Simple Economic Management Approaches of Overlay Traffic in Heterogeneous Internet Topologies

European Seventh Framework Project FP7-2008-ICT-216259-STREP

Deliverable D3.2
Initial Documentation of Engineering and Implementation

The SmoothIT Consortium

University of Zürich, UZH, Switzerland
DoCoMo Communications Laboratories Europe GmbH, DoCoMo, Germany
Technische Universität Darmstadt, TUD, Germany
Athens University of Economics and Business - Research Center, AUEB-RC, Greece
PrimeTel Limited, PrimeTel, Cyprus
Akademia Gorniczo-Hutnicza im. Stanislawa Staszica W Krakowie, AGH, Poland
Intracom S.A. Telecom Solutions, ICOM, Greece
Julius-Maximilians Universität Würzburg, UniWue, Germany
Telefónica Investigación y Desarollo, TID, Spain

For more information on this document or the SmoothIT project, please contact:
Prof. Dr. Burkhard Stiller
Universität Zürich, CSG@IFI
Binzmühlestrasse 14
CH—8050 Zürich
Switzerland

Phone: +41 44 635 4355
Fax: +41 44 635 6809
E-mail: info-smoothit@smoothit.org

Document Control

Title: Initial Documentation of Engineering and Implementation

Type: Public (there also exists a confidential version for internal usage)

Editor(s): Konstantin Pussep, Nicolas Liebau

E-mail: pussep@kom.tu-darmstadt.de, liebau@kom.tu-darmstadt.de

Author(s): Konstantin Pussep, Osama Abboud, Nicolas Liebau, Michael Makidis, George Karakatsiotis, Spiros Spirou, Peter Racz, Maria Angeles Callejo Rodriguez, Rafael Cantó, Marcin Niemiec, Sergey Kuleshov, Haris Neophytou, Zoran Despotovic

Doc ID: D3.2-v1.7-public.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1, V0.2	January 19, 2009	Konstantin Pussep	First version, initial table of contents.
V0.3	January 29, 2009	Konstantin Pussep	Incorporated feedback from Michalis Makidis, adjusted document structure.
V1.0	February 10, 2009	Konstantin Pussep	Finalized structure based on partners feedback.
V1.2	February 24, 2009	Michael Makidis, George Karakatsiotis, Konstantin Pussep, Peter Racz, Marcin Niemec, Osama Abboud, Maria Angeles Callejo Rodriguez	Initial content for all sections included.
V1.3	March 3, 2009	All	Merged updated contribution and comments from all partners. Added Admin component description.
V1.4	March 12, 2009	All	Updated content from all partners. Included initial Appendix.
V1.5	April 3, 2009	Konstantin Pussep, All	Merged updated version of all sections, updated Introduction and executive summary.
V1.6	April 8, 2009	Konstantin Pussep, Nicolas Liebau	Version for the internal review prepared based on the updated partners input. Extended Appendix. Improved layout.
V1.7	April 20, 2009	Konstantin Pussep, Peter Racz, Maria Angeles Callejo Rodriguez, George Karakatsiotis, Burkhard Stiller	Integrated reviewer feedback Final version generated

Table of Contents

1 Executive Summary 7

2 Introduction 9
2.1 Purpose of the Document D3.2 10
2.2 Document Outline 10

3 Architecture Overview 11
3.1 Interfaces 12
3.2 Supported ETM Mechanism 13
3.3 System Deployment 13

4 Implementation Framework 15
4.1 Tools 15
4.1.1 Programming Languages 15
4.1.2 Libraries 15
4.1.3 Tool Chain 16
4.1.4 Version Control 16
4.1.5 Issue Tracking 16
4.2 System Tree Layout 17
4.3 Building and Running 18
4.3.1 System 18
4.3.2 SIS Server Subproject 18
4.3.3 Peer 19
4.3.4 System Test 19
4.3.5 Summary of Current Ant Commands 20

5 SIS Server Components 22
5.1 SIS Controller 22
5.1.1 Requirements 22
5.1.2 Offered Interfaces 23
5.1.3 Component Design 24
5.1.4 Customization for trials 28
5.1.5 Testing 28
5.2 SIS Database 32
5.2.1 Requirements 32
5.2.2 Offered Interfaces 33
5.2.3 Component Design 35
5.2.4 Parameters 39
5.2.5 Customization for trials 39
5.2.6 Testing 39
5.3 Metering 41
5.3.1 Requirements 41
5.3.2 Offered Interfaces 42
5.3.3 Utilized External Interfaces 43
5.3.4 Component Design 43
5.3.5 Testing 47
5.4 Admin Component 48
5.4.1 Requirements 49
5.4.2 Offered Interfaces 49
5.4.3 Component Design 49
5.5 Security Component 50
5.5.1 Requirements 50
5.5.2 Offered Interfaces 51
5.5.3 Component Design 51
5.6 QoS Manager and NGN Integration 58
5.6.1 Requirements 58
5.6.2 Offered Interfaces 59

5.6.3 Utilized External Interfaces 59
5.6.4 Component Design 59
5.6.5 Customization for trials 59
5.6.6 Testing 59
5.7 Summary 60

6 Extension of a P2P Client with SIS Functionality 61
6.1 Generic SIS Client 61
6.2 Overview of Tribler and NextShare 62
6.3 Integration of SIS in NextShare 63
6.3.1 Requirements 63
6.3.2 Relevant Mechanisms 63
6.3.3 Extensions/Modifications to NextShare 64
6.3.4 Offered Interfaces 64
6.3.5 Structural View 64
6.3.6 Behavioral View 64
6.3.7 Bipartite Sorting Algorithm 64
6.3.8 Tests 65

7 Summary and Conclusions 67

References 68

Abbreviations 70

Acknowledgements 72

Appendix A. Technology Scouting 73
A.1 Control Data Protocols 73
A.1.1 YAML 74
A.1.2 SOAP 74
A.1.3 REST 75
A.1.4 Google Protocol Buffers 75
A.1.5 Conclusion 75
A.2 Measurements 75
A.3 Security 76
A.3.1 SSL 76
A.3.2 RADIUS/Diameter 76
A.3.3 JBoss-based AA 78
A.3.4 Conclusion 78
A.4 Traffic Management Mechanisms 79
A.4.1 CISCO 79
A.4.2 Linux-based traffic shaping 79
A.4.3 Zebra 80
A.4.4 XORP 81
A.4.5 Conclusion 82

Appendix B. NGN Transport Control Functionalities 83

Appendix C. WSDL Interface Definitions 85
C.1 Interface sis.1 Definition 85
C.2 Interface sis.3 Definition 85

Appendix D. Integration and Release Procedure 86
D.1 Component Release Procedure 86
D.1.1 Prerequisites 86
D.1.2 Issue Tracking 88
D.1.3 Test-bed Access 89
D.1.4 Documentation 89
D.1.5 Release Procedure 89
D.1.6 Component Deliverables 90

D.1.7 Delivery *91*
D.1.8 Component Release Flow Chart *92*
D.2 Component Release Report 93
D.2.1 Component Owner *93*
D.2.2 Version Description *93*
D.2.3 Source Code *94*
D.2.4 Release Build *94*
D.2.5 Tests *95*
D.2.6 Comments *95*
D.3 Component Release Procedure: Component Integration and System Release Procedures 96
D.3.1 Prerequisites *96*
D.3.2 Component Integration Procedure *98*
D.3.3 System Release Procedure *100*
D.3.4 Component Integration Flow Chart *102*
D.3.5 System Release Flow Chart *103*
D.4 SmoothIT System Release Report 104
D.4.1 Build Identification *104*
D.4.2 Issue Tracking Report Document *104*
D.4.3 Component List *104*
D.4.4 Implementation Framework *105*
D.4.5 Change Summary *105*
D.4.6 Comments *106*

(This page is left blank intentionally.)

1 Executive Summary

The aim of this deliverable is to document the initial engineering and implementation activities within the SmoothIT project. The architecture design applied here has been already presented in deliverable D3.1. The final results of engineering and implementation tasks of the SmoothIT project will be described in deliverable D3.3, which is a follow on deliverable to this one, and will provide further details on the SmoothIT architecture's components.

Thus, this deliverable covers the architecture's state in the first SmoothIT release from March, 13th 2009. The interfaces and components described here are not yet final since the specification of ETM (Economic Traffic Management) mechanisms is still ongoing within WP2 and because the implementation will be synchronized with new proposals. Additionally, the deliverable describes the requirements and implementation plans of components that will be implemented in future releases.

