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.1.11

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.1.11
Document Title	Software development process for the ANA Core implementation
Workpackage	WP1
Editor	Ghazi Bouabene (UBasel)
Authors	Ghazi Bouabene (UBasel) Christophe Jelger (UBasel)
Dissemination level	Public
Contractual delivery date	31^{st} December 2008
Delivery date	15^{th} February 2008
Version	Version 1.0

Abstract:

This document is a companion document of the software deliverable D.1.11. The goal is to provide a concise summary of the development choices and activities of the ANA Core software during the year 2008.

Keywords:

ANA, development coordination, development milestones.

Table of content

1 Introduction

Since the development of the initial prototype in 2007, the effort on the implementation of the ANA Core software has continued at an important pace. In 2008, the ANA core prototype was considerably redesigned and refactored to produce an improved second generation of the software. Compared to 2007, in 2008 almost all the partners of the ANA project joined in the software development effort. This increase in the development man-power helped to produce a wider diversity of code and enriched the second ANA Core prototype with a large panel of functionalities.

This document motivates the design choices of the year 2008 and presents the development process of the second prototype of the ANA Core software. It is in fact the third part of deliverable *D1.11* consisting of :

- **D.1.11a** — The second prototype of the ANA Core software.
- **D.1.11b** — The updated documentation of the ANA Core software.
- **D.1.11c** — The description of the software development process (this document).

The present document does not describe the software itself. For a detailed description of the software, we refer the reader to the updated "ANA Core documentation" (D.1.11b) which describes inner details of the Core software in addition to a tutorial to develop ANA bricks.

2 Modifications to the first prototype

Shortly after its release, the first ANACore prototype was used by several ANA project partners to develop their bricks. This "testing" phase provided the core development team with valuable feedback which permitted to identify the following weaknesses of the ANA Core software:

1. Over-complexity of the code:
    - Due to the wrapper functions allowing to execute the same code in 3 different environments: user-space, kernel-space and NS2 simulation environment.
    - Due to the built-in support for various IPC mechanisms as the unique communication means between the minmex and the bricks.

2. Performance limitation due to the overhead introduced by the IPC mechanisms for inter-brick communication.

3. Complex compilation system : new brick's integration into the ANACore Makefile system was difficult.

4. Unsuitability of debugging information for the test-bed environment: all the brick's debug info was printed out on the standard-output making it difficult to collect status of remotely deployed bricks and minmexs.

To fix all these problems, the following changes were applied to the new prototype:

Removal of NS2 support - This has solved part of the first problem as it releaves the code from all the NS2 low level wrappers and therefore greatly reduces code complexity. This choice was motivated by the over-complexity of emulating system-related behaviors in NS2 (e.g. thread synchronization and multi-threading in general), the lack of available man-power within the developers team, and the important fact that NS2 is no longer maintained and will soon be replaced by NS3 (a fully -re-developed version of NS).

Plugin model for minmex - This modification has solved part of the first problem as well as the second. In this programming model, the minmex functionality can be extended dynamically (during execution) by loading *.so* (i.e. user-space shared libraries) or *.ko* (kernel-space) plugins. Using this model, bricks can be directly incorporated in the minmex as plugins. Plugin bricks have the advantage of direct function calls (via shared memory) with the minmex (for API level 0 calls) and are therefore more performant than bricks relying on IPC mechanisms (Gates model) for communication with the minmex. The Gates model was however not abandoned since it offers flexibilities such as heterogeneity of programming language between the minmex and the bricks as well as the possibility to attach remote bricks via UDP sockets. The support for the gates model at the minmex level was introduced in the form of a dynamically loadable plugin. This has the advantage of relieving the core minmex code from the IPC handling complexity when it is not required. Figure 1 depicts the different brick attachment possibilities to the minmex. More technical details on the plugin model can be found in D1.11b.

Revisited build system - A new compilation mechanism was developed from scratch: it operates via a front-end control file, using a simple syntax allowing users to specify, within this file, the list of bricks to compile as well as in which mode (i.e. user-space or kernel). Ready-to-use template Makefiles are also provided to ease the integration of the developed bricks into the main compilation system. More technical details on this matter can be found in D1.11b.

