ANA Project
Autonomic Network Architecture

Sixth Framework Programme
Priority FP6-2004-IST-4
Situated and Autonomic Communications (SAC)

Project Number: FP6-IST-27489

Deliverable D.1.8c

Software development process for the ANA Core implementation

ANA Project

Autonomic Network Architecture

Project Number	FP6-IST-27489
Project Name	ANA - Autonomic Network Architecture
Document Number	FP6-IST-27489/WP1/D.1.8c
Document Title	Software development process for the ANA Core implementation
Workpackage	WP1
Editor	Ghazi Bouabene (UBasel)
Authors	Ghazi Bouabene (UBasel) Christophe Jelger (UBasel)
Reviewers	Vera Goebel (UiO)
Dissemination level	Public
Contractual delivery date	31^{st} December 2007
Delivery date	15^{th} February 2008
Version	Version 1.0

Abstract:

This document is a companion document of the software deliverable D.1.8. It summarizes how the development of the first ANA Core prototype was carried out in 2007 and recapitulates major development milestones and software changes. The goal is to provide a concise summary of the development activities of the ANA Core software during the year 2007.

Keywords:

ANA, development coordination, development milestones.

Table of content

1 Introduction

As described in the initial description of work, the ANA project is organised around two prototyping cycles which are used to test, validate, and refine the design of ANA over the course of the project. In other words, prototyping in ANA is a key research activity which is fully part of the design process. As such, the prototyping effort in ANA cannot follow standard software development models which assume that software is developed according to well defined requirements and specifications. In contrast, to support our design strategy we have organised our prototyping activitities according to a flexible software development process based on rapid prototyping, testing, and integration of feedback. In order to document this development process, the ANA consortium has decided to produce a short report complementing the software deliverable D.1.8.

We here remind the reader that the objective of Task 1.6 "ANA core implementation" during 2007 was to develop the first prototype of the ANA Core software. This prototype development is based on the first version of the ANA Blueprint [1] and its main outcome is Deliverable D.1.8 which actually consists of three parts:

- **D.1.8a** — The first prototype of the ANA Core software.
- **D.1.8b** — The documentation of the ANA Core software.
- **D.1.8c** — The description of the software development process.

The present document covers the part **'c'** of deliverable D.1.8. That is, it presents the development process of the ANA Core software but does not describe the software itself. For a detailed description of the software, we refer the reader to the "ANA Core documentation" (D.1.8b) which describes in details how the code is organised and how one can develop extra components with the ANA API.

2 Development process

2.1 Basic design

In 2007, the development of the ANA Core software has been an evolving process which has involved multiple refinements of some of the core ANA architectural concepts and multiple revisions of the API supported by the ANA Core. However, the software itself has been developed according to a stable design introduced in the ANA Blueprint and illustrated by figure 1. In this design somehow similar

to a micro-kernel, the ANA Node is composed of a minimal core called the MINMEX. The sole objective of the MINMEX is to support the low-level machinery of ANA which allows more advanced functionalities to interact and eventually form complex services. These advanced functionalities are called *bricks* and form the so-called ANA "Playground".

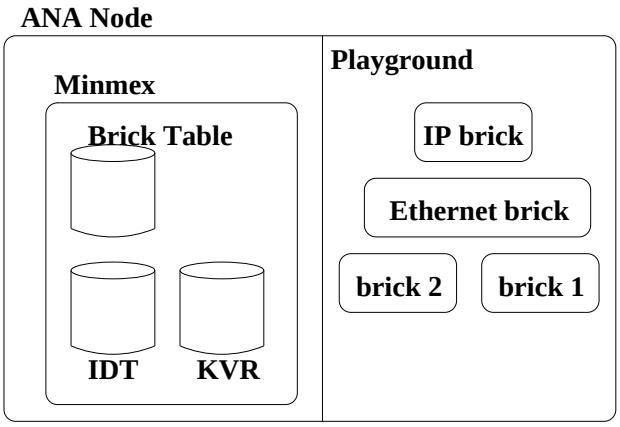

Figure 1: Basic design of the ANA Node.

Following this basic design, a small set of core development requirements were fixed in early 2007 and frozen until the second design phase starting in 2008. These core requirements are summarised as follows.

1. To provide a clear separation between the implementation of the MINMEX and the implementations of all the *bricks* of the "ANA Playground".
2. To provide a flexible system for inter-connecting the MINMEX and the bricks via various underlying mechanisms including Unix and UDP sockets, named pipes, and shared memory.
3. To ensure that all the components of the software can compile as standard userspace applications, Linux kernel modules, or standalone agents for the NS2 simulator software.