Therefore, the deliverable presents the results of the first phase of tasks 3.3 and 3.4, covering the implementation of the SmoothIT architecture and initial ETM implementation. The latter is the BGP-based locality ETM mechanism as proposed by the ETM task force. The documentation of further ETM implementation will be subject to D3.3. Note that technological decisions made inside SmoothIT, and documented here, are based on the results of the technology scouting task T3.1. These are documented in the appendix.

This deliverable is related to the milestone MS3.3, due in June 2009, and will contain the second implementation release. Accordingly, the main objectives covered in this deliverable are to document the current state of the development, assurance of the proper integration of components, and support of ETM mechanisms together with the specification of implemented and planned components.

The main results documented in this deliverable are:

- Instantiation and elaboration of the general architecture proposed in D3.1.

  Deliverable D3.1 presents the design of the general software architecture to be developed within the SmoothIT project. This deliverable presents the engineering results and implementation details of the first release of this software.

- Detailed definition of components, interfaces, and functionalities.

  The SmoothIT architecture is structured into several major components. In this deliverable these components are described in terms of functionalities, interdependencies, and interfaces.

- Description of the implementation framework.

  SmoothIT follows a professional software development approach. In order to allow efficient and flawless trials of our ETM concepts, we built an implementation framework that allows us to build, test, and deploy the SmoothIT software efficiently.

- Integration with the NextShare client software.

  SmoothIT will evaluate the researched concepts within an internal and external trial. For this trial, NextShare was selected as a real world application. NextShare is a BitTorrent-based p2p filesharing application that additionally provides a video-on-demand feature. The results will be used for further work in WP3 and WP4.

As algorithms and specifications may be subject to patents and standardization involvement, there are two versions of this document: a private version for internal usage, and a public version that will be available at the SmoothIT homepage.

2 Introduction

This deliverable presents the results of the initial engineering and implementation efforts within SmoothIT's tasks T3.3 and T3.4. Within these tasks the SmoothIT architecture, described in deliverable D3.1, is being implemented in order to prepare the internal and external trial of SmoothIT. This deliverable is the initial version of the documentation of the engineering and implementation results; D3.3 will present the final results after the implications from the internal trial on the architecture have been integrated.

The implementation of Economic Traffic Management (ETM) mechanisms proposed in WP2 and discussed in [D2.2] must be supported by the SmoothIT architecture as proposed in [D3.1]. This architecture is intended to support a Triple-Win situation for end-users, ISPs, and overlay providers. Therefore, it must be incentive-compatible, support different overlay applications and various optimization schemes. Another core requirement covered by this architecture is the support of Quality-of-Service (QoS) as offered by modern Next Generation Network (NGN) equipment.

These requirements are supported by additional intelligence on the ISP side and (partially) by the information exchange between overlays and ISPs. Other approaches, from a related work (e.g. [P4P], [oracle]), that try to provide an interface between overlays and network providers, differ from SmoothIT requirements in terms of granularity, flexibility, and operation requirements. For example, the oracle approach does not consider overlays acting in multi-source download mode, while P4P offers no clear solution to tracker-less applications.

Another aspect of the implementation is the proof-of-concept integration of the SmoothIT architecture with the NextShare client, which is a modern P2P file sharing and streaming overlay application. It allows testing the SmoothIT architecture with a real world p2p application within the project trials.

The scheduled internal and external trials of the SmoothIT project cover two major areas: test-bed and real-life evaluation of the approach. This results in additional requirements for the SmoothIT architecture regarding easier deployment, extensibility, scalability, efficiency, security, and robustness [D3.1]. These requirements have been taken into account while implementing the first release of the SmoothIT architecture.

The SmoothIT architecture is also designed to be flexible in order to support different ETM mechanisms. The goal of the first software release is to support the initial ETM mechanism that offers a locality mechanism based on BGP information from the ISP's equipment. The system is easily used to support extensions and modifications of this ETM mechanism, as well as further ETM mechanisms currently under specification within WP2.

The design of the architecture's components presented here is based on the technology scouting activities performed by Task 3.1, in 2008. The architecture is based on the specification provided in deliverable D3.1. Due to the early stage of the development activities, which started in January 2009, most components are still subject to changes based on the extended specification (e.g. QoS Manager) that will be coming from WP2 or have not been implemented yet. Such components will be implemented in the future releases as they are relevant mostly in the external trial (Admin and Security components). Detailed results of the further engineering and implementation efforts within SmoothIT will be documented in D3.3, which will be presented after project month 27.

Finally, this deliverable also shows the detailed designs of specific components (interfaces, data structures and behavior diagrams) that are the basis for the exploitation, dissemination, and standardization of results (via patents). It should be noticed that the detailed specification of the system and its future implementation allows the consortium to present consistent use cases, which will allow the consortium to present SmoothIT solutions in relevant forums.

2.1 Purpose of the Document D3.2

Deliverable D3.2 documents the initial development of the SmoothIT implementation architecture. It further documents the implementation of the first ETM approach within this architecture.

This deliverable assists the SmoothIT partners in elaborating on the architecture design, on details of single components, and in identifying additional inter-dependencies between the components. It documents the development framework, allowing developers to more easily understand the system, and makes the contribution and deployment process more efficient. It documents the initial component implementation and, as such, permits a cross-checking between owners of related components. This especially applies to the documentation of the initial ETM mechanism implementation that allows verifying of the correctness, feasibility, and completeness of specifications developed in WP2. Furthermore, this deliverable documents the adaptation of the NextShare client, which serves as the real world application used by SmoothIT within the trials.

2.2 Document Outline

The structure of this deliverable is as follows: Section 3 describes the revised architecture design, interfaces between components, and shortly presents the initial ETM mechanism. In addition, it presents the deployment of the developed architecture. Section 4 presents the implementation framework prepared to integrate the components developed by single partners. Sections 5 and 6 constitute the core of the deliverable. Section 5 describes all the components that make up the SmoothIT Information Service (SIS), while Section 6 contains the integration of SIS support in the NextShare overlay client. Section 7 summarizes the deliverable.

Additionally, a set of annexes is available in order to provide further details and/or present support information. In Appendix A the results of the technology scouting activities are presented, while Appendix B describes NGN functionalities. Appendix C presents the exact definitions of the external SIS interfaces. Finally, Appendix D shows the component release and integration procedure applied within SmoothIT.

3 Architecture Overview

This section summarizes the SmoothIT architecture as described in [D3.1]. Since then, the SmoothIT architecture has not undergone any major modifications. However, the detailed design of ETM mechanisms in [D2.2] has delivered more insights into the functionality of the architecture's components.

The architecture assists overlay applications in achieving a more efficient overlay of traffic routing, a reduction in the operational costs of ISPs, and an improvement in the service performance of overlay applications. The SmoothIT Information Service (SIS) is provided by the SIS Server, which is deployed in the network of the ISP. Peers in overlay applications query the SIS Server with a list of peers who are candidates to establish a connection. The SIS Server sends back the SIS preference value for all peers in the list. The SIS preference represents the preference of a peer according to a certain ETM mechanism, e.g., a peer in the local Autonomous System (AS) is preferred over a peer in a distant AS. The SIS preference value is calculated by the SIS Controller and used by the SIS Client in the overlay application according to a given ETM mechanism.