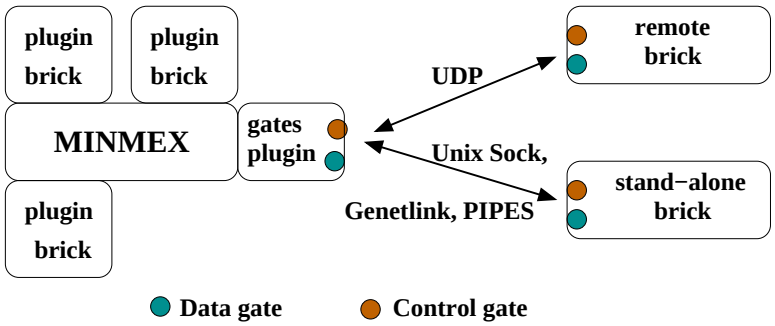

Figure 1: Brick attachment to minmex

Debugging information dumped in log files - The user can configure, via a shell environment variable, a path to store all the log files produced by the components of the ANA software. Both the minmex and brick's debug output are dumped in logfiles named with their process id (PID) and the name of the brick (or *anaMinmex* in case of the minmex). These files can then be easily retrieved after the brick's deployment in the testbed environment. More details on this can be found in the D1.11b deliverable.

The second prototype also incorporates the implementation of several new features and abstractions:

Brick instantiation from within the code - This allows a brick, e.g. the Functional Composition brick, to instantiate another brick from within the code (i.e. via a function call). The new brick can be loaded as a plugin or a stand-alone brick. Such a functionality can prove useful in several self-adapation scenarios.

Event notification system - In order to enhance the bricks' autonomous reactivity, the second prototype incorporates a system allowing bricks to be informed when some events happen within their node compartment. This system operates as a pub/sub system where bricks subscribe to particular events. The events can be very specific such as "Alert when IDP x is deleted" or generic such as "Alert when any brick is attached or detached". More details on the events available so far can be found in deliverable D1.11b.

Release IDP primitive - This important API primitive was missing in the first prototype design and implementation. It allows a brick using an IDP to alert the IDP's owner that it no longer requires the IDP. This allows the owner brick to free some resources in case the IDP (service) is no longer needed by any other bricks.

IDP busy flag - By setting this flag on, an IDP's owner brick can indicate that it temporarily refuses to receive messages on that IDP. Other bricks sending messages to the IDP will receive an error indicating that it is temporarily busy.

This functionality offers autonomous bricks (e.g. schedulers) more flexibility in managing their work flow.

IDP information repository - This repository allows bricks to set and retrieve information regarding IDPs. This information indicates the IDP's owner brick and whether or not the IDP is permanent. In case the IDP is the start of an information channel, the IDP's owner brick can (the brick can choose not to) indicate the destination's SERVICE and CONTEXT within the network compartment containing the channel, as well as the channel's MTU. Such network information visibility can strongly enhance inter-compartment operations.

Erlang implementation of the ANA API - The second ANACore prototype offers to developpers the possiblitiy to implement bricks in the *Erlang* programming language. The prototype does indeed offer the necessary libraries and low level bricks to allow *Erlang* bricks to attach to the C-programmed minmex. This feature opens a new horizon for ANA application's diversity.

doxygen documentation - The second ANACore prototype code incorporates some *doxygen* specific comments. This documentation shortly summarizes the available ANA API functions as well as the most vital structures. The role of this documentation is to serve as a quick manual to the "daily" ANA developer; role for which the main ANACore documentation (D1.11b) is not well suited.

3 Refactoring strategy

The implementation of the second ANACore prototype was done in an incremental re-construction process. Indeed, a new *empty* SVN branch was started to which the re-factored components were incrementally added (starting by the most vital ones i.e. minmex, APIs etc.). As a final step, project partners that already implemented bricks within the first prototype ported their work to the new branch. The major milestones of the second prototype development process are more detailed in Section 6.

4 Developers team

In 2008, the ANA core software development team has been composed of 17 persons from 8 different project partners. The persons involved in 2008 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.

Developer of the IP2ANA adaptation layer and the Compartment finder (cfinder) brick. 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. Karl André Skevik (UiO) : in charge of the development of the *custcom* peer to peer live streaming system compartment (several bricks).
- Ing. Marco Bicudo (UPMC) : in charge of the development of the Content Centric Routing (CCR) compartment bricks.
- Ing. Pierre Imai (NEC) : in charge of the development of the *turfnet* compartment bricks
- Ing. Hans Vatne Hansen (UiO) : in charge of the development of the *MCIS* compartment bricks.
- 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 the node compartment, Ethernet compartment and netShare 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. Deelopment of agnostic chat brick. Also in charge of supervision of the development of the *IP*, *Field based Routing* and *SAFT* compartments.
- Christoph Mertz, PhD candidate (Fokus): in charge of the development of the flow measurement brick and the *Failure Detection Engine*.
- Stephan Dudler, Master student (ETHZ): Finalization of contribution to the IP compartment and development of the *Field Based Routing* compartment.
- Andre Graf, Master student (UBasel): in charge of the development of the *Erlang* API wrappers for ANA as well as demonstrator *Erlang* bricks.
- Andreas Louca, Master student (ULanc): in charge of the development of the first *Functional Composition* brick.
- Nikolay Tcholtchev, Master student (Fokus): in charge of the *Failure Detection Engine* bricks development.
- Thomas Other, Master student (ETHZ): in charge of the "Podnet" port to ANA
- Stefan Lienhard, Bachelor student (ETHZ): development of the *Store and Forward Transport Protocol (SAFT)*

