Design, Implementation and Validation of a generic and reconfigurable Protocol Stack Framework for mobile Terminals

Thorsten Schöler
Siemens AG
Information and Communication Mobile, Technology and Innovation,
Siemens ICM MP P TI10, PO Box 801707, 81617 Munich, Germany
thorsten.schoeler@siemens.com
schoeler@sra.uni-hannover.de

Christian Müller-Schloer
University of Hannover
Institute for Systems Engineering, System and Computer Architecture
Appelstraße 4, 30167 Hannover, Germany
cms@sra.uni-hannover.de

Abstract

This paper introduces a modular and reconfigurable software framework for protocol stacks implemented in platform independent manner. Simulation tools useful for software validations are introduced and a new distributed, three-staged procedure for validation of protocol stack software is proposed. Assertion-based virtual prototyping (based on non-resident assertions), utilising simulation of hardware software co-systems as well as software probes containing code-resident assertions are used in the proposed validation process.

1. Introduction

The trend for ubiquitous and pervasive computing has led to powerful mobile terminals with an enormous complexity of their hardware as well as their software implementations. Still more limited in resources than their desktop counterparts, mobile terminals give the user his well-known desktop application experience (web browsing, e-mail, media streaming, etc.) while being connected wirelessly. Mobile device manufactures now face the challenge, to support high and still growing software complexity with short update cycles on resource constrained (e.g. memory, battery power, etc.) mobile devices.

To deal with higher complexity, more and more applications are based on similar core functionalities. This accounts also for a growing part of system software. Capturing core functionalities in modular libraries is common knowledge and this idea will be consequently exploited much more in future mobile terminal platforms to tackle memory constraints.

Founding software on libraries generally speeds up software development, yielding short software update cycles, which in turn will introduce higher software security demands on future mobile terminals. Whilst security issues, raised by the introduction of third-party applications on terminals operating in a mobile communication network, are already tackled (see [1] and [2]), the introduction of third-party system software for mobile terminals still yields new security challenges. Because protocol stack software, as an integral part of mobile terminals' system software, need to get much more modular and even more flexible to satisfy future requirements, they will be in the focus of this paper.

This paper introduces a platform-independent software architecture for implementing protocol stacks for mobile terminals, which are modular and reconfigurable. Furthermore, a validation process will be introduced that minimises the risk of a terminal becoming a so-called rogue terminal[1].

The following sections of this paper will describe the design and implementation of a platform-independent, generic and flexible framework for protocol stack composition. Furthermore, security issues will be shown and a new way of validating composed protocol stack software behaviour by assertion-based virtual prototyping will be introduced.

[1] Rogue terminals will interfere with network operation and thus may be even able to interrupt a working communication network.

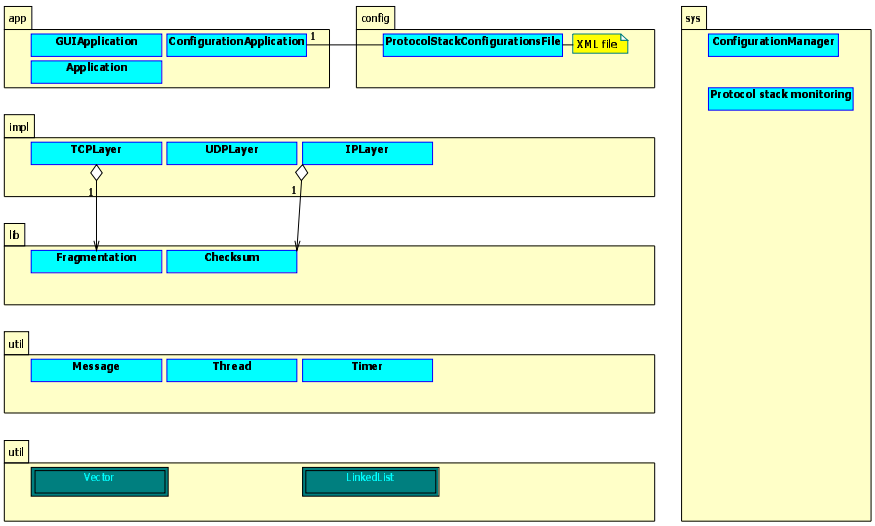

Figure 2. Abstraction levels of software architecture

2. Design

The proposed protocol stack software architecture, as seen in Figure 1, is based on the idea of composing protocol stack software in an object-oriented manner from a library of generic components and implementation specific software objects [13]. Further to being configurable at compile time, as in [5] and [10], the proposed protocol stack software can be configured and re-configured during runtime as in [7].