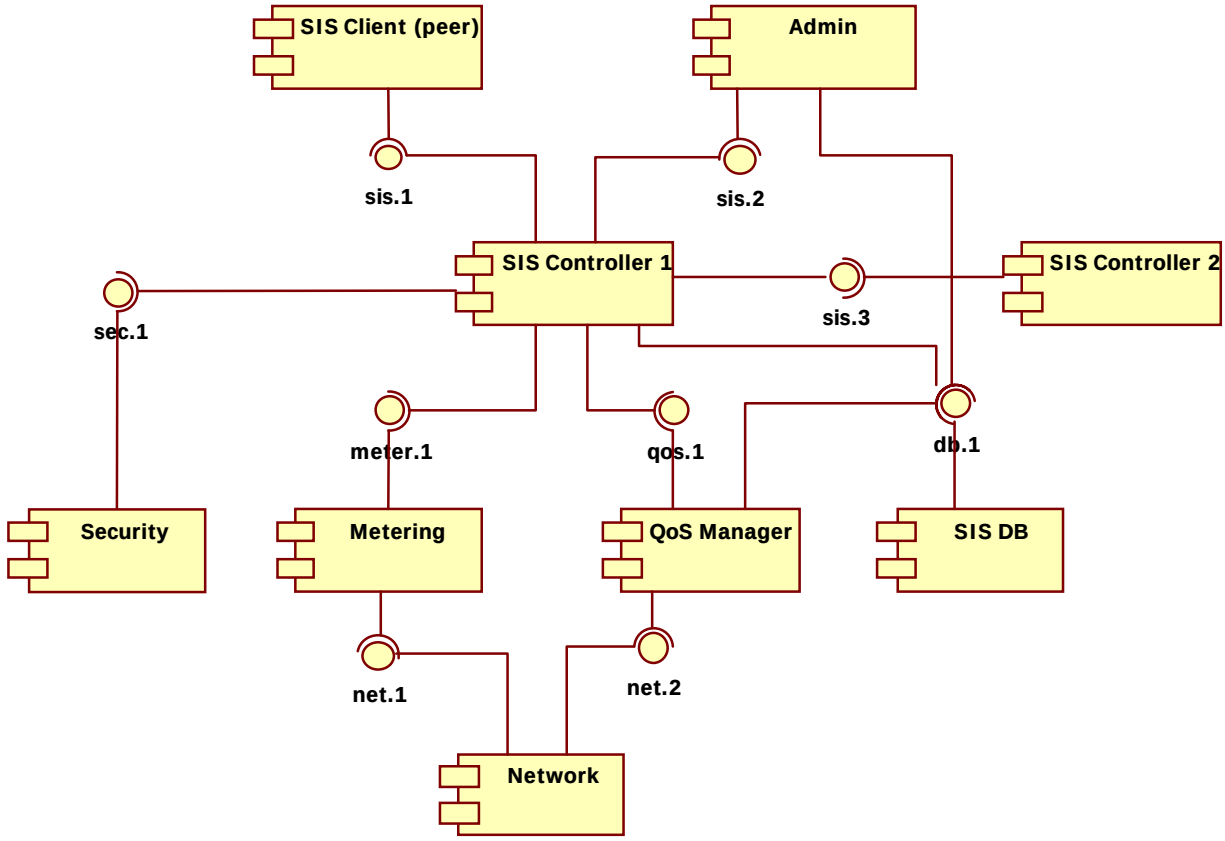


Figure 1: SIS architecture and interfaces

Figure 1 shows the components of the SIS (SmoothIT Information Service) architecture and the interfaces between the components. The SIS Client is integrated in the overlay application and communicates with the SIS Controller over the SIS protocol. The SIS Controller is the main component of the architecture, which includes the ETM logic and assists in the peer selection of the overlay application. In order to implement the ETM, the SIS Controller can get information from and provide information to other components, like

Metering, QoS Manager, Security, and SIS Database (SIS DB). The system can be configured and managed over the Admin component.

3.1 Interfaces

The interfaces between SmoothIT architecture components are shown in Figure 1 and summarized in Table 1. Each interface has a unique name that consists of the name of the component providing it and an index number, e.g., meter.1 is an interface provided by the Metering component. This scheme enables the adding of new interfaces by assigning it a new index. Previous interface names do not have to be changed.

The SIS Controller provides interfaces to the SIS Client (sis.1), to the Admin component (sis.2), and to other SIS Controllers (sis.3). The QoS Manager provides an interface to manage QoS settings (qos.1) to the SIS Controller. The interface of the Metering component (meter.1) is used by the SIS Controller to receive BGP information. The Security component provides an interface (sec.1) to the SIS Controller. The SIS DB provides an interface (db.1) that is used by the SIS Controller, the Admin component, and to the QoS Manager to store/retrieve information. The Network provides an interface (net.1) to read BGP routing tables over SNMP and an interface (net.2) to manage QoS settings.

Table 1 summarizes the interfaces and states the sections in this document where the details about the interfaces are explained.

Table 1: Interfaces

Interface ID	**Component providing the interface**	**Component using the interface**	**Purpose**	**Protocol**	**Reference**
sis.1	SIS Controller	SIS Client	Get a ranked list of peers	SOAP	Sec. 5.1.2
sis.2	SIS Controller	Admin	Store/Retrieve admin data	Java API	Sec. 5.1.2
sis.3	SIS Controller	SIS Controller	Get info from the other SIS Controller	SOAP	Sec. 5.1.2
qos.1	QoS Manager	SIS Controller	Perform QoS reservations	Java API	Sec. 5.6.2
meter.1	Metering	SIS Controller	Get BGP information	Java API	Sec. 5.3.2
sec.1	Security	SIS Controller	Provide security services	Java API	Sec. 5.5.2
db.1	SIS DB	SIS Controller, Admin, Metering, QoS Manager	Store/Retrieve configuration data	Java API and JDBC to access the database server	Sec. 5.2.2
net.1	Network	Metering	Read BGP routing table	SNMP	Sec. 5.3.3
net.2	Network	QoS Manager	Enforce new policies in the network	SOAP (network technology dependent)	Sec. 5.6.3

Two interfaces are not offered by developed components, but must be offered by existing network equipment interface instead: net.1 and net.2. They are described together with the components that make use of them.

3.2 Supported ETM Mechanism

The first release of the SmoothIT architecture implementation supports the BGP-based locality promotion (BGP-Loc) ETM mechanism [D2.2]. In the BGP-Loc ETM mechanism the SIS Client contacts the SIS Controller and sends a list of potential peers to the SIS Controller. The SIS Controller assigns a SIS preference value to each IP address in the list and sends back the sorted list with preference values to the SIS Client. The overlay application uses the provided preference values to select and connect to peers. In the first release, the SIS Controller calculates the preference value based on the BGP information received from the Metering component. The BGP-Loc ETM mechanism includes the following high-level steps and uses the following components and interfaces (refer also to [D3.1] and [D2.2]):

- When the system is started, the SIS Controller reads the BGP routing information from the Metering component via the meter.1 interface. The Metering component gets this BGP routing information either from a BGP router or route reflector via the net.1 interface over SNMP or it can also read this information from a file (in order to support a test-bed environment).
- Based on the BGP routing information the SIS Controller calculates the SIS preference value for each prefix in the routing table. The Controller can re-read the routing information and re-calculate the SIS preference value at any time. After the SIS preference value is calculated, the SIS Controller is ready to receive requests from the SIS Client.
- Overlay applications use their regular peer lookup procedure to determine candidate peers to connect to and their peer selection procedure to unchoke peers. The SIS Client is integrated in the NextShare overlay application and it sends a selected list of peers (IP addresses) to the SIS Controller via the sis.1 interface. The SIS Controller assigns the pre-calculated SIS preference value to each peer in the list and sends it back to the SIS Client via the sis.1 interface. The SIS Client adjusts the unchoking of peers according to the received SIS preference value.

To support the BGP-Loc ETM mechanism, the SIS Client, the SIS Controller, the Metering component, and the SIS DB are required and implemented in the first version. Additionally, the current implementation includes basic functionality of the Security component and the QoS Manager that will be used in future ETM mechanisms.

3.3 System Deployment

The deployment diagram of the SmoothIT architecture is shown in Figure 2. The SIS Client is deployed within the NextShare overlay application on the user's computer. The SIS server is based on the JBoss [jboss] application server and includes the SIS Controller, Metering, QoS Manager, Security, and SIS DB components. For the SIS DB a DB backend, preferably, a MySQL server is deployed. The BGP router or route reflector provides the BGP routing information to the Metering component. The operator can configure the SIS server over a web browser.

