A SOFTWARE FRAMEWORK FOR MODEL DRIVEN CONFIGURATION AND CONTROL OF WIRELESS BASEBAND SYSTEMS

Rollo Burgess (Toshiba Research Europe Limited, Bristol, UK;
rollo.burgess@toshiba-trel.com)

ABSTRACT

In this paper we describe an application of the software-defined radio concept to baseband signal-processing in current and future cellular and hotspot wireless systems, such as WCDMA and WLAN. We begin by highlighting the challenges that confront us in implementing a truly flexible baseband system. These challenges stem from the real-time and compute-intensive nature of the signal-processing, the limitations and diversity of processors and of course the need to minimise power consumption. Taking a strictly software-centric viewpoint we show how these challenges can be met using software engineering principles, state-of-the-art software technologies and several novel concepts. We combine these into a proposed baseband software framework. The framework defines a runtime environment, which configures and controls low-level signal-processing through the action of hardware and algorithm models and reusable software components. Finally we point out the unresolved issues and areas for future work.

1. INTRODUCTION

Today's mobile baseband systems are not known for their runtime flexibility. A truly Software Defined Radio (SDR) [1] requires a level of baseband flexibility more readily associated with desktop computing. Unlike a PC application, however, automated baseband configuration must guarantee applications are both predictable and reliable. In other words spurious radio emissions and sudden crashes are not acceptable behaviour. These guarantees must be made in the face of technical challenges derived from the unique nature of wireless baseband signal-processing, and the computing environment where it is undertaken. In particular the required functions are constrained by time and demand enormous processing power. In addition, implementations must be sparing in their use of the limited power reserves available to mobile terminals [2] [3].

Further we undertake these challenges for the most generic processing environment, one without a standard hardware (HW) platform. Rather we propose a more flexible solution using a standard software (SW) infrastructure.

In the following section we describe our assumptions regarding the HW and SW environment in more detail. In section 3 we describe the resulting challenges, and in section 4 we show how these challenges can be met using modern software engineering techniques. Finally in section 5 we review our proposals and highlight those areas requiring future work.

2. ASSUMPTIONS

Figure 1 portrays our assumptions concerning the terminal signal-processing HW, using elements of the Unified Modelling Language (UML) deployment diagram [4].

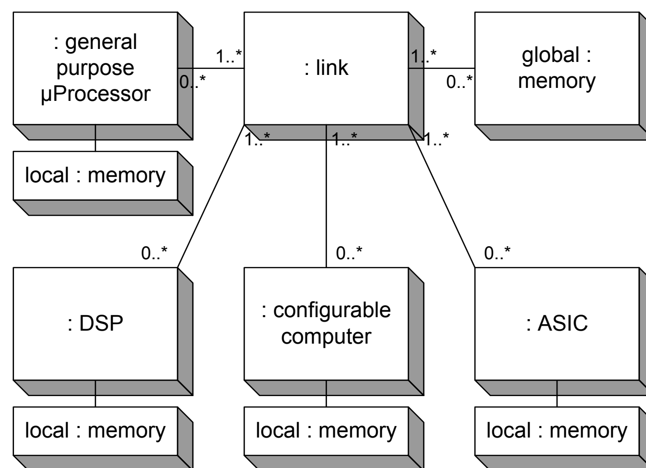

Figure 1: assumed hardware architecture

Essentially a network of program, control and data *links* physically connect processors to each other and also to one or more global memory devices. We assume a *control domain* exists, containing at least one General-Purpose microProcessor (GPP), as shown at top left in Figure 1. The *data-path signal-processing domain* is assumed to consist of any number and combination of the three fundamental types of *accelerating processors*, Digital Signal Processors (DSPs), Configurable Computers (CCs) [5] and Application Specific Integrated Circuits (ASICs). DSPs are considered

 Copyright © 2003 SDR Forum. All Rights Reserved

microprocessors for accelerating signal-processing, while CCs and ASICs are *HW accelerators*. Finally we assume each processor owns a tightly-coupled local memory device. The link between the local memory and its processor is considered private; it is not shared with other processors and can be expected to have a low latency.

Figure 2 shows our assumptions regarding the wider software environment of the terminal, in terms of dependencies between packages representing SW layers.

