed for third-party development. DSP software has been the most restricted element of all. However, the need for advanced services has created a pressure to open the software platforms. Furthermore, cost-effective, multimedia-related services clearly need to access DSP resources. Consequently, architecture and mechanisms that simplify third party development may contribute to an application explosion.

It is in the interests of the platform manufacturers to make a controlled transition to open software development. The most important element in this process is the software architecture.

6. Conclusions

Despite its very performance-oriented requirements and currently closed nature, the DSP software of mobile communications can be opened even to third party software components. The architecture proposed in this paper is based on a known and accepted design pattern, and provides a basis for opening the necessary interfaces in a controlled manner. The communications mechanisms of the proposed architecture can be implemented very efficiently in a DSP environment, and the performance is not compromised by the overhead.

The solution clearly supports system scalability and dynamic reconfigurability. Together with a flexibility with respect to changes in hardware/software partitioning, these result in a stable environment for third-party software. It is a strong argument for the proposed architecture.

References

[1] Anonymous, Binary runtime environment for wireless [Online], Available from http://www.qualcomm.com/brew/.
[2] Anonymous, OMAP1510 Application processor for 2.5G and 3G wireless devices [Online], Available from http://www.ti.com/sc/docs/apps/omap/overview.htm.
[3] S.S. Bhattacharyya, E.A. Lee, P.K. Murthy, Software Synthesis from Dataflow Graphs, Kluwer, Boston, 1996, p. 189.
[4] T. Chen, J. Datta, B. Karley, An open DSP software architecture for consumer audio, IEEE Transactions on Consumer Electronics 45 (4) (1999) 1253–1258.
[5] D. Dart, DSP/BIOS II Technical overview, Application report, SPRA646 [Online], Available from www.ti.com.
[6] S. Dirksen, DSP/BIOS II Timing benchmarks on the TMS320C54x DSP, Application report, SPRA663 [Online], Available from www.ti.com.
[7] K. Hautamäki, Integration of GPRS into GSM mobile phone cellular software, Department of Electrical Engineering, University of Oulu, Finland, Master's Thesis, 1998, pp. 25–30.
[8] P. Havinga, G. Smit, Design techniques for low-power systems, Elsevier Journal of Systems Architecture 46 (2000).
[9] J. Hämäläinen, J. Knuutila, Migration towards multimedia capablity in GSM high-speed data terminals, IEEE GLOBECOM Global Telecommunications Conference 1 (1998) 89–94.
[10] L. Keate, A real-world approach to benchmarking DSP real-time operating systems, Wescon/97, Conference Proceedings, 1997, pp. 418–424.
[11] F. Koushanfar, M. Potkonjak, V. Prabhu, J.M. Rabaey, Processors for mobile applications, IEEE International Conference on Computer Design (2000) 603–608.
[12] D.L. Parnas, J. Xu, On satisfying timing constraints in hard-real-time systems, Software Engineering, IEEE Transactions on Software Engineering 19 (1) (1993) 70–84.
[13] A. Purhonen, Quality driven multimode DSP software architecture development, VIT Publications 477, Espoo, Finland: Technical Research Centre of Finland (VTT), 2002, 150p [Online]. Available from http://www.inf.vtt.fi/pdf/.

[14] S.H. Redl, M.W. Oliphant, M.K. Wever, An Introduction to GSM, Artech House Inc, 1995, 379 p.
[15] M. Tasker, Professional Symbian programming, Mobile Solutions on the EPOC Platform (1995) 1031.
[16] T. Scallan, CORBA Primer Whitepaper [Online], Available from www.omg.org/news/whitepapers/index.htm.

Kari Jyrkkä received his MSc degree in electronic engineering from the University of Oulu, Finland 1990. In 1991 he joined Technical Research Centre of Finland, where he participated to spread spectrum receiver collaboration project and simulation of receiver algorithms of a power line communication system. From 1993 to 1998 he was with Nokia Telecommunications Radio Access Systems, where he was developing DSP software and hardware to GSM base stations. During 1998–2000, he was with Nokia Mobile Phones contributing to specification of narrowband GSM control channel solution. Since 2000 his research interest at Nokia Mobile Phones has been DSP software architectures for multimode multimedia terminals.

Olli Silven received his MSc and Dr Tech degrees in electrical engineering from the University of Oulu in 1982 and 1988, respectively. He was a post-doctoral researcher at the Center for Automation Research of the University of Maryland in 1989–1990. He was nominated an associate professor of computer engineering in 1993, and professor of signal processing engineering in 1996, both at the University of Oulu. He has been the prime proposer and leader of several domestic and international collaborative and industrial projects in the fields of image sequence processing and analysis, very high speed visual inspection technology, and wireless digital video.

Olli Ali-Yrkkö received his MSc degree in electronic engineering from Tampere University of Technology, Finland in 1993. He joined Nokia Mobile Phones in 1991, where he has been involved with several speech coding related tasks varying from product development to GSM standardization activities.

Ryan Heidari received his BSE and MSEE degrees from San Diego State University in 1975 and 1977, respectively. He continued with his advanced studies and research and in 1985 he completed his Engineers Degree (EEE) from University of Southern California with concentration on digital speech, image and video coding. During his career, he has taken number of technical and leadership positions in companies such as Emerson Electric, Xerox, Rockwell International, and Dictaphone Inc. He joined Nokia Mobile Phones in 1992 and since then he has been working as Engineering Manager of Multimedia Application team. He is very active in TIA-TR45.5 (CDMA technologies) and vice-chaired the Speech Coding group. He has participated in many technical papers and publications including four granted patents with Nokia. He is a senior member of IEEE.

Heikki Berg received his MSc degree in electronic engineering from the University of Oulu, Finland in 1998. He has been staff of Nokia Mobile Phones since 1995, where he has been involved in the design and implementation of the baseband receiver algorithms for IS-136, GSM and EDGE. Recently he joined Nokia Research Centre where he is involved with smart antenna research. His research interests are estimation and detection algorithms.