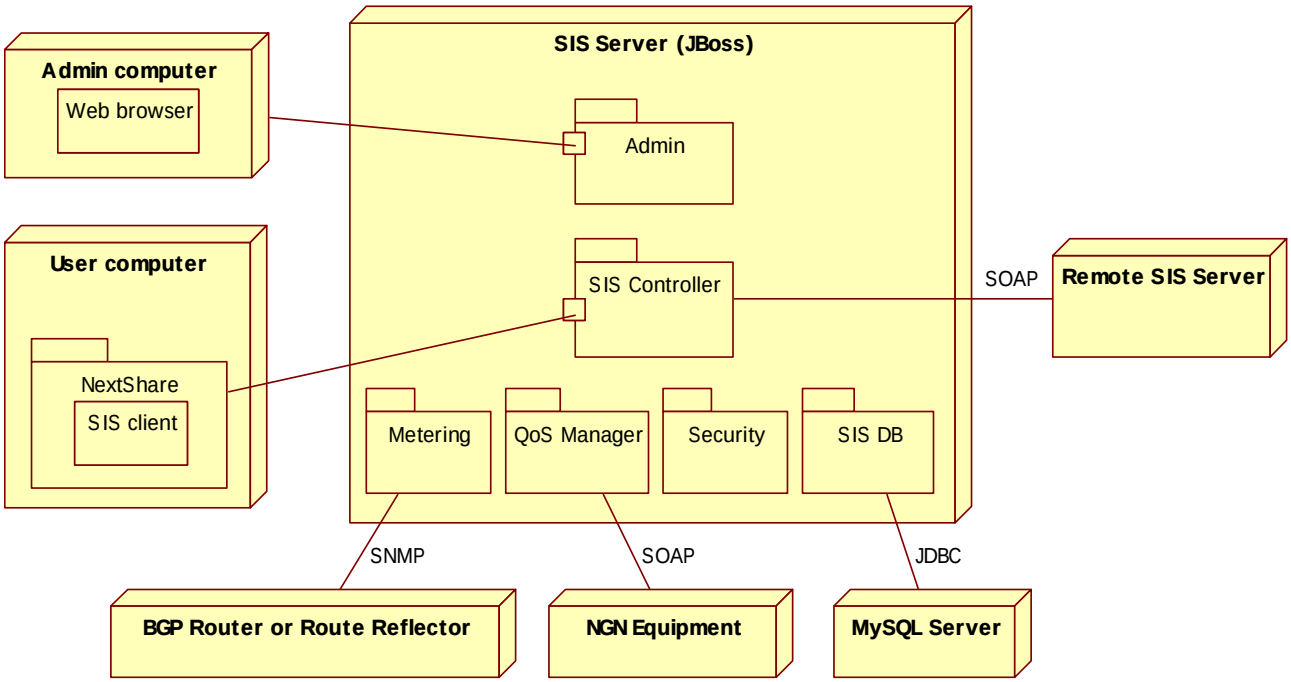


Figure 2: Deployment diagram

4 Implementation Framework

The implementation framework for the SmoothIT System includes all the tools and support libraries for designing, developing, testing and deploying the software created in SmoothIT as well as the system tree layout for the SmoothIT software. These tools are used by developers of SmoothIT components, to create their respective components, as well as by integrators, in order to produce a coherent, integrated system. The Implementation Framework includes the script-based Build System for SmoothIT, which automates the usage of some of these tools in order to ease the development and integration processes. It also ensures to some extent that developers and integrators use a common development, building, and testing environment.

4.1 Tools

The SmoothIT architecture is modeled in UML by using the free `StarUML 5.0` tool [staruml]. This tool was used to design the interfaces and data structures. The most representative diagrams per component are shown in the next sections, where each component specification is presented.

SmoothIT will be tested in the internal trial on a test-bed running the medium- to large-scale ModelNet Network Emulator [modelnet].

4.1.1 Programming Languages

The SIS components of the SIS server are developed in Java 6 (`JDK 1.6.0_10` [jdk]). Python 2.4 or later [python] is used for the NextShare client.

Although not required, the Eclipse IDE [eclipse] can be used to develop the Java-based components of SmoothIT. The PyDev (Python Development plugin) [pydev] can be used for the Python-based components.

4.1.2 Libraries

4.1.2.1 Java libraries

These libraries are used by the SIS and are included in the SVN tree:

- `log4j 1.2.15` for logging [log4j].
- `SNMP4J 1.9.3d` [snmp4j] to talk to network elements via SNMP.

The SIS server is deployed on the JBoss 5.0 Application Server [jboss]

4.1.2.2 Python Libraries

These libraries must be installed on the build machine before building the NextShare client:

- `ZSI 2.1_1a` for Python Web Services [zsi].
- `wxPython 2.8.0.1-unicode` [wxpython] Wrapper for the wxWidgets GUI toolkit. See also http://wiki.wxpython.org/InstallingOnUbuntuOrDebian for Linux installation information.

- `M2Crypto 0.18` [m2crypto] cryptographic library.
- `PySqlite` and `APSW` wrappers for the SQLite DB [pysqlite] [apsw].
- `PyXML 0.8.4` for Python XML processing [pyxml].
- Extra packages for Linux version (Debian 4): `OpenSSL 0.9.8c`, `swig 1.3.29-2.1`, `vlc 0.8.6` [openssl] [swig][vlc]

4.1.3 Tool Chain

Apache Ant 1.7 [ant] is the main tool used to build, deploy and test the SmoothIT system and the SIS server. It requires a JVM to run. The NextShare client software uses platform-specific shell scripts for building and testing. The build process also utilizes the following tools:

4.1.3.1 Java Tools

These tools are used by the SIS server and are included in the SVN tree.

- `Xdoclet 1.2.3` [xdocklet] to generate XML descriptors for J2EE applications.
- `Sun JAX-WS 2.1` [jaxws] reference implementation to generate Web Service artifacts. Please note the extra installation steps for use in Java 6.
- `JUnit 4.5` [junit] for Java unit testing.
- `Cobertura 1.9` [coberture] to calculate code coverage for JUnit tests.
- `FindBugs 1.3.6` [findbugs] for Java code static analysis.

4.1.3.2 Python Tools

These tools must be installed on the build machine before building the NextShare client.

- `py2exe 0.6.6` [py2exe] to generate a Windows executable from Python sources (for Windows version of the client only).

4.1.4 Version Control

The Subversion (SVN) version control system [svn] is used to archive and version the source code of the SmoothIT system. The SVN server used by SmoothIT is integrated with the RedMine platform provided by TUD. It can be accessed at http://dev.kom.e-technik.tu-darmstadt.de/svn/smoothit-integration (by authorized consortium members only).

4.1.5 Issue Tracking

The issue tracking feature of RedMine is used in order to help to debug the system and to track bugs, features, and changes. It can be found at: http://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/smoothit-integration/issues.

4.2 System Tree Layout

The system tree layout is the layout of the files and folders that contain the source code along with the other files of the project. Once a build has been performed, the root of the system tree contains the following elements:

- **/checksums**: Contains the checksums of the files of this version. Used to verify changed files between system and component releases.
- **/dist**: Contains the output of the build process, i.e. the binary code for all components that will be deployed to test-bed or trial environment. It is created upon a successful build.
- **/dist/testbed/sis**: The binary code for the server-side component (SIS) for the internal trial (testbed)
- **/dist/testbed/peer**: The Linux version of the binary code that runs on the test-bed
- **/dist/trial/sis**: The binary code for the server-side component (SIS) for the external trial. Currently, this is the same as the internal trial (test-bed) version.
- **/dist/trial/peer**: The Windows version of the binary code that runs on the end user machines for the external trial
- **/peer**: Contains the source code of the NextShare peer.
- **/sis**: Contains the elements for building the SIS. The root of this folder contains the Ant build scripts for the SIS.
- **/sis/dist**: Contains the output of the build process. It is created upon a successful build.
- **/sis/docs**: Contains the Javadoc documentation. It is created upon a successful build and test run.
- **/sis/lib**: The support libraries (jars) used for running the SIS.
- **/sis/resources**: XML files and other artifacts required by J2EE for running the SIS that are written by the developers.
- **/sis/resources-gen**: XML files and other artifacts required by J2EE for running the SIS that are created automatically by the build process. It is created upon a successful build.
- **/sis/src**: The source code for the SIS (organized in packages) that is created by the developers.
- **/sis/src-gen**: The source for the SIS that is automatically generated by the build process. This includes the Web Service stubs. It is created upon a successful build.
- **/sis/test**: The reports generated by the source code tests of JUnit, Cobertura, FindBugs etc. in HTML format. It is created upon a successful build with testing.
- **/sis/tools**: The support tools (jars) used for building and testing the SIS.
- **/sis/webroot**: The web components, such as JSP files, of the SIS.
- **/test**: Contains the SmoothIT System Test that runs on the test-bed. Version 0.1 of the SmoothIT system contained a simple Python Web Service client.

- **/tools**: Contains tools relevant to building and running the SmoothIT system.
- The root of the system tree contains the Ant script for automatically building, deploying and testing the entire SmoothIT system on the test-bed and the trial.

4.3 Building and Running

The Ant tool is the core of SmoothIT's build system. Before building the SmoothIT system the necessary tools and libraries mentioned above must be installed first. Once the SmoothIT system is built, the SIS part must be deployed on a JBoss AS. This section describes the specific steps to build and run the SmoothIT system.

4.3.1 System

The SmoothIT System build process is initiated by running Ant at the *project root directory*. The script at the root directory reuses the component-specific processes (described below) for building and testing the SIS and Peer (client) components. The output of this build process appears at the `/dist/trial` or `/dist/testbed` directory (see section 4.2), depending on whether this was an "external trial" or "internal trial" (testbed) build. You can set the build by passing the type as a parameter, e.g. `ant dist-testbed` or `ant dist-trial` (see section 4.3.5). This procedure can also be used to deploy and test the SmoothIT system on the test-bed.