The objective of the first requirement was to develop a modular software where each development track can proceed as independently as possible. In a project with 11 partners it is indeed very difficult to coordinate all developments on a daily or weekly basis and it was hence essential to allow each brick development to remain independent. The goal of the second requirement was to be able to run a "distributed" ANA Node with different components possibly being executed on different computers. The idea was to fully decouple the notion of a "node" from physical constraints. Finally, the objective of the third requirement was to allow developers to focus on the development of their algorithms and protocols without having to bother about execution environments. In particular, we wanted that the same brick code could be tested in real systems but also in network simulation software.

2.2 Prototyping cycles

As explained in the initial description of work, the ANA project is actually organised around two prototyping cycles illustrated by Table 1. The objective was to develop a first prototype of ANA after the first half of the project in order to be able to refine and revise the architecture based on the experience gained during the prototype implementation. Like others [2, 3], we believe that this strategy of embedding the prototype implementation into the main research activities is a key for providing a viable (i.e. experimentally validated) architecture which will "grow" and mature over the course of the project. This contrasts with other projects where the implementation is often a final activity after which any architectural refinement is no longer possible.

2006		**2007**	
1st Design phase		1st Prototyping phase	
2008		**2009**	
Testing phase	2nd Design phase	2nd Prototyping phase	Final evaluation

Table 1: ANA Project lifetime.

Clearly, such a development strategy prevented us from following the traditional and somehow static "Waterfall" development model composed of subsequent phases like e.g., requirements, design, implementation, validation, and maintenance. In ANA, the prototyping process is indeed a research activity in itself which, during the first prototyping phase, is used to refine the design of ANA: reactivity and flexibility were hence key requirements of the development process. This has lead to following a development process known as *extreme programming*.

2.3 Extreme Programming

As briefly described earlier, the basic design of the ANA Node was described in the first version of the ANA Blueprint. However, while the first version of the Blueprint describes the high-level operation and architecture of ANA, it does not provide detailed implementation guidelines for the various communication mechanisms involved. ANA is indeed a FET (Future and Emerging Technology) research project with a strong experimentation profile which considers prototyping as a key research activity. In particular and as stated in the previous section, the design of ANA is organised around two prototyping cycles during which we have planned to refine, update, or even fully modify some parts of ANA while we gain more experience from the prototyping activities.

As a result and to reflect the design of ANA in the software enginering process, the development of the first prototype could only be done through incremental

complexity additions and constant testing and refinement of both the Blueprint concepts and their implementation. In other words, the prototyping effort is fully part of the architecture design and is by itself a research activity. In particular, designing a new network architecture also implies designing new programming models which mature in parallel to the high-level design. To support such a flexible and open design, we have decided to follow a development strategy known as *extreme programming* which we briefly summarize in the following section.

Extreme programming [4] is a strategy that promotes multiple development iterations throughout the life-cycle of a project. Paraphrasing one of its proponents, "extreme programming is designed for use with small teams who need to develop software quickly in an environment of rapidly-changing requirements". This actually fits perfectly with the design objectives and resources constraints of the ANA project.

As a brief introduction, some of the main practices of extreme programming are detailed below and linked to the development methodology followed in ANA.

- Cyclic planning process (also called "game"): depending on the importance and development cost of software components, the development team decides what should be developed first and what can be deferred until the next planning phase. In ANA, the core development team has met once a month and coordinated via emails in order to plan the upcoming development tasks.
- Small releases: meaning that the project should advance by iteratively fixing new close goals while quickly getting implementation feedback. In ANA, a very early working prototype was released already in June 2007 and was permanently updated and revised.
- Preferring simplicity : this means trying to design the simplest code units possible that fits to the unit's objective. In ANA, early versions of many components (KVR, QuickRep, vlink, API) were released very early and refined after initial experience and feedback.
- Test driven development: each unit of code must be validated through unit tests before being released. In ANA, each developer was responsible for testing his/her contributions before commiting changes to the current development branch.
- Leave optimization until the end: for each software component, the goal is to quickly produce a working system in order to quickly identify weaknesses, revise the design, and only optimize when the final design is reached. In ANA, the objective is to revise the current ANA core software and release a more mature software in mid-2008.
- Collective code ownership: all developers should have access and modification rights to every piece of code on the common code repository. In ANA, each developer is free to modify whatever code as long as this was previously discussed and agreed by the development team.

2.4 Development Process of the ANA core software

Not having detailed specifications at the begininning of the development process, we proceeded by incremental complexity addition, following an approach inspired by Extreme programming illustrated by figure 2.