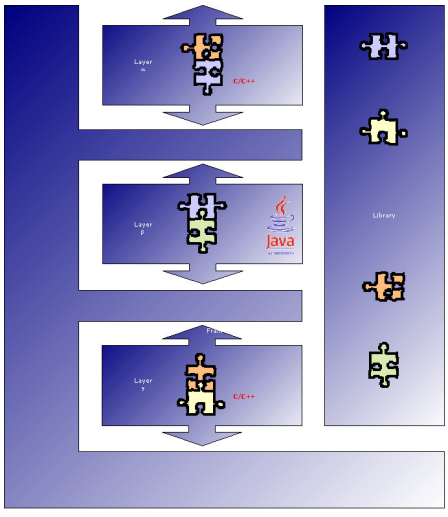

Figure 1. Overall architecture of protocol stack framework

The actual protocol stack software, which has been composed by the framework, is assisted by a software library, which supplies core functionalities (threads, timers, etc.) as well as the frameworks' system services for reconfiguration, monitoring, etc.

The distribution of library components, framework and other software components according to their abstraction level into various packages can be seen in Figure 2.

One major component of the framework is the central configuration manager. The configuration manager decides, which protocol stack configuration will be instantiated and used to fulfil the application's communication requirements. The framework uses XML files for storing and exchanging protocol stack descriptions and configurations.

Applications connect to the protocol stack using a socket style interface, similar to the interface described in [14]. They express their communication requirements by supplying the protocol stack configuration manager with their Quality of Service requirements for the requested socket connections.

2.1. Communication model

Protocol stack implementation theory describes two basic process models for implementation of protocol stack entities (e.g. layers) and their communication [11].

In the *thread-per-layer* approach, each layer is implemented by a separate thread or process. To

communicate between layers, messages have to be passed between different threads or even processes. Communication between protocol stack entities will afford a time-consuming context switch between the communicating threads and thus, will slow down the message passing process.

On the contrary, the *thread-per-message* model treats each layer as a more or less static piece of software. A dedicated thread is used to process each message passed between protocol stack layers. For the execution of protocol stack layer actions, a dedicated pair of function calls (methods) is used. One method is responsible for handling messages, which move upwards (from networking hardware to application) in the protocol stack and another method handles messages going down in the protocol stack.

The proposed protocol stack framework uses the *thread-per-message model* for passing information up and down the stack. The message-passing model can be seen in Figure 3.

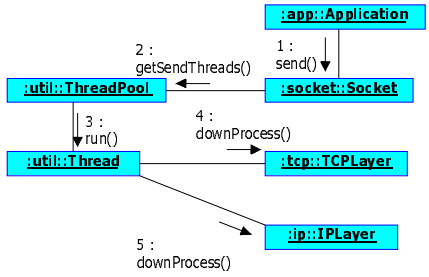

Figure 3. Thread per message model used in protocol stack communication

In step one, an application sends a message via the socket to the protocol stack. The socket requests a new send thread from the thread pool in step two. The thread is started in step three and subsequently works down its way through the protocol stack layer entities in steps four and five.

2.2. Protocol stack configuration

As mentioned before, the configuration manager is a central component of the protocol stack framework. It is responsible for instantiating protocol stacks from given abstract protocol stack configuration descriptions (stored in a database or files, locally or in the network).

Protocol stack configuration descriptions (protocol stack graphs) are stored in the framework in XML files. The configuration manager contains a parser to generate protocol graphs from abstract protocol stack descriptions used inside the framework.

A protocol graph is basically a map of a protocol stack instance, contained in each protocol stack thread, which tells the thread where to go next to process its data according to the protocol stack specification. It determines, where (in which stack component or layer) a particular message being processed up and down the stack by the thread, needs to be dealt with next.

In case of a necessary or requested protocol stack reconfiguration, the execution of threads will be stopped by the framework's thread pool. Subsequently, the framework's configuration manager replaces protocol stack entities, which need to be replaced, and protocol stack graphs will be updated accordingly, reflecting the new protocol stack configuration. Finally, the execution of protocol stack threads will be resumed.

Because protocol stack layers are more or less static pieces of software, little state information will be stored inside the protocol stack entities (i.e. data used for dynamic protocol stack optimisation during run-time). To enable smooth updates of those less-static entities, a mechanism must be found to store entity state information in a way, that future implementations are able to utilise such information for a smooth transition.