4.3.2 SIS Server Subproject

The SIS server subproject is written in Java using the JavaEE (J2EE) framework and can be built by running the Ant scripts at its root directory (`/sis`). The output is a `sis.war` file that appears in the `/sis/dist` directory. In order to deploy this file, the command `ant deploy –Ddeploy.dir=$JBOSS_HOME/server/default/deploy` needs to be run. *`$JBOSS_HOME`* is located in the directory where JBoss AS 5.0 is installed.

The SIS build system includes a test procedure that combines the JUnit test cases, JUnit code coverage, (with Cobertura) and static analysis tests (with FindBugs). These tests can be executed by running `ant run-tests`. The output of the tests appears at the /sis/test directory. Each component owner of the SIS is responsible for providing their own JUnit test cases with adequate code coverage. This is ensured by the SmoothIT integration press, described in Appendix 2. Moreover, each component has clear interface and implementation parts. As such, each component that resides in its own package (where its interface lies) has two extra subpackages: the `.impl` subpackage, which contains the concrete implementation of this component, as well as the `.test` subpackage, which includes all the JUnit test cases of this component.

The SIS sub-project can be imported as a Java project in the Eclipse IDE [eclipse]. The source code for the SIS is annotated with Java and Xdoclet annotations, so the build process automatically generates all the necessary web service and web application artifacts. As a result, writing simple Java classes (POJOs) is all it takes to create web services, servlets and other J2EE components. For the InterSIS web service, the building

process automatically generates both the server-side and client-side web service stubs that are part of the SIS.

The SIS SOAP web services have only been tested on the `JBossWS` [jbossws] web service stack that is integrated with JBoss. Tomcat, Jetty and other web service stacks (such as Apache Axis) and application servers have not been tested and thus are not supported.

4.3.3 Peer

The SmoothIT peer is based on the NextShare peer provided by the P2PNext EU/ICT project [p2p-next]. It is written in Python and it uses custom shell scripts that are platform specific for the build process. These scripts are reused by the main Ant script at the SmoothIT project root directory. The output of the build process appears at the `/peer/dist` directory; to start the peer, run `swarmplayer.exe` under Windows or `python linuxRun` script under Linux.

The syntax of the `linuxRun` script is as follows:

linuxRun {nofilter|local|simple|sis [SIS-URL]}

The options have the following meaning:

1. **nofilter**: normal NextShare functionality
2. **local**: local filtering of IPs (for testing and debugging purposes)
3. **simple**: invoke simple ranking at the SIS side (for testing and debugging purposes)
4. **sis**: invoke the regular ranking at the SIS side

The optional SIS-URL parameter is only meaningful in **simple** and **sis** modes, where it allows overwriting the default URL of the SIS server.

4.3.4 System Test

The SmoothIT System Test is being implemented at the /test directory. It will be comprised of all necessary scripts and other software that are required in order to deploy and test the SIS and peer for the internal trial.

In the initial version, the system test can be started by running `ant run-localhost` on a Windows host. This will build the SIS and the Windows version of the peer. It will deploy the SIS on a JBoss running on localhost (as configured in file `localhost.properties`) and it will start the client. The user must manually start the JBoss application server and run a test with the client.

In a subsequent version, the SmoothIT system test will test one or more usage scenarios on the ModelNet test-bed. Again, the `ant` tool will be the front end to this test.

4.3.5 Summary of Current Ant Commands

The Ant tool is the front end to the entire system build, deploy, and test process. This section summarizes the Ant commands (called *targets* in Ant) implemented in the initial version.

The ant targets in the project *root directory* are:

Command	Result
`ant`	Same as `ant dist-trial`
`ant dist-trial`	Builds and tests the SIS. It also builds the client for Windows.
`ant dist-testbed`	Builds and tests the SIS. It also builds the SIS Client for Linux.
`ant clean`	Deletes the output of the previous build process.
`ant run-localhost`	Builds, tests, and deploys SIS to the default application server (as it is configured in the file `localhost.properties`). It also builds the client for Windows and starts an instance of the client
`ant checksum`	It produces the checksums for all the files of the tree. Used only when requested by the integration team.
`ant verify`	It verifies that all files of the tree against their checksum. Used only when requested by the integration team.

The available commands at the `sis` directory (for developers of the SIS):

Command	Result
`ant`	Builds and produces the binary version of the SIS under the /sis/dist directory
`ant run-tests`	Same as `ant` but also performs the JUnit tests.
`ant clean`	Deletes the output of the previous build process.
`ant deploy –Ddeploy.dir= <path to JBoss deploy>`	Builds and produces the sis.war and deploys it to the application server. The path of the application server deploy folder must be indicated after the argument –D. For example, if the application server is at `C:\jboss-5.0.0.GA\` the command should be `ant deploy –Ddeploy.dir=C:\jboss-5.0.0.GA\server\default\deploy`
`ant build-test-deploy –Ddeploy.dir= <path to JBoss deploy>`	Builds and produces the sis.war, performs the JUnit tests and deploys the sis.war to the application server. The path of the application server must be indicated after the argument –D. For example, if the application server is at `C:\jboss-5.0.0.GA\` the command should be `ant deploy –Ddeploy.dir=C:\jboss-5.0.0.GA\server\default\deploy`

5 SIS Server Components

The SIS is the core element of the SmoothIT architecture. Its components are the SIS Controller, the SIS DB, Metering, Admin, Security, and the QoS Manager with NGN integration. For each of these components the implementation is documented by means of presenting the functional and non-functional requirements, the offered interfaces, the structural and behavioral views, the implementation technology, parameters, and the implemented tests for system integration.

The SIS Controller implements SmoothIT's ETM mechanisms; thus, it is the central component that drives the design of the remaining components. It collects information from peers and Metering, returns information to the peers, controls the QoS Manager, and is configured using the Admin component. The initial implementation of the SIS Controller is based on the BGP-based locality ETM mechanism and is built using JBoss.

The SIS Database stores all information collected and required by all other components. It is built using Java Persistence API to access an SQL database. The SIS Database can be customized towards the internal and the external trial.

The Metering component collects information from the network and provides this data to the SIS Controller. The current implementation focuses on reading BGP information from BGP routers.

The Admin component has not been implemented for the first release of the SmoothIT architecture. The implementation details will follow in deliverable D3.4.

Also the Security component was not required for the first architecture release, as there were no security concerns to be taken into account during the internal trial. However, required functionality and implementation alternatives are discussed.

The QoS Manager and NGN integration are components required for the further ETM mechanisms, which are not part of the SmoothIT architecture's first release. The QoS Manager controls NGN equipment in order to manipulate the QoS delivered by the overlay application. The implementation details are private to the project.

5.1 SIS Controller

This component is the central part of the SIS server that implements the ETM mechanisms of SmoothIT. It receives input from the SIS Client (peer) and the Security and Metering components as well as configuration parameters from the Admin. It mainly provides output to the QoS management component. It further provides SIS Clients with preference rankings of other peers. Future functionalities will be described by WP2 through further ETM mechanisms.

5.1.1 Requirements

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1.	When the component receives a list of IP addresses from a client, it returns a list of those IP addresses with attached preference values. It can also add potential	X	

	good peers to the reply.		
F.2.	Provides the other SIS with its ETM data	X	
F.3.	Informs the QoS Manager to adjust the priority of a user for some stream	X	
F.4.	Whenever it is needed, the component updates its data about the network state by asking Metering and (external) SIS servers	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1.	Easy deployment: It must be independent of the environment in which it is being used. The configuration must be the same for internal and external trials.	X	
N.2.	Scalability: SIS Controller should be able to handle sufficient number of requests (large user population might produce many SIS requests for IPs from multiple ASes).	X	
N.3	Standard compliance: component shall use standard protocols and formats.	X	

5.1.2 Offered Interfaces

This component offers the following interfaces: sis.1 to the client, sis.2 to the Admin component, and sis.3 to other SIS servers. In the following, these are described in detail.

5.1.2.1 Interface sis.1

This interface is used by the SIS Client. The SIS Client sends through this interface a list of IP addresses to the SIS Controller. The SIS Controller ranks these IPs and returns the ranked peer list with their preference values (see Test Case 1).

