ARCHITECTING SOFTWARE RADIO

Ari Ahtiainen (Nokia Research Center, Helsinki, Finland; ari.p.ahtiainen@nokia.com); Heikki Berg (Nokia Research Center, Helsinki, Finland; heikki.berg@nokia.com); Ulf Lücking (Nokia Research Center, Helsinki, Finland; ulf.lucking@nokia.com); Aarno Pärssinen (Nokia Research Center, Helsinki, Finland; aarno.parssinen@nokia.com); Jan Westmeijer (Nokia Research Center, Helsinki, Finland; jan.westmeijer@nokia.com);

ABSTRACT

Applying design principles and methodologies constituted in the software domain and being adapted to the complete execution environment provides new perspectives for future multi-radio computers. The overall system architecture will allow hardware/software repartitioning and different hardware variants depending on prior defined requirements without extensive software rewrites. This demanding target can be addressed from two directions.

Firstly, as implementation technology advances it has to be possible to move services, or functionality, previously implemented in hardware to software or vice versa. Secondly, due to cost, power consumption, time-to-market or other customer needs the architecture has to support the creation of also hardware variants which still conform to the prior agreed system specification. This paper will present the utilized concepts and corresponding benefits to constitute the proposed architecture and platform for future radio computers.

1. INTRODUCTION

As compared to conventional software design methods, which have concentrated on documentation of system structures and HW/SW partitioning, the proposed formal design method created by Nokia systematically combines the methodologies of object-orientation and functional decomposition to model reactive systems. Key concepts comprise also the service specification model and system distribution. Active objects interact with their environment by means of abstract service primitives, which are visible at the Provided Service Access Point (PSAP), whereas the correct ordering of messages is specified by a PSAP state automaton. Utilizing PSAP state automata allows the SDR system architecture to contain the system structure and at the same time its behavior by specifying the services provided by the system components.

The outcome is a functional specification where the behavioral model supplies accurate rules on how to correctly use the architecture by defining the order of function calls and signals together with any constraints on their parameter values. Designing the architecture itself captures only externally observable behavior of the service – it neither represents any reference implementation nor dictates any implementation method for the realization of the service, but by including the state automata of the services it is possible to create executable models already at the architecture design phase.

Another key aspect in architecting of radio computer device is to specify the generic behavior which every radio access system in the SDR device must fulfill in order to become subject of software based control. Such software control mechanisms include, but are not limited to, network/device discovery, radio reconfiguration, flow control and scheduling of multiple simultaneously active radios. In our approach we don't require, but do allow also such option as joint radio resource management. Therefore any radio computer fulfilling these architectural requirements can be seen as an autonomous device participating communications in multiple networks simultaneously.

2. THE LYRA DESIGN METHODOLOGY

As compared to conventional software design methods, which have concentrated on documentation of system structures and HW/SW partitioning, the Lyra design method [1] created by Nokia utilizes the full power of modeling with scientific foundation. It should be emphasized that applying the formal design methods of Lyra to a system like a radio computer is the cutting-edge approach for future designs.

Lyra systematically combines the methodologies of object-orientation and functional decomposition approaches to model reactive systems. One key concept is the service specification model comprising of active objects which interact with their environment by means of abstract service primitives. The primitives are visible at the Provided Service Access Point (PSAP), whereas the correct ordering of messages is specified by a PSAP state automaton.

Utilizing PSAP state automata allows our SDR system architecture to contain the system structure and at the same time its behavior by specifying the services provided by the SDR system components.