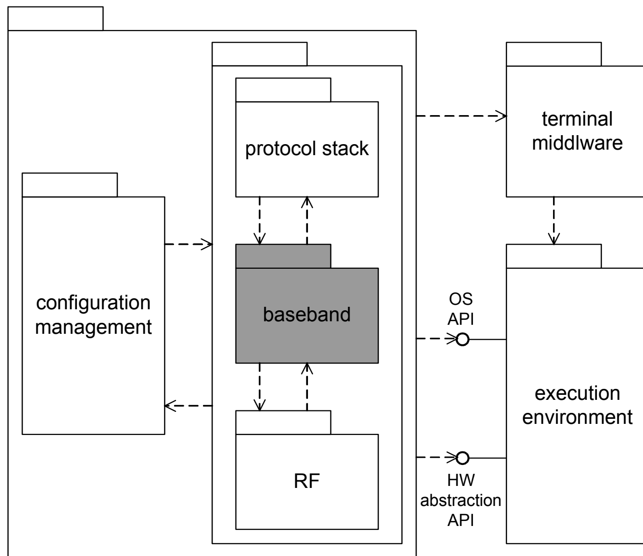

Figure 2: assumed software environment

The reconfigurable baseband layer is shown shaded at centre. It is logically part of a data processing chain, running from top-to-bottom that includes reconfigurable protocol stack, and Radio Frequency (RF) layers. To the left a high-level *Configuration Management* (CM) layer has responsibility for directing the reconfiguration procedures. To the right is the lowest layer, the execution environment. This layer is critical since it provides independence from low-level HW issues. Two Application Programmer Interfaces (APIs) are shown. The Operating System (OS) API acts as a virtual operating system, by providing a standard interface to OS services. Likewise the HW abstraction API provides a standard approach to accessing radio specific HW. Finally the terminal middleware package wraps several execution environment functions to provide standard mechanisms for inter-process communication.

3. CHALLENGES

The first challenge is the diversity of target terminal hardware architectures, with each manufacturer having a preferred, possibly proprietary, architecture. This issue is compounded as each architecture will contain a heterogeneous set of processors with a wide range of programmability [3]. Microprocessors such as DSPs and GPPs will continue to dominate as they are well-understood and very flexible. At the other end of the spectrum ASICs provide severely limited or zero flexibility, however we expect they will continue to be necessary for low power and to meet the high computational requirements of future radio access technologies [3]. In-between are the new breed of CCs which are expected to play an increasing role due to their inherent flexibility and relatively low-power performance [5].

In fact the limited power available to battery powered terminals continues to be a major constraint [6]. Therefore minimising power usage is an important challenge facing reconfiguration. A further challenge also concerns constraints, namely the need to guarantee the hard real-time deadlines inherent to baseband signal-processing.

Two challenges remain. Installation of new or partial baseband applications must have no observable effect on the radio behaviour of existing baseband applications. Lastly, dynamic configuration forces us to automate what is essentially a complex hardware / software co-design process, normally handled by a team of experienced engineers [3].

4. SOLUTIONS

To meet the challenges outlined in section 3 we describe a number of solutions that rely on software abstraction.

4.1 Software Abstraction

The challenges described in section 3 require the efficient management of complexity. Fortunately software engineering is particularly good at tackling complex problems using *abstraction*. Abstraction identifies the relevant aspects of an entity while ignoring or hiding the rest [7]. In object-orientation (OO) classes capture the essential characteristics of a set of objects by encapsulating both their behaviour (methods) and state (attributes). Abstraction is not limited to OO and is widely used in software engineering to simplify a system by hiding the detail of a service behind an interface. Table 1 lists some common software abstractions.