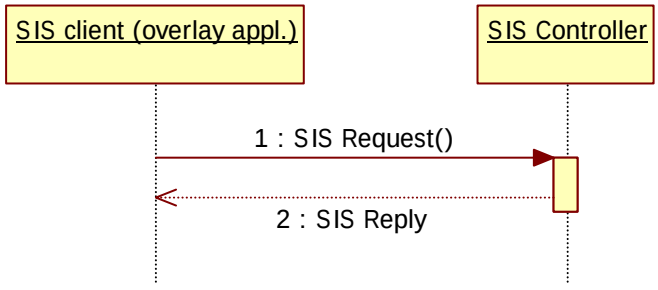


Figure 3: SIS Client protocol interaction

The communication between clients and the SIS Controller is based on SOAP via the provided method. It is realized in Java interface `ClientEndpoint:`

- `public Response getRankedPeerList(Request req)`

See Figure 6 for the details on Response and Request data types.

5.1.2.2 Interface sis.2

This interface will be used by the Admin component to get data about the SIS Controller's operation. Through this interface the Admin component will also be able to set any required parameters to the SIS Controller.

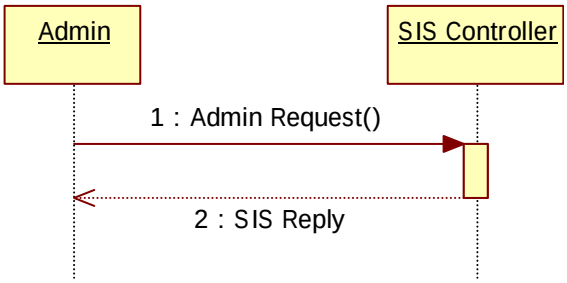


Figure 4: SIS Controller – Admin interaction

5.1.2.3 Interface sis.3

This interface will be used by the SIS Controller to request information from another SIS Controller.

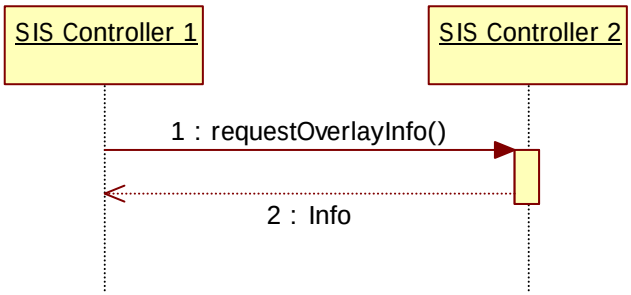


Figure 5: Inter-SIS interaction

This interface is represented by Java interface ***InterSisEndpoint*** providing the following methods:

- `public boolean authorizeServer(String key)`
- `public Response getRankedPeerList(Request req)`

The two SIS are exchanging messages in the SOAP format.

5.1.3 Component Design

The SIS Controller is being developed in Java 6 (JDK 1.6.0_10). It uses the library log4j 1.2.15. The communication between the SIS and a P2P client, as well between two SIS instances, are based on SOAP. The SIS interacts with the other components with Plain Old Java Objects (POJO).

5.1.3.1 Structural View

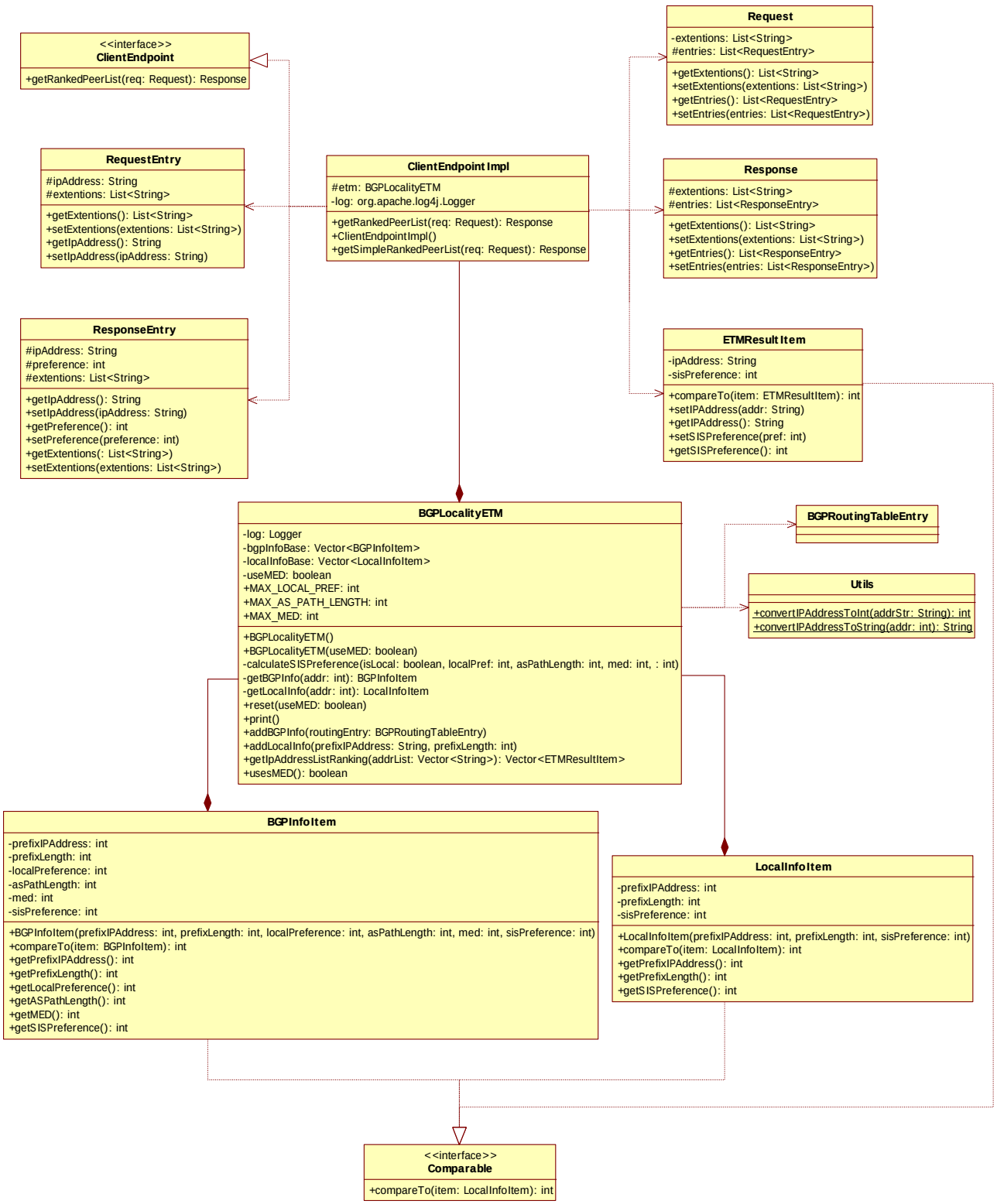


Figure 6: Class diagram of the SIS Controller

The component is located in package `eu.smoothit.sis.etm`

5.1.3.2 Behavioral View

5.1.3.2.1 The ETM algorithm (Commercial)

This section remains confidential for the time being.

5.1.3.3 Sequence Diagrams

To evaluate an IP, the SIS Controller will use one of the proposed ETM algorithms as described at D.2.2 (sections 5, 6 and 7).

Figure 7, Figure 8, and Figure 9 depict the sequence diagrams of the main functions of the SIS Controller:

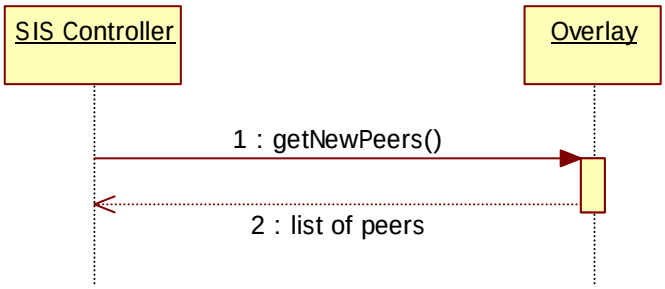


Figure 7: The SIS Controller gets new peers from the Overlay

Future versions of the SIS Controller implementation could be able to suggest new peers to the client. It could discover these new peers by asking the overlay application.