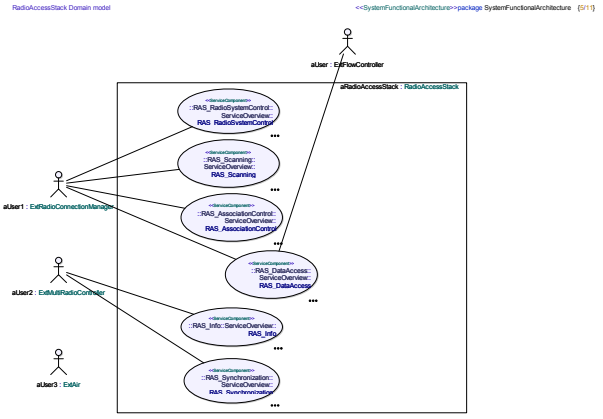

Figure 1 Radio Access Stack - Domain Model

The outcome is a functional specification, where the interface behavior model supplies accurate rules on how to correctly use the interface by defining the order of function calls and signals together with any constraints on their parameter values. Starting point for specifying the system is a domain model, capturing the relationships between

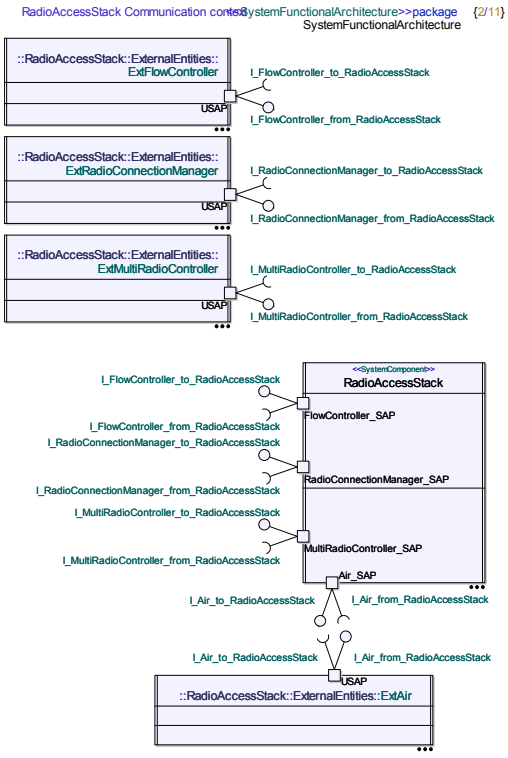

Figure 2 Radio Access Stack - Communication Context

services of service components and their external users. A detail of the domain model with its logical interfaces for our SDR Radio Access Stack (RAS) is shown in Figure 1. The purpose of a domain model is to identify the the logical interfaces, which become PSAPs. A domain model represents the services provided by the system component together with all different types of external users. Domain model is an informal description used for drafting. When using UML domain model is presented with a use case diagram.

Based on the information presented in the domain model, active classes for users and service components can be defined. Each logical interface, respectively a PSAP, is now represented as a port, enabling the communication between entities. Corresponding diagrams for the communication context of the RAS are shown in Figure 2. The active classes for service users have Used Service Access Points (USAPs), which represent the points of

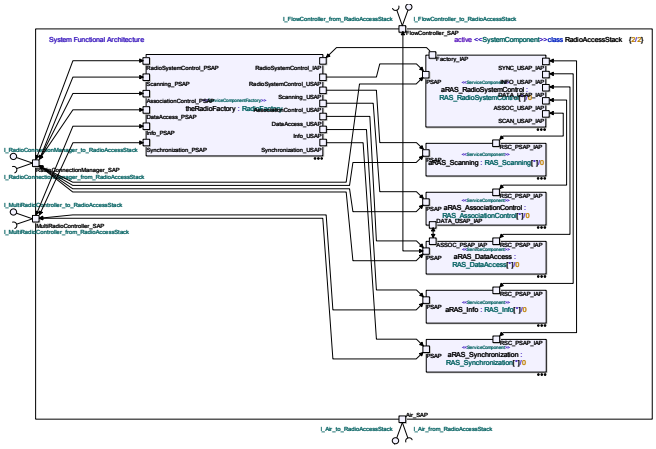

Figure 3 Radio Access Stack - System Architecture

communication between the users and the service components.

After this the design continues by specifying the behavior related to communication between each service component and its users. This behavior is represented as PSAP state automaton, which establishes a well-defined communication protocol between the users and services.

Finally all service components are collected into System Functional Specification model, which serves as highest level of system architecture. Functional Specification model for the RAS system is shown in Figure 3. Thanks to the state automata used to specify the behavior of service components the architecture model is fully executable already at this stage.