Table 1: well-known software abstractions

Abstraction	Encapsulates	Interface
method	sequence of actions	operation
process	concurrent and iterative sequence of actions	synchronised operations
object	data and methods	operations
component	data and methods	reflected operations
active object	object and process	synchronised
active component	component and process	synchronised operations / middleware
package	groups of related objects / components	operations
middleware	communication mechanism	communication operations
design pattern	software mechanism	operations
model	essential features of a device or algorithm	operations
software layer	components / objects	operations (API)
framework	all other abstractions	operations

The history of software engineering shows a trend toward using higher-levels of abstraction [8]. We aim to follow this trend by raising the level of abstraction used in wireless baseband systems. The following abstractions are singled-out as being crucial to our approach:

- *Process*. An abstract representation of behaviour in a concurrent system. A Process is commonly defined as an iterative sequence of actions. Typical software implementations are known as tasks or threads. Message-based communication between concurrent processes is often made via a *channel* abstraction [9].
- *Software Component*. Any software artefact that is deployed at runtime [4]. We recognise three types; *dynamic executable components* (d-ex-components), *HW configuration components* (hw-c-components) and *SW configuration components* (sw-c-components). D-ex-components enable runtime flexibility and off-the-shelf software reuse. They provide a set of services whose interfaces can be exactly determined at runtime, a capability known as *reflection* [10]. Knowing the interfaces allows d-ex-components to be dynamically connected together at runtime to compose an application. Hw-c-components are non-executable components that specify how a HW accelerator should be configured for a specific function. Examples include arranging part of a CC to act as a rake receiver or parameters for initialising a turbo-decoder ASIC to the UMTS standard. Sw-c-components are non-executable meta-software components. These specify the runtime arrangement of other groups of SW components.
- A *model* is an abstraction that captures the essential features of a domain-specific device or method. To our knowledge no one has explicitly used modelling in a dynamic, embedded runtime environment, though this is certainly possible given adequate resources. Models are more commonly used during design and development for simulation and problem solving, for example [3]. Executable systems must be constructed under a *Model of Computation* (MoC) that is a set of "laws of physics" that govern the interaction of components within the system. These are particularly important for concurrent systems, such as those described by processes and channels [10].
- A *software framework* is a re-useable infrastructure that provides common structure and behaviour for a particular application domain [11]. They can be thought of as partially completed applications, pieces of which are customised by a developer to complete an actual application. Frameworks are commonly composed from *design-patterns*, another abstraction which is essentially a software mechanism for solving common [12] and domain-specific [11] implementation problems.

We have already seen abstraction at work using layers in our assumptions about the software environment. For example the terminal middleware abstracts inter-process communication. We will now focus on how the abstractions discussed above can be applied to the challenges facing flexible baseband signal-processing.

Figure 3 shows an example application for part of an extremely simplified UMTS-style receiver. We will use this example to illustrate the solutions in the following discussion.

4.2 Processes

We begin by tackling the issue of diverse heterogeneous HW architectures. Our approach is to decompose all required functionality into processes that communicate via channels. Assuming independence from inter-process communication issues, using an appropriate middleware to fulfil the channel communication, we are free to develop a multitude of different implementations for each process. In this way we can target processes to the most appropriate processors. In the example, configuration and control processes are assigned to the GPP while the signal-processes are assigned to the accelerators, i.e. the DSP and CC. Signal-processes can also benefit from multiform implementation for different types of accelerators. Such processes have maximum opportunity for reuse; they can be deployed in many ways within a single terminal and indeed to multiple manufacturers' terminals.