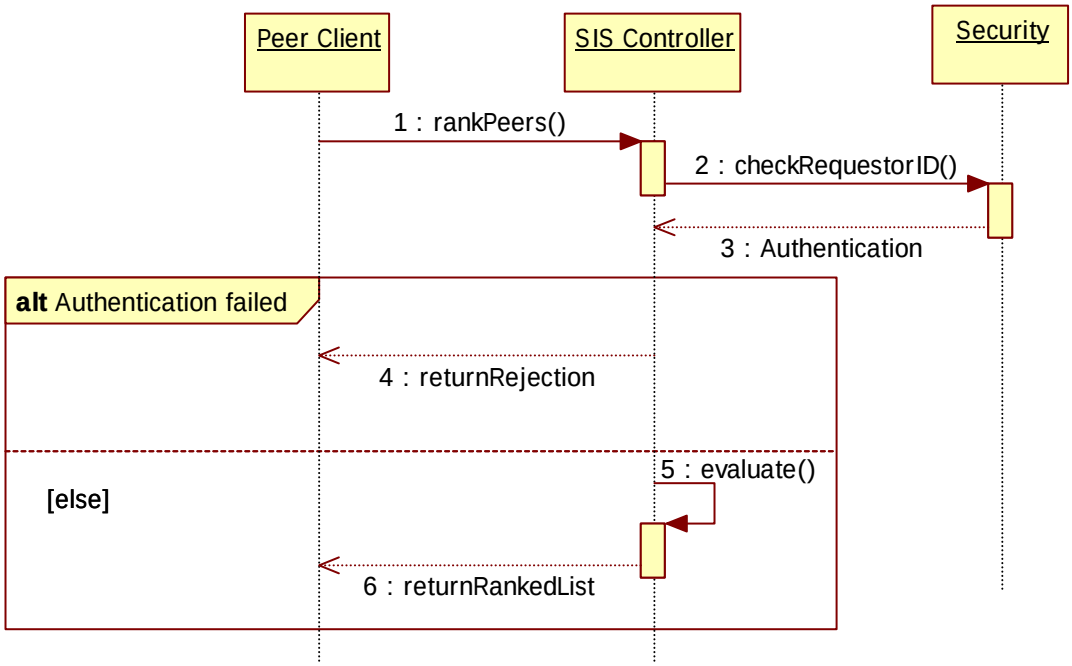


Figure 8: SIS Client request a sorted list of IP addresses

5.1.3.4 State Machine Diagrams

Figure 9 shows the state machine diagram of the SIS Controller to implement the BGP-Loc ETM mechanism:

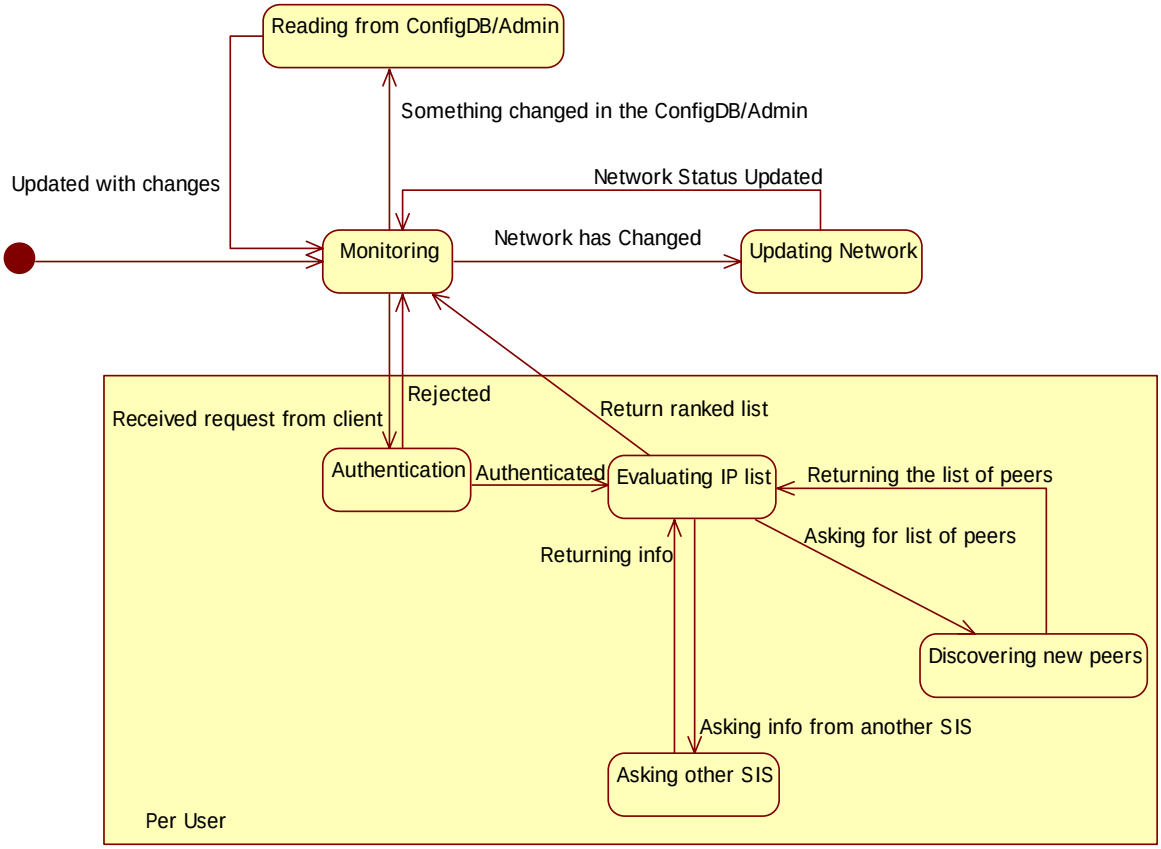


Figure 9: State machine diagram

5.1.3.5 Technology

The component is developed in J2EE and provides web services to SIS Client and InterSIS servers.

5.1.3.6 Parameters

The SIS Controller has the following parameters:

- For the ETM algorithm the component needs to know:
  - Which ETM algorithm to use
  - The list with the IP addressed that must be sorted
  - upload/download bandwidth for the user
  - A threshold for the SIS preference, after which the new IP is suggested
- Information about the network's state:
  - POP properties
    - upload bandwidth
    - download bandwidth

    - delay
    - reliability
  - Distribution of P2P users over POPs
  - Traffic share of P2P users
  - Size distribution for POPs, Distribution of peers on AS/POPs
  - Number of users total
  - Number of POPs (Access Networks) / AS
  - Link utilizations
  - Bandwidths Access networks
  - inter-AS links
  - Delay of inter-AS links
  - background traffic AN/Intra/Inter-AS
  - Inter AS traffic
  - Peering charging schemes.
- Information about additional sources:
  - A list with the (external) SIS URLs that can be used to acquire extra info

5.1.4 Customization for trials

The component must be independent of the trials form. The same configuration must be used for both internal and external trials.

5.1.5 Testing

There are stand alone tests for the component which are integrated with the Ant scripts. The tests cover more than 90% of the component's code. Only the tests for the inter-sis communication require having the SIS already deployed to JBoss.

All tests are integrated with the Ant scripts.

5.1.5.1 Scenario: *Communication with SIS Clients*

A client sends the list with the IP addresses of the peers that he can connect to and this component has to sort them.

Test case	Initial Conditions	Input	Expected Result
1. Client requests updated SIS Controller	The system has already been updated with the network status	A list of IP addresses	A sorted list of the aforementioned IP addresses
2. Client requests non updated SIS Controller	The system has not been updated with the network status	A list of IP addresses	A sorted list of the aforementioned IP addresses

3. Client sends incorrect list	The system has not been updated with the network status	A list of incorrect IP addresses	An error message

5.1.5.2 Scenario: *Network Status Retrieval*

Test case	Initial Conditions	Input	Expected Result
1. Updating network status	The system has already been updated with the network status	None	The component acquires the state of the network
2. Updating networks status for the first time	The system has never been updated with the network status	None	The component acquires the state of the network
3. Unable to update network status	The system has already been updated with the network status and currently one or more sources are off-line	None	New data overwrites the old data and the component uses the old state to fill any blanks
4. Unable to learn the network status	The system has never been updated with the network status and currently one or more sources are off-line	None	The component uses some default

5.1.5.3 Scenario: Request to Another (External) SIS

Test case	Initial Conditions	Input	Expected Result
1. Contacting on-line external SIS	The external SIS is on-line	A list with the SIS URLs	Receives the info
2. Contacting off-line external SIS	The external SIS is off-line or responses some errors	A list with the SIS URLs	The SIS Controller computes the outcome without this extra info

5.1.5.4 Scenario: Response to (External) SIS

Test case	Initial Conditions	Input	Expected Result
1. Contacting on-line external SIS	The external SIS is on-line and this SIS has the info	None	Sends the info
2. Doesn't have the requested info	The external SIS is on-line and this SIS hasn't got the info	None	Sends the info
2. Contacting off-line	The external SIS is off-line	None	Sends no info

external SIS			

5.1.5.5 Scenario: Parameter Update from SIS DB (or Admin)

Test case	Initial Conditions	Input	Expected Result
1. Updating for the first time	The component is just started	None	Loads all the parameters from SIS DB (or Admin)
2. Updating	The component is already running	Command to reload the component's parameters	Loads all the parameters from SIS DB (or Admin)
3. Receiving invalid parameter	The component is already running and the parameter is invalid	None	The component informs about the error at the specific parameter