3. Implementation alternatives

A major requirement imposed onto the protocol stack framework, is to include legacy protocol stack software (implemented in a native programming language, such as C/C++). To accomplish that, a message-passing interface, based on the Java native interface (JNI), is integrated into the framework, to enable integration of native protocol stack software implementations (see also Figure 1).

It is envisaged to connect the protocol stack software to the OS networking system by a packet capture device (such as WinPCap [12]) to enable real-life applications to be built upon the protocol stack framework to communicate with remote network nodes.

Furthermore, it is to combine this proposed architecture with concepts and implementations from [3], which is in contrast to this proposal, based on a thread-per-layer-model. The thread-per-layer-model, in contrast to the used thread-per-message-model, has performance advantages in protocol stack configurations, where many transitions between Java

and native implementations layer instances have to be crossed. As seen in Figure 4, the JNI has to be crossed four times in the proposed framework implementation for a protocol stack configuration consisting of two layers implemented in C/C++ and one layer implemented in Java.

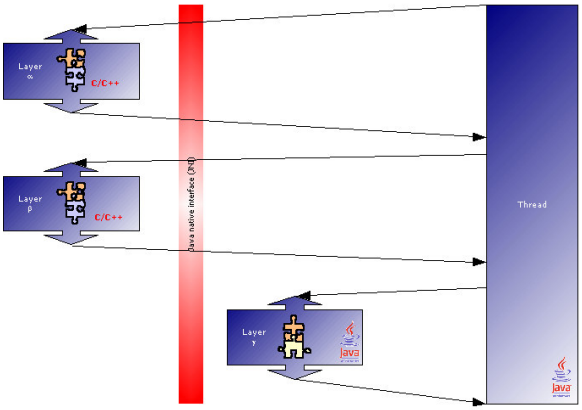

Figure 4: Thread per message model (JNI)

If upper and lower termination of the protocol stack has to be implemented in Java, the situation for the thread-per-process model is as seen in Figure 5.

The JNI boundary has to be crossed only twice to pass a message through the same protocol stack configuration compared to Figure 4. This means a speed-up of factor two, when the JNI boundary implementation in both architectures is estimated as equally complex.

The combination of those two architectures will help to minimise JNI boundary crossings by using the optimal thread model for a given configuration and will yield a sophisticated framework for the implementation of many protocol stacks.

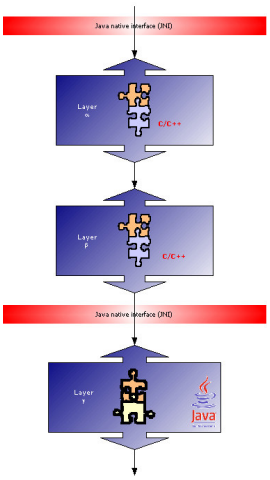

Figure 5: Layer per message model (JNI)

4. System security

According to [8] there are many ways of improving systems software security. Described concepts range from piracy prevention based on establishing server connections or using license files. Furthermore, hardware based privacy protection with checksums or cryptographic methods, watermarking and fingerprinting of documents or software and even security concepts proposed by the Trusted Computing Group (TCG formerly known as TCPA[2]) are used in contemporary software systems.

The proposed security concept follows the mentioned guards and assertion checking approaches in software implementations (discussed later as assertion-based virtual prototype and software probes in Section Assertion based virtual prototype) and introduces *system validation by simulation* as new system security measure.

Using the virtual prototype and system simulation approach, those probes may reside in productive protocol stack software (resident probes) or they may only be used in the virtual prototype and be removed in the real mobile terminal productive software (non-resident probes). Non-resident probes will avoid the drawback of using assertions: Slow down of software execution.

The actual protocol stack validation process, based on system simulation and software probes containing assertions, is described in the following subsection.