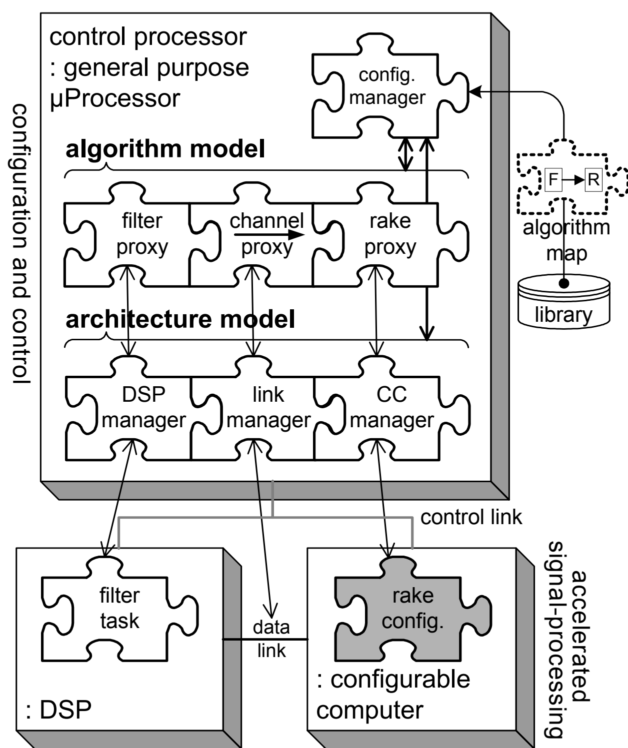

Figure 3: example simplified application

4.3 Components

How do we implement the polymorphic processes of the previous section? Our answer is to encapsulate them in d-ex-components. This approach also helps meet the challenge of automating the reconfiguration. The traditional approach to creating an application is an engineering one; a programmer develops objects, and these are grouped at design-time to form an application. In contrast d-ex-components are suited to manufacturing; the application is automatically assembled from component parts which are dynamically bound at runtime. Together decomposition of functionality into communicating processes and implementation of these using d-ex-components gives great flexibility. A baseband application can be freely assembled from any available components that meet the terminal's needs. The minimum criterion for choosing a component is compatibility with one of the terminal's processors. Figure 3 shows several d-ex-components using an open-jigsaw symbol with solid border; for example the **DSP manager** and the **filter task**.

Of course more sophisticated criteria will be needed to truly optimise the implementation. In particular the challenges of meeting real-time constraints and minimising power can only be met dynamically if sufficient performance metrics are available. One way to do this is by building processor-specific performance figures, such as worst-case execution time and average power consumption, into the components. However signal-processing components should be just that, components optimised for processing signals. For example a signal-processing d-ex-component which is hand-crafted in assembler for a DSP, such as **filter task**, should not be burdened with control-domain functionality. We therefore propose separating this behaviour by creating a control-domain representation of the implemented signal-process, called a *signal-process proxy component*. The signal-process proxy represents everything about the signal-process except the core "number-crunching". Figure 3 shows two of these; **filter proxy** and **rake proxy**. Proxies are therefore interrogated for performance metrics and memory requirements, they are used to initialise their processes with parameters, and to start and stop the signal-processing, and so on. Each proxy is tightly coupled to an implementation of the signal-process using an accelerator; however it runs in the control domain. For example **Filter proxy** identifies **filter task** as its implementation. **Rake proxy** similarly identifies **rake configuration**. The latter is a hw-c-component for the CC, and is shown as a shaded jigsaw piece. Once again we see abstraction at work. Diverse and heterogeneous accelerators can be configured and controlled just by manipulating proxy components. In this way the CM is relieved of any direct knowledge of the underlying accelerator HW. As far as it is concerned configuration is only a matter of managing software components, which is what we would expect for a SDR solution.

4.4 Modelling

But how can the CM choose the set of proxies that map function to architecture in an optimal manner? This is perhaps the most difficult challenge. To achieve this we propose moving to a higher level of abstraction, one where we model the interaction between the static terminal architecture and the dynamic baseband algorithms. We define an *architecture model* as a representation of the accelerating processors and their physical communication links, using entities called *HW managers*. In Figure 3 the architecture model is shown in the lower portion of the GPP; it contains 3 managers, one for each accelerator and one for the data link between them. We also define purely functional *algorithm models* that are used to explore the implementation of baseband applications. Each baseband application is represented by a single algorithm model. In Figure 3 part of a UMTS algorithm model is shown above the architecture model.

Reconfiguration, involving a new application, begins with the construction of a new algorithm model according to a machine-readable specification, called an *algorithm*

Proceeding of the SDR 03 Technical Conference and Product Exposition. Copyright © 2003 SDR Forum. All Rights Reserved

map. The map is a sw-c-component which specifies prototypical versions of both the processes and communication channels of the algorithm. The algorithm model is actually constructed from proxy components that realise these prototypes. For example in Figure 3 an algorithm map (open jigsaw piece with intermittent border) is retrieved from the library. The CM parses it and determines that the model requires a signal-process that conforms to the filter prototype, shown as **F**. The CM selects and uses the **filter proxy**, to realise this prototype.