Designing the architecture itself captures only externally observable behavior of the service – it neither

represents any reference implementation nor dictates any implementation method for the realization of the service, but by including the state automata of the services it is possible to create executable models already at the architecture design phase. The ability to execute the model contributes to the many benefits of architecture modeling:

- Communication of the design by visualization
- Analysis and exploration of architectural design options and performance aspects
- Code generation by model compilers
- Verification by using model-checking and theorem proving methods
- Model-based testing allowing on-the-fly verification and validation of implementations

These benefits can only be realized by using a well-defined meta-modeling framework to support the selected modeling language. In case of UML2 such framework is ensured by the UML2 language model (i.e. a meta-model) and its implementation as a uniform model repository. Models created by an architect are stored into such repository in their UML language structure format, which makes it possible to traverse models for the above listed transformation purposes.

3. UNIFIED MODEL FOR RADIO SYSTEMS

For transparently attaching our SDR architecture (Figure 4) to higher layer networking protocols, it is necessary to generalize the behavior of the underlying radio system and specific states.

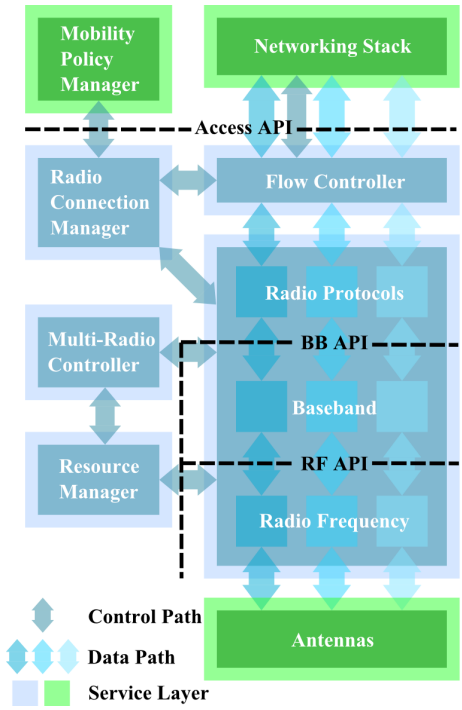

Figure 4 Overall SDR Architecture

According to such unified radio system model each radio system has to provide three sets of services.

- User data services
- Radio configuration services
- Multi-radio scheduling services

User data services include both control services for managing user traffic flows, as well as the actual radio data transfer (uni- or bi-directional). These services are visible at the Multi-radio Access API, where data coming/going to the network layer is defined as a flow. The Flow Controller component then routes data of this flow from/to a radio connection that has been selected by the Radio Connection Manager.

The Multi-Radio Access API is able to multiplex several flows to use a single radio connection. Several different networking entities can also use same radio connection. One access system can also provide more than one radio connection. Note, that in physical layer these multiple radio connections might share the same physical radio resources. In order to support media independent handovers the Radio Connection Manager is capable to move any flow from one radio connection to another. Configuration services are used to set up and reconfigure the set of active radios by managing the involved components. Services for multi-radio scheduling guarantees that the various radio systems follow the rule sets to avoid radio interference in the time domain, when simultaneously active. Each radio system has its local timing scheme, which has to be mapped onto a unified time basis. Scheduling is then expressed by using this unified time.

4. LAYERING AND ABSTRACTING EXECUTION ENVIRONMENT SERVICES

Software portability from one SDR platform to another is an important aspect which is not yet completely solved. As the development of Real Time Operating Systems (RTOS) and Remote Procedure Call (RPC) mechanisms for distributed embedded devices is still strong, the industry can not fix to only certain RTOS interface, such as POSIX, or to specific RPC mechanism, such as CORBA. Naturally in some domains, such as military SDR, interface compliance is necessary in order to achieve customer requirements. However, portability can be achieved also using well known design patterns, instead of fixing to a single implementation. In the following paragraph some of the main design patterns, to be utilized in our SDR platform are highlighted.