[2] Trusted Computing Platform Alliance

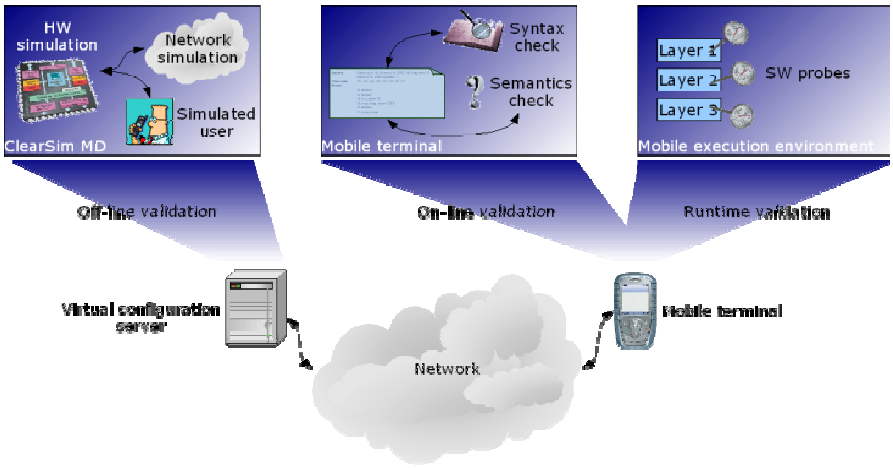

Figure 6. Three stages of validation

4.1. Protocol stack software validation

A three-staged process for protocol stack software validation as seen in Figure 6 is proposed: Network-based off-line validation, On-line validation and Runtime validation (both terminal based).

Off-line validation uses extensive system simulation for validation of the collaboration of the mobile terminal hardware and the installable protocol stack software.

For that, a virtual prototype of the mobile terminal is designed (see System simulation section). It contains modules of application processing hardware (i.e. ARM processor model), specific terminal MODEM hardware (i.e. base band and RF processing, DSPs, etc.) and other assisting modules, which will be simulated using functional and timing accurate system simulation. The terminal system simulation is connected to interfacing simulation modules, which simulate communication network and terminal user behaviour, stimulated by test vectors modelled after real-life behaviour.

Due to computing resource constraints and security issues, terminal-based *on-line validation* cannot fall back on extensive simulations and is limited to simpler checks. The terminal validates the to-be-installed protocol stack configuration according to pre-defined rules and tries to identify suspicious software configurations. These rules may contain plausibility checks, syntactic and semantic software configuration checks.

After the protocol stack software configuration has been validated and no suspicious configurations have been identified, all necessary software modules will be downloaded. The terminal now validates the actual protocol stack software modules according to its validation rules. Subsequently, supposed that all checks passed without validation violations, the downloaded protocol stack software will be executed on the terminal.

During execution, the mobile terminal is able to supervise protocol stack behaviour with code-resident assertions and checkpoints. Such assertions may check for valid content of communication messages as well as for compliance to certain threshold conditions. This is called *run-time validation.*

Following this brief overview of the proposed validation scheme, used validation technologies will be described in the following sections.

4.2. System simulation

Nowadays, many computer-aided tools assist the design and implementation of complex hardware software co-systems. A modern way of prototyping system behaviour is by using a virtual prototype. A virtual prototype shows the same functional behaviour and timing as a real hardware-prototype, but is completely simulated by a simulation tool. A virtual prototype contains, besides the simulation of the target system, a simulation of the system's environment.

A simulation tool, which allows functional and timing-accurate system simulation, is *ClearSim MD.* ClearSim MD is a system simulator, developed at the

Institute System Engineering, System and Computer Architecture at University of Hannover. It is able to integrate system simulations from *multiple domains* [6] and on various abstraction levels into a combined simulation model. Heterogeneous systems would be modelled in their suitable domain specific languages, such as UML, SDL, C/C++, Modellica and EFSM (Extended Finite State Machines). Maximum flexibility is reached by using UPSI (Unified Portable Simulation Interface), which enables easy integration of other system simulation tools.

Current developments in ClearSim MD extend the timing-accurate simulation model with the ability to insert assertions into the model. Assertions can be used to supervise and control simulation state and thus, help in validating system behaviour.

4.3. Assertion based virtual prototype

A virtual prototype, as introduced before, will be enriched with assertions (specifying parameter ranges, timing behaviour, etc.) in order to obtain an abstract system model, a so-called *AViP* (Assertion-based Virtual Prototype). Inside the AViP system model, the actual protocol stack software, which is later downloaded to the terminal, is executed. By executing the assertion-enriched system simulation, more information about the functional and timing behaviour of the complete hardware-software-co-system, especially the to-be-validated protocol stack software, can be gained by exploiting assertion data. This information will be used to get a detailed picture of the system's behaviour [9] and detect non-valid or rogue terminal behaviour.

Assertions can be differentiated into *simulation-only (non-resident) assertions*, which are utilised for validation only in the system simulation stage and are not present in actual productive software, and so called *code-resident assertions*, which remain in productive software. Code-resident assertions can be used to gain run-time diagnostic information during software run-time for validating protocol stack software behaviour. This is used in the so-called *software probes*.