- Beat Gebistorf, Bachelor student (ETHZ): in charge of Interconnecting the Field Based Service Discovery (FBSD) and the IP/RIP Compartments
- Mathias Fischer, Bachelor student (ETHZ): development of the Autonomic Identifier Allocation (AIA) brick.

5 Coordination tools

The same coordination tools, i.e. the *subversion* repository and the *ana-dev* mailing list, described in last year's deliverable D.1.8c, proved to be efficient even with the major increase in the number of developers With the number of developped bricks rapidly increasing, a wiki page providing information on the available bricks was also started. For each brick, the page indicates the brick's dependencies (which other bricks are required for it to work) and the KVR keywords used by the brick. The goal of this page is to ease the integration of new bricks in the ANACore software and to provide a global overview of the development process in order to avoid overlapping (redundant) efforts.

6 Milestones in development process

- May 2008 :
  - Start of the new SVN development branch
  - Minmex with *.so* plugins support (and without NS2)
  - Addeed support for API Levels 0, 1 and 2
  - Support for minmex and APIs in Kernel space
  - All brick's debug messages are dumped into log files
  - Support for brick validation system
  - First version of the *doxygen* documentation
- June 2008 :
  - Addeed *vlink* brick
  - First version of the *ip2ana* framework
  - First version of the gates plugin
  - Added the *eth-vl* brick (Ethernet compartment)
  - New makefile system. Compilation is now controlled by the file *config.txt* in the top-level directory
  - Initial support for IP compartment
- July 2008 :

  - Kernel version of the *gates plugin* with generic netlink and UDP
  - Initial version of *field based routing* compartment
  - Initial version of the compartment finder (cfinder) brick
  - Improvements of the ip2ana framework
  - Added support for the IDP information repository
  - Improvements to the IP compartment : RIP routing protocol
  - First version of the compartment agnostic *chat* brick
  - Added support for the event notification system
  - First public release of the ANA Core software
- August 2008 :
  - Major update of the ANA Core documentation
- September 2008 :
  - First versions of the *Vivaldi* and latency bricks
  - Initial versions of the *Content Centric Routing* bricks
  - Vlink support for TOSCANA web-based GUI
  - Added support for *Erlang* bricks
- October 2008 :
  - Initial version of the *Functional Composition* brick
  - Initial version of the flow measurement brick
  - First version of the zeroconf test brick
  - Field Based Routing compartment improvements
  - Added support for IDP release primitive, IDP busy flag
  - Added support for brick instantiation from the code
- November 2008 :
  - Integration of *SAFT* routing protocol to the IP compartment
  - Improvements on the Content Centric Routing compartment
  - Added Multi-Compartment Information Sharing (MCIS) bricks
  - Initial work on FBR and IP compartments co-operation
- December 2008 :
  - First version of the *Autonomic Identifier Allocation* (AIA) brick
  - First version of the *turf* brick
  - ANACore documentation review
  - Improvements of the CCR bricks

- January 2009 :
  - Various bug-fixes to the Core software
  - Added neighbor discovery support to the monitoring framework
  - Improvement of the flow measurement brick
  - First stable version of the p2p video streaming compartment

7 Future work

The main activity of this task in 2009 is to carry on the development of the ANA Core software. The main objective is the delivery of the final stable release at M48 (December 2009), including the main components of ANA (minmex + API), an extensive documentation, code templates, and all the bricks (developed by other work packages) that reached a stable status. Note that this task requires a close collaboration with Task 4.3 (Integration) to coordinate the integration of bricks into the M48 software release.

Compared to the previous code releases, the M48 version will also include an integrated command-line interface and scripting facility that will allow users to configure and control all the components of ANA in a flexible and easy manner. This development, code-named CLOSE (Control-LOop Scripting Engine), will eventually become the core enabler for autonomicity in ANA as it will permit to easily and dynamically encode the behavior of ANA components.