A good design pattern for hiding the creation of basic RTOS services is AbstractFactory [2] (see Figure 5). The concrete factory of course, has to be implemented for each RTOS separately and requires some manual work, but the main goal of RTOS portability is achieved.

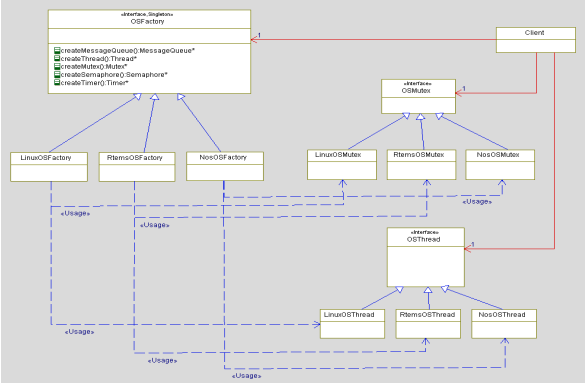

Figure 5 Abstract Factory Pattern

As the SDR radio platform becomes more and more distributed, where different parts of the radio protocol implementation have their dedicated control processors, a lightweight system wide RPC and Inter Process Communication (IPC) mechanism is needed. For serving this purpose there are multiple commercial and open alternatives such as DDS [3], CORBA/e [4], LINX from ENEA [5] or TIPC [6] as well as many other designs available. Ideally you would like to compare different alternatives and plug-in suitable RPC mechanism depending on your platform.

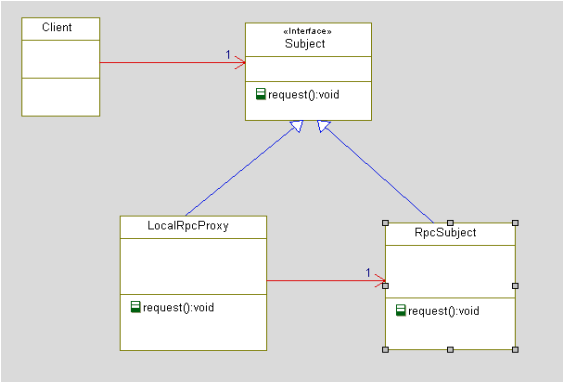

Figure 6 Proxy Pattern for hiding RPC Mechanism

Fortunately RPC mechanisms can be hidden behind a local proxy quite easily. The basic design pattern is depicted in Figure 6 and Figure 7.

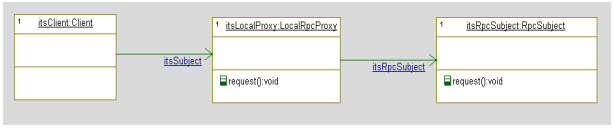

Figure 7 RPC Proxy Objects

Here the client uses the interface of Subject for issuing the request() from LocalRpcProxy instead of issuing the request directly using RPC mechanism. Proxies allow to write portable applications, by providing the possibility to select between multiple RPC mechanisms, even in runtime, by using the factory method-pattern.

The most critical part of software portability is probably to implement the physical layer signal processing across different platforms. Numerically intensive algorithms are still implemented using dedicated ASICs or software programmable signal processors with specialized instruction sets. From a SW programmer perspective they are treated as devices which of course require specific drivers. In order to achieve portability these physical devices need to be available in the alternative platform.

It can be assumed that this kind of algorithm acceleration (or parallelization) cannot be avoided due to the demanding timing restrictions specified in the upcoming radio standards. This trend shall continue also in the future at least in small size, power sensitive portable devices. Simultaneous usage of these accelerators needs to be optimized by a scheduler to effectively execute required computational tasks. In order to strive for application SW portability the interfaces towards these dedicated accelerators need to be well designed and maintained. Generalizing the interfaces towards these accelerators in order to satisfy the requirements of multiple radio standards is therefore a major task and needs a stable programming

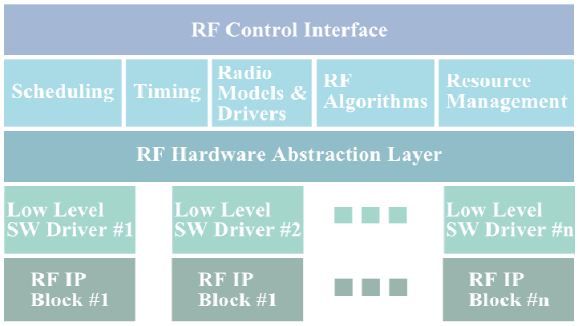

Figure 8 RF Control Software

model over several HW generations. This is mandatory to achieve SW radio also in the physical layer. As an example the involved layers of RF control software is shortly described in the next paragraph.