Those small program units will be integrated in the protocol stack software by the framework. The to-be-installed protocol stack software cannot avoid probe insertion, because the terminal resident and tamper-proofed protocol stack framework triggers it. Software probes will secure the correctness of software execution in a distributed but networked manner by supervising for example communication messages content and check for compliance to certain threshold values, such as messages sizes, repetition timings, etc.

For that, they will be inserted at distinct places in the modular protocol stack software, for example between each protocol stack layer at module boundaries or interfaces. An inserted software probe for example, could be used to validate the actual content of layer-to-layer communication messages. It may check for correctness of message length and / or check message header or footer for protocol specification accordance.

The software probes are modules designed against the same data communication interface as the protocol stack modules (e.g. layers or library modules). They implement the method pair upProcess() for messages going up the protocol stack and downProcess() for message going down in the protocol stack. This enables the framework to include them virtually unrestricted at any place in a particular protocol stack, which needs to be validated during run-time.

5. Conclusion

A software framework for dynamic composition of protocol stack software has been introduced. The trust in to-be-installed protocol stack software can be increased by validation of such software using system simulation and by inserting software probes containing assertions into the protocol stack software. A system simulator with the ability to exploit various kinds of assertions has been described and a particular validation process has been depicted.

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, DoCoMo Communications Laboratories Europe GmbH [4].

7. References

[1] 3rd generation partnership project (3GPP): Mobile execution environment (MExE), Service description. *Technical report stage 1, release 4, TS 22.057 V4.0.0*. October 2000.

[2] 3rd generation partnership project (3GPP): Mobile execution environment (MExE), Functional description. *Technical report stage 2, release 4, TS 23.057 V4.1.0*. March 2001.

[3] Farnham T., Schöler, T.: Flexible protocol stack framework: Design, validation and performance. *Proceedings of 2003 Software Defined Radio Technical Conference and Product Exposition*. SDR Forum. November 2003.

[4] Georganopoulos, N., Farnham, T., Schöler, T., Burgess, R., Warr, P., Golubicic, Z., and Sessler J.: Terminal-centric view of software reconfigurable system architecture and enabling components and technologies. *IEEE communications magazine*, 2003.

[5] Hutchinson, N. C. and Peterson, L. L.: The X-Kernel: An architecture for implementing network protocols. *IEEE transactions on software engineering*. 17(1):64-76. 1991

[6] Krisp, H., Bruns, J., Eilers, S., and Müller-Schloer, C.: Multi-domain simulation for the incremental design of heterogeneous systems. *ESM 2001 conference proceedings*. pp. 318-323. June 2001.

[7] Moessner K, Vahid S, Tafazolli R.: Terminal reconfiguration: The OPtIMA Framework. *Second International Conference on 3G Mobile Communication Technologies, IEE-3G2001*, pp. 241-246, London, United Kingdom, 26-28 March 2001.

[8] Naumovic, G. and Memon, N.: Preventing piracy, reverse engineering, and tampering. *IEEE computer magazine*. Volume 36, Issue 7. pp. 64-71. July 2003.

[9] Oodes, T. and Müller-Schloer, C.: UML-basierter Systementwurf sicherheitskritischer, heterogener Systeme. *Proceedings of Simulationstechnik 16. Symposium*. pp. 365-370. Rostock. September 2002.

[10] O'Malley, S. and Peterson, L.: A dynamic network architecture. *ACM Transactions on Computer Systems*, Vol. 10, No. 2, May 1992, pp. 110-143.

[11] Peterson, L. L., Davie, B. S., Bavier, A.C.: X-Kernel Tutorial. http://www.cs.arizona.edu/classes/cs525/tutorial/tutorial.html. January 1996

[12] Risso, F. and Degioanni, L.: An architecture for high performance network analysis. *Proceedings of the 6th IEEE symposium on computer and communications (ISCC 2001)*. pp. 686-693, Hammamet, Tunisia. July 2001.

[13] Siebert, M. and Walke, B.: Design of generic and adaptive protocol software (DGAPS). *Proceedings of the Third Generation Wireless and Beyond (3Gwireless '01)*. San Francisco, US. June 2001.

[14] Stevens, W. R.: *Programmieren von UNIX-Netzwerken, Netzwerk-APIs: Sockets und XTI*. Hanser. Second edition. 1998.