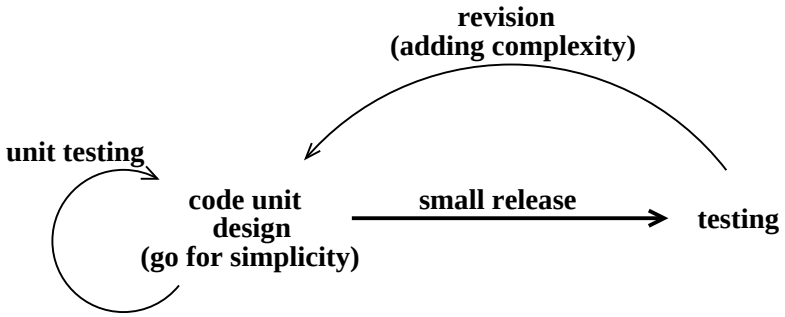

Figure 2: Development process

We first started by designing the most simple code units (MINMEX, bricks, API primitives) that fullfilled the initial (and currently needed and understood) goals. During this initial development phase, and before any release, these units were tested to fulfill their initial goals. The next step was then to release these code units to other coding partners. Note here that on the contrary of the generic Extreme programming approach, we do not have any customers using the ANA core software. However, since each code unit was meant to be used by other development partners, these partners could be seen as customers that provide necessary feedback to improve each unit.

This approach was most notably used for defining the generic compartment API functions as well as specifying their needed arguments. Note that while following the *Extreme programming* development strategy depicted above, the core ANA Blueprint concepts and communication paradigms were constantly refined (revisited) during the development process. In particular, we have not waited for the completion of a first prototype generation before starting to revisit some of the core design principles.

Finally, it is worth mentioning that major parts of the current ANA core software are becoming quite stable which indicates that we will soon be able to start optimizing the current code. In addition, the second coding workshop organized in January/February 2008 will be used as a major milestone to gather feedback and apply a last round of changes to the core software before we start the optimization process.

3 Developers team

In 2007, the ANA core software development team has been composed of 9 persons from 4 different project partners. Note that while each developer was responsible for some particular component(s) of the software, there was a very close collaboration between the developers taking care of the core elements (i.e., the MINMEX and the ANA API) in order to minimize the code required for running ANA in different environments (i.e., userspace, Linux kernel, NS2 simulator). The persons involved in 2007 and a brief summary of responsibilities are summarized below.

- Dr. Christophe Jelger (UBasel): in charge of the development of the vlink abstraction layer between the ANA software and the host's physical devices. Also in charge of the overall coordination of the development team.
- Dr. Sylvain Martin (ULg): in charge of the development of packet probing bricks and a virtual coordinate system compartment.
- Dr Franck Legendre (ETHZ): in charge of porting the ANA core software to the Network Simulator 2 platform.
- Ghazi Bouabene, PhD candidate (UBasel): in charge of the development of the ANA core software (MINMEX, API levels 0, 1 and 2) as user-space applications. Development of initial compartments: node compartment, Ethernet compartment.
- Ariane Keller, PhD candidate (ETHZ): in charge of the development of the ANA core software (MINMEX, API levels 0, 1) for linux kernel space. Development of monitoring bricks and code validation utility for the ANA software.
- Theus Hossmann, PhD candidate (ETHZ): in charge of porting the ANA core software (minmex, APIs) to the Network Simulator 2 platform.
- Lorenzo Peluso, PhD candidate (FOKUS): in charge of the development of the information flow framework and of monitoring bricks.
- Manolis Sifalakis, PhD candidate (ULanc): in charge of the development of the functional composition framework.
- Stephan Dudler, Master student (ETHZ): in charge of the development of an Internet Protocol (IP) Compartment.

4 Coordination between developers

The coordination between geographically distant ANA core developers was done with the help of a mailing list, regular meetings and a *Subversion* repository.

4.1 'ana-dev' mailing list

This list, hosted by the University of Basel was created on March 19^{th} 2007 and is managed by the University of Basel in conjunction with ETH Zurich. It is used as a media for general discussions, bug reports, announcements of major code changes, and for in-depth technical discussions about the code. The archive of discussions on this list can be accessed at *https://www.maillist.unibas.ch/mailman/private/ana-dev/* (the archive is currently private and access requires a username and password). The list currently has 22 users subscribed to it.

4.2 Regular (monthly) meetings

The list of the held meetings is detailed below. The main objective was to regularily report on developement status, identify and plan the next development activities as well as discuss technical details.