The selection of a proxy is polymorphic since different proxies can be chosen to vary the implementation. The only constraint on selecting an implementation is that it targets one of the terminal's processors. The mechanism to guarantee this requires that the chosen proxy must successfully couple with one of the HW managers in the architecture model. For example **filter proxy** couples to **DSP manager**. Thus each proxy process is considered to 'run' on a processor manager. Similarly each channel proxy can be considered to 'run' on one or more link managers, where the process implementations imply inter-processor communication is necessary. This is essential to correctly model the latencies inherent in passing data across a link.

The joint model is a virtual implementation of the baseband and it is this that is analysed during optimisation. Remember that proxy components carry performance measurements for the accelerator component they represent. The CM pairs this information with details of real-time constraints conveyed by the algorithm map, and subsequently stored in the map's runtime representation, the algorithm model. Thus sufficient information is available to the CM to evaluate the real-time capability of the implementation specified by the current model. Other evaluations, such as those for power consumption and memory budget can be similarly made, although it is likely that any constraints on these would be specific to the terminal and its current configuration.

Optimising a single permutation of the model, i.e. unique selection of proxy components, is in itself a challenge. In fact the implied mapping of processes to shared HW resources requires solution of the well-known multiprocessor scheduling problem, known to be NP-complete [13]. We will not discuss this problem here. Instead we note that practical methods for obtaining near optimal solutions exist given a suitable MoC. The MoC that we have adopted for signal-processing algorithms is Synchronous Data-Flow (SDF). This sub-class of the well-known data-flow model, where process computation is triggered by the arrival of input data using asynchronous buffers, was first explored by Lee and Messerschmitt [13]. A correctly designed SDF model can use multiple sample rates, is determinate, has bounded memory requirements and static periodic schedules can be constructed for both uni-processor and multi-processor environments. If the algorithm map, and hence algorithm model are constrained to SDF, the CM can use the model to predetermine both memory usage and a process schedule. This is an efficient approach requiring minimal intervention during execution of the signal-processing.

Using SDF also has a beneficial side-effect; the combined function and architecture model can be executed in the control domain, providing a ready-made mechanism for controlling the execution of accelerated signal-processes. Accordingly proxies become active when they receive all the tokens representing memory references for input data, and a single control token from their HW manager. Managers issue their control tokens according to the predefined schedule. The signal-processing proxies can then initialise, start, monitor and stop their accelerated processes, finally returning the control token to their manager. Channel proxies can likewise dynamically manage both the memory, i.e. the memory references stored in the tokens, and physical mechanisms associated with communication between the processes. Data-flow is an ideal model for the signal-processing. Aspects of the control domain execution requiring data-dependent choice are suited to a more general synchronous MoC, such as Communicating Sequential Processes [14]. An example would be intervention during an error handling scenario.

So-far we have only considered the optimisation of a single model. In general multiple models will require simultaneous optimisation; for example multiple permutations of proxies for a new application, or re-optimisation of an existing application in tandem with a new one. The CM can include these by making each model an extra dimension in the scheduling problem space. Thus for maximum speed and efficiency proxy components must initially be lightweight; their accelerated part only need be invoked after the parent model has been selected for service.

4.5 Framework

Finally, to ensure that the above solutions are used in a consistent manner, we propose capturing them in a standardised software framework. The framework provides a template for the SW infrastructure, including the all-important interfaces of the many abstractions. The framework will also include design patterns for domain-specific software mechanisms, for example those that define elements of the SDF model of computation.

We anticipate that the framework will be described in an implementation-independent manner using UML. Useable versions of the framework can be implemented, (also in UML,) by realising the interface specifications using libraries imported from specific OO languages such as C++ or Java.

 Copyright © 2003 SDR Forum. All Rights Reserved

5. CONCLUSION

We have discussed a software-centric solution to the challenges facing the implementation of dynamically reconfigurable wireless baseband systems, assuming the most generic scenario, one where it is the software that is standardised and the underlying hardware is free to vary from terminal to terminal. We have proposed a standard software framework that meets the challenges of terminal diversity, heterogeneous processing resources, power minimisation, hard real-time constraints and automated application installation. The framework manages the complexity through the use of software abstraction. In particular the abstractions process, software component and model are crucial. Processes are used to define abstract baseband algorithms. Software components are used to implement these in a flexible manner such that they scale to multiple processor types. Finally modelling is used to analyse and optimise potential component combinations, according to known constraints, and ultimately to manage the execution of the resulting signal-processing.