The control software, attached to generic hardware drivers is based on the time concept for multi-radio and will implement amongst other state functionality, the scheduler, resource manager and calibration manager. The basic concept of the "self-contained" RF control software architecture is presented in Figure 8.

5. MAPPING TO PLATFORM SPECIFIC IMPLEMENTATION

The executable specification describes the complete set of reconfiguration services and signaling, as well as the generalized interfaces for radio access. It also provides the means to observe the local conditions of the device and enabling the system to reason about its own structural and behavioral features. To support also future cognitive radios, mechanisms for decision making and policy enforcement have to be considered to enable autonomous adaptation and cross layer optimization bound to the environmental context. Reliability will be increased by fault and recovery management which will take care of falling back to the last known state to keep the device operational. The executable specification should also model the design patterns needed for mapping the services to different platforms.

An implementation of such framework will incorporate a combination of the event-driven and publisher/subscriber principles to launch distributed tasks on demand. It abstracts different transport mechanisms for inter-processor and inter-process communications in a heterogeneous multiprocessor environment. Depending on the platform environment, in this context Symmetric Multi-Processing (SMP) or Asymmetric Multi-processing (AMP), the framework on top shall be configured at compile time to use the most suitable IPC mechanism. The dedicated reconfiguration mechanisms of the signal path need to address the different kinds of device types, as there are general purpose processors including multi-core processors, specialized processors (like vector processors) and configurable hardware accelerators (including customized digital logic for high speed signal processing and RF IP blocks up to antenna).

6. CONCLUSION

The presented structured and modular architectural concepts for a radio computer is a direct response to the evolution of communication standards, software development, computer architecture, circuit design and the advances in semiconductor technology.

Mastering the complexity can be only achieved by appropriately raising the level of abstraction and using formal methodologies for specifying hardware and software components. In that sense a holistic view on radio modem architecture and platform is promoted, taking into account every aspect to successfully fulfill the requirements of future standards or disruptive technologies like cognitive radio. Each of the facets of the protocol, baseband and radio frequency domain requires further studies including architecture validation and implementation of selected platform components. Maybe the biggest challenge is to keep hardware and software development aligned, but still decoupled as much as possible.

7. REFERENCES

[1] Sari Leppänen, "The Lyra Design Method", Tampere University of Technology, Finland, 2005.

[2] E. Gamma, R. Helm, R. Johnson and John Vlissides, *Design Patterns – Elements of Reusable Object Oriented Software*, Addison Wesley, New Jersey, USA, 1995.

[3] OMG Specification, *Data Distribution Service for Real-time Systems, Version 1.2*, 2007.

[4] OMG Final Adopted Specification, *CORBA for embedded Specification,* 2006.

[5] ENEA, *LINX Protocol - Document ver.10 for protocol 1.0,* 2006.

[6] Multicore Association, TIPC Working Group, *TIPC: Transparent Inter Process Communication Protocol,* 2006.

ARCHITECTING SOFTWARE RADIO

Ari Ahtiainen (Nokia Research Center, Helsinki, Finland; ari.p.ahtiainen@nokia.com); Heikki Berg (Nokia Research Center, Helsinki, Finland; heikki.berg@nokia.com); Ulf Lücking (Nokia Research Center, Helsinki, Finland; ulf.lucking@nokia.com); Aarno Pärssinen (Nokia Research Center, Helsinki, Finland; aarno.parssinen@nokia.com); Jan Westmeijer (Nokia Research Center, Helsinki, Finland; jan.westmeijer@nokia.com);