- March 14^{th} 2007 in Basel: kickoff meeting. First draft design of the ANA core software, identification, definition, and repartition of the first development tasks, assignment of responsible person to each task. First milestone is to have a first running basic prototype (with limited functionalities) by the end of June 2007.
- April 18^{th} 2007 in Zurich: technical discussions about the port of the userspace ANA core code into Linux kernel modules. Definition of a new structure for the Subversion repository.
- May 21^{st} 2007 in Basel: technical discussions about the ANA abstraction layer ('vlink'). Planning for the minmex status reporting interface. First version of the software documentation.
- June 26^{th} 2007 in Zurich: technical discussion about the node compartment repository (KVR) and the generic ANA thread interface. Planning for the Ethernet compartment development.
- July 4^{th} 2007 in Zurich: as part of the WP3 monitoring meeting, discussion sessions about the basic of the ANA core software and the bootstrapping procedures for the Ethernet compartment. Goal was to understand how to include the future monitoring software.
- July 20^{th} 2007 in Basel: technical discussions on the wrapper functions. Planning and organisation of the upcoming coding workshop.
- August 22^{nd} to 24^{th} in Basel: First ANA core coding workshop. Detailed presentation of the ANA core software APIs and various code examples to other project partners. Planning of future developments. The feedback obtained from the participants has lead to various changes in the API and has initiated the development of the API level 2.

- October 25^{th} 2007 in Basel: development status. Discussions on Ethernet compartment message exchange details. Initial discussions about the IP compartment.

4.3 A subversion repository

Subversion (SVN) is a version control system initiated in 2000 by CollabNet Inc. It is used to maintain current and historical versions of files such as source code, web pages, and documentation. The SVN repository for the ANA core software was started in mid-April 2007 and is hosted and managed by the University of Basel at the address *https://subversion.cs.unibas.ch/repos/ana/*. The main usage was for code and documentation storage. The choice of *Subversion* over other version control systems was motivated by the fact that it fitted our needs for concurrential development of the ANA core software in addition to the fact that most of the collaborators had previously worked with it.

The development currently follows two svn branches:

- An unstable branch: Developers that are not yet sure of the stability of their code, but yet willing to share it on the svn for collaboration or testing reasons, may add it to this branch. Once the code is stabilized, it can then be merged with the trunk.
- A stable branch (or trunk) of the repository: This branch contains the stable ANA core software. Chekpoints are regularily made by merging the unstable branch with this one as shown in the figure below.

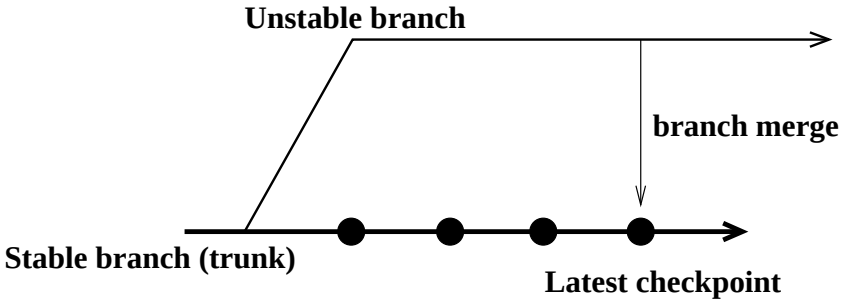

Figure 3: Subversion branches

5 Development platform

The following choices were made by the development team :

- Programming language : the choice for the programming language for the initial prototype was fixed on the C programming language. This choice came very naturally due to the low system level of the programming (networking stacks, raw sockets, Linux kernel code, etc) and the ease of all partners with this standard programming language. Note that a version of the ANA core software in the JAVA programming language was also envisaged. This would have enhanced the portability of the code to a larger panel of devices, notably those taking part of Personal Area Networks (mobile phones, PDAs). However due to time and resource constraints, this JAVA version was put on hold. Note however that this port to JAVA is not abandonned, and should be initiated as soon as a stable prototype is available.

- Operating system and compiling platform : Choice of development platform went for the linux operating system and using the open source GNU C Library and development tools. Again the choice here came naturally due to the popularity of the Linux operating system in the academic world (all of the ANA core developers are daily Linux users). Note also that this would be helpfull if the ANA core software is to be licenced as an open source software.

- Code debugging and validation tools : the following open source tools were regularily used during development :

  - Gdb (*http://sourceware.org/gdb/*) : the GNU debugger. The purpose of a debugger such as GDB is to allow the programmer to see what is going on "inside" another program while it executes to e.g., help identify bugs and deadlocks.
  - Valgrind (*http://valgrind.org/*) : Valgrind is an award-winning instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile programs in detail. In our case, we mainly used the *memcheck* tool of this framework for detecting memory leaks.