Much future work is required to validate the ideas outlined in this paper and there are many unresolved issues. In the short term we intend to develop the concept focusing on methods and technologies for implementing the framework. In the longer term we aim to answer the wider question of whether an implementation of the concept is sufficiently practical and efficient to make software-centric reconfiguration and control of baseband signal-processing in a diverse, heterogeneous environment a reality.

The following are some of the important implementation issues that will require in-depth investigation:

- Realistic methods for scheduling the SDF signal-processes in a heterogeneous multi-processor environment.
- A lightweight runtime environment for d-ex-components suited to embedded systems. Current environments, such as the Corba Component Model, are considered too heavyweight [15].
- Classification and registration of hardware devices, resulting in components for the architecture model.
- Classification and identification of, and mechanisms for retrieval of, the many types of software components.
- Procedures and methods for embedding performance metrics into proxy components.
- Implementation of the algorithm map.
- Detailed study of the CM elements required to manage and control the reconfiguration procedure.

6. ACKNOWLEDGEMENTS

This work has been performed in the framework of the IST project IST-2001-34091 SCOUT, which is partly funded by the European Union. The authors would like to acknowledge the contributions of their colleagues from Siemens AG, France Télécom – R&D, Centre Suisse d'Electronique et de Microtechnique S.A., King's College London, Motorola SA, Panasonic European Laboratories GmbH, Regulierungsbehörde für Telekommunikation und Post, Telefonica Investigacion Y Desarrollo S.A. Unipersonal, Toshiba Research Europe Ltd., TTI Norte S.L., University of Bristol, University of Southampton, University of Portsmouth, Siemens ICN S.p.A., 3G.com Technologies Ltd., Motorola Ltd.

7. REFERENCES

[1] J. Mitola III, *Software Radio Architecture*, John Wiley & Sons Inc., New York, 2000.

[2] Z. Salcic, C.F. Mecklenbrauker, "Software Radio - Architectural Requirements, Research and Development Challenges," 8th International Conference on Communication Systems, Vol.2, pp 711 -716, 2002.

[3] H. Blume, H. Hübert, H. T. Feldkämper & T. G. Noll, "Model-based Exploration of the Design Space for Heterogeneous Systems on Chip," Proceedings of the Workshop on Heterogeneous Reconfigurable Systems on Chip, Hamburg, 2002

[4] G. Booch, J. Rumbaugh & I. Jacobson, The *Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[5] R. Hartenstein, "A Decade of Reconfigurable Computing: a Visionary Retrospective," International Conference on Design Automation and Testing in Europe, Munich, Mar. 2001.

[6] T. Makimoto, K. Eguchi, M. Yoneyama, "The cooler the better: new directions in the nomadic age," IEEE Computer Vol. 34, No. 4, Apr. 2001, pp 38-42.

[7] B.P. Douglass, *Real-Time UML, Second Edition, Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 2000

[8] D. Cook, "Evolution of Programming Languages and Why a Language is Not Enough to Solve Our Problems," Crosstalk, the Journal of Defense Software Engineering, Dec. 1999.

[9] A. Burns & A. Wellings, *Real-Time Systems and Programming Languages*, Addison-Wesley, 2001.

[10] E. A. Lee, "Embedded Software," Advances in Computers, Vol. 56, Academic Press, Sept. 2002.

[11] A. Pasetti, "Software Frameworks and Embedded Control Systems," Lecture Notes in Computer Science, Vol. 2231, 2001.

[12] E. Gamma, R. Helm, R. Johnson & J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[13] E.A. Lee & D.G. Messerschmitt, "Synchronous Data Flow," Proceedings of the IEEE, Vol. 75, No. 9, Sept. 1987.

[14] S. Schneider, *Concurrent and Real-time Systems, the CSP Approach,* John Wiley & Sons Ltd, Chichester, UK, 2000.

[15] Object Management Group, "Lightweight CCM, Request for Proposal," OMG Document: realtime/2002-11-27.

Proceeding of the SDR 03 Technical Conference and Product Exposition. Copyright © 2003 SDR Forum. All Rights Reserved