6 Milestones in development process

- May 2007 :

  - First merged version of the minmex and API level 0, working on both user space and kernel space.
  - First clean makefile hierarchy.
  - First version of API level 1 and Node Compartment (Key Value Repository).
  - First version of the vlink brick (Abstraction Layer) running on kernel space and userspace.
  - First code documentation skeleton in LaTex.

- June 2007
  - First interface to configure vlink brick.
  - First version of the Status Interface. This interface is intended in future to inform the user about the minmex's status, and to help for debugging during the development process.
  - Backporting of the ANACore code to linux kernels 2.6.7, 2.6.12, 2.6.17, 2.6.20.
- July 2007 :
  - First version of generic ana timers.
  - Revision of the node compartment, added support for boolean queries.
  - First version of the quick repository.
  - First version of the Ethernet compartment brick.
  - Adoption of Unix Sockets as default IPC for brick to minmex communication.
- August 2007
  - Restructure of the svn repository directories.
  - First version of generic ana locks.
  - Revision of vlink brick. Added support for UDP tunneling.
  - First version of ana threads API.
- September 2007
  - First version of a demonstrator Chat application brick.
  - Following first coding workshop feedback, revision of wrapper functions strategy.
  - Following first coding workshop feedback, simplification of code by moving some kernel specific code to include files.
  - Following first coding workshop feedback, revision of the generic API arguments.
- October 2007
  - First version of API level 2.
  - Port of API level 2 to kernel space.
  - Revision of the vlink brick. Added support for attachment to multiple minmexs.
  - Revision of the Ethernet Compartment brick. Changed bootstrap mechanisms and message exchange methods between Ethernet peers.
  - First version of the Internet Protocol compartment.

- November 2007
  - First port of the ANACore software to the Network Simulator 2 platform.
  - Port of the Ethernet Compartment brick to the NS2 platform.
  - First versions of monitoring bricks.
  - Revision of API level 1 message encodings (XRP).
  - Development of additional monitoring bricks.
  - Port of chat application to run over IP compartment.
- December 2007
  - Start of Functional composition developments.
  - Port of IP compartment brick to the NS2 Platform.
  - Revision of IP compartment. Added RIP routing.
  - First version of validation testing infrastructure. This tool will help developers check that code modifications do not have negative side effects on other parts of the ANA core software.

7 Extra activities

Although this was not explicitly stated in the revised description of work, a critical activity of this prototype development phase has been to help revising the high-level concepts of the ANA Blueprint based on implementation experience and feedback. That is, while developing the prototype software we also wanted to assess and eventually update the core communication concepts and paradigms of ANA.

The implementation experience gained in 2007 has actually lead to the release of an updated version of the ANA Blueprint in 2008 [5]. In particular, this new version introduces new communication concepts which should clarify how resolution ("routing") requests are handled by network compartments. The next phase in the first half of 2008 is to gain more practical experience while developing advanced autonomic services on top of the ANA core software in order to, once again and during the second half of 2008, revise the ANA core software implementation and the ANA Blueprint.

8 Future work

In 2007, the initial development activities have focused on producing a first prototype of the core ANA software upon which more advanced autonomic functionalities will be built. The focus in 2008 will now shift to two major developments tracks.

1. The consolidation and optimization of the current ANA core software. The main objectives are to enchance the ease of use and performance, and to incorporate new core features being defined during the next revisions of the architecture.

2. The development of autonomic features. The main objectives are to demonstrate autonomic networking principles and eventually embed/include some autonomic mechanisms in the core software.

It is also worth mentioning that 2008 will also mark the first public release of the ANA software.

References

[1] C. Jelger and S. Schmid (editors). ANA Blueprint (version 1), February 2007. ANA Deliverable D.1.4/5/6v1, available on http://www.ana-project.org.

[2] T. Roscoe. The End of Internet Architecture. In *Hotnets-V*, November 2006. Irvine, CA, USA.

[3] J. Crowcroft. Network Architecture Research Considerations Or The Internet Conspiracy, December 2006. Invited talk at ACM CoNext 2006, Lisboa, Portugal.

[4] K. Beck. Extreme Programming explained [Embrace changes]. *Addison Wesley.* ISBN 0201616416.

[5] C. Jelger and S. Schmid (editors). ANA Blueprint (version 2), February 2008. ANA Deliverable D.1.4/5/6v2, available on http://www.ana-project.org.