5.1.5.6 Scenario: *Update User QoS*

Test case	Initial Conditions	Input	Expected Result
1. Updating QoS with a valid value	A QoS parameter has to be changed	The parameter and its new (valid) value	The parameter must be set
2. Updating QoS with an invalid value	A QoS parameter has to be changed	The parameter and its new (invalid) value	An error message describing what is wrong
3. Updating an non existing parameter in QoS	A QoS parameter has to be changed	The (not existing) parameter and its new (valid) value	An error message describing what is wrong

5.1.5.7 Scenario: *Performance Test*

Test case	Initial Conditions	Input	Expected Result
1. Testing the SIS Controller's performance	The component has been loaded correctly and has been updated with the network status	Lists of IP addresses (from different users –some lists are incorrect)	Sends back the sorted lists
2. Testing the SIS Controller's behavior at startup	The component has been loaded correctly, but has not been updated yet with the network status	Lists of IP addresses (from different users –some lists are	Sends back the sorted lists

		incorrect)	

5.1.5.8 Scenario: Networks State Update Notification Received from Metering

Test case	Initial Conditions	Input	Expected Result
1. Receiving notification from Metering	The component has already been updated	None	The component acquires the state of the network
2. Receiving notification from Metering for the first time	The component has never been updated	None	The component acquires the state of the network

5.1.5.9 Scenario: Get Swarm Info from Overlay Tracker

Test case	Initial Conditions	Input	Expected Result
1. Getting info from an on-line tracker	The tracker is on-line	Tracker's IP	The component acquires the info
2. Getting info from an off-line tracker	The tracker is off-line	Tracker's IP	The component doesn't acquire the info and returns a proper message

5.2 SIS Database

The SIS Database (SIS DB) is persistent storage for the server-side SIS components. It disburdens other components from managing their own configuration. Instead, all components can manage their data, in the SIS Database, in a transparent way. The component further provides information about the topology of the autonomous system of the given ISP (since it is assumed that SIS is run by an ISP), such as the own IP ranges and the properties of inter-domain links to other ISPs. This component only offers services (via interfaces) to other components and does not rely on the existence of other components. Therefore, other SIS components will be referred to as *client components* in this subsection, since they utilize services of this component.

The component design follows the patterns of Data Access Objects (DAO) and Data Transfer Objects (DTO) as a common praxis in Java to offer a transparent access to a relational backend. DTOs represent entities and are usually mapped to database tables, while DAOs encapsulate the access and query complexity.

The component is designed in such a way that it can offer different implementations transparently. For testing and debugging purposes a memory-only solution is available. For the test-bed and the external trial a persistent implementation with a relational backend will be used.

5.2.1 Requirements

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1.	Provide the SIS components with the manageable configuration. For example, the maximum allowed request rate per peer can be stored here.	X	
F.2.	Provide information about other known ISPs. This includes: relationships (provider/peer/customer), inter-AS link capacity, inter-AS delay.	X	
F.4	The component shall be able to store topology information of the test-bed. Additional properties (peering agreements) can be set.	X	
F.5	For the external trial the component shall communicate with the ISP's current databases to obtain relevant information (peering agreements, link properties…).		X

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Easy deployment: Must be adjustable for usage in internal and external trials. Different modes must be available:	X	

	• Static configuration for the internal trial • Customizable for the external trial		
N.2	Security: The access to the database content will be restricted to authorized clients (SIS Controller, Admin, and Metering).	X	
N.3	Extensibility: It shall be easy to extend the component with additional data that might be required by ETM mechanisms.	X	
N.4	Scalability: SIS DB should be able to handle sufficient number of requests (large user population might produce many SIS requests for IPs from multiple ASes).	X	
N.5	Standard compliance: component shall use standard protocols and formats.	X	
N.6	Efficiency: component shall utilize storage space efficiently and answer queries with low delay.	X	

5.2.2 Offered Interfaces

This component offers the following interface:

5.2.2.1 Interface db.1

This interface offers the common way to store and access configuration data by other components. In the first place it provides information about the IP ranges owned by the local ISP and other ISPs together with link properties to neighbor ISPs. Additionally it offers a generic way to store configuration data of single components. The interface encapsulates access to a database backend and offers a Java API to other components.

The interface consists of four parts:

1. `ConfigDAO` interface is the main interface to be used by other components to access and update the database content.
2. `AbstractDAO` is an abstract implementation of the `ConfigDAO` interface and can be additionally used to create concrete implementations.
3. `LinkProperties` encapsulate the information about the link between two ISP domains.
4. `IPRange` represent a range of IP addresses owned by an ISP.

The **ConfigDAO** interface is used to retrieve and store configurations by offering the following methods:

- `public List<IPRange> getLocalIPRanges();` Retrieve all IPRanges, that belong to the local ISP
- `public void addLocalIPRange(IPRange range);` Add an IPRange to the local ISP

- `public List<IPRange> getRemoteIPRanges()`: Retrieve all `IPRanges` that belong to remote ISPs
- `public void addRemoteIPRange(IPRange range)`: Add an `IPRange` to the remote ISPs
- `public LinkProperties getLinkProperties(String localIP, String remoteIP)`: Retrieve Properties assigned to the link from localIP to remoteIP
- `public void addLinkProperties(IPRange src, IPRange dest, LinkProperties props)`: Assign Properties to the link from localIP to remoteIP
- `public Serializable getProperty(String component, String propertyName)`: Retrieve a property Object stored for *component*
- `public String getStringProperty(String component, String propertyName)`: Retrieve a property String stored for *component*
- `public void addProperty(String component, String propName, Serializable value)`: Store a property Object for *component*
- *`public void addStringProperty(String component, String propName, String value)`:* Store a property String for *component*
- `public void clear()`: Delete all configurations
- `public void importConfig(File configFile) throws IOException`: Import configurations from a file

The factory methods of the `AbstractDAO` class, used to create configuration DAOs are:

- `public static ConfigDAO createConfig()` Creates and returns a concrete `ConfigDAO` containing all configurations stored in the default config file (`setup.conf`)
- `public static ConfigDAO createConfig(File configFile)`: Creates and returns a concrete `ConfigDAO` containing all configurations stored in the given config file

The methods of the `LinkProperties` interface are:

- `public double getCapacity()`: Returns the capacity of a link
- `public double getDelay()`: Returns the delay of a link
- `public String getPolicy()`: Returns the policy/relation String of a link

The methods of the `IPRange` interface are:

- `public String getPrefix()`: Returns the Prefix of an `IPRange`

- `public int getPrefix_len()`: Returns the Prefix length of an `IPRange`

The following constructors can be used to create own Transport Objects in order to store configurations in the DB:

- `public ConfigEntry(String relation, double delay, double link_capacity)`: Creates a `LinkProperties` Transport Object
- `public IPRangeConfigEntry(String prefix, int prefix_len)`: Creates an `IPRange` Transport Object

5.2.3 Component Design

The component design follows the patterns already mentioned in the beginning of this section:

- Data Transfer Object (DTO) [dto],
- Data Access Object (DAO) [dao].

The component offers two kinds of functions:

- Custom configuration describing the network topology of the given ISP and the relationships to neighbor ISPs. Here, it covers only the relevant aspects at the level of autonomous systems. The values can be loaded from configuration files, modified via addXY()-methods (e.g. for use by the Admin interface later on), and read via getXY() methods.
- Generalized functionality allows for storing arbitrary configuration values. To prevent name clashes, component names define virtual namespaces. Any serializable objects can be stored in the database.

The following subsections describe the structural view of the component and its behavior.

5.2.3.1 Structural View

The client components can access the database content via the `ConfigDAO` interface of db.1. The data itself is represented via interfaces `LinkProperties` and `IPRange` depicted in Figure 10.

The actual implementation is hidden from the client components. Different implementations (sharing the same abstract implementation in `AbstractDAO`) are available:

- `MemoryDAO` is the basic non-persistent implementation suitable for testing and integration purposes.
- `PersistentDAO` is the persistent implementation based on Java Persistent API (JPA) [jpa].

Figure 10 shows all relevant classes of the SIS DB component. The additional `PropsConfigEntry` class encapsulates arbitrary configuration settings and can be used by client components to store and retrieve their settings, without modifying the SIS DB interfaces and implementation process. Arbitrary objects (implementing `Serializable` interface) can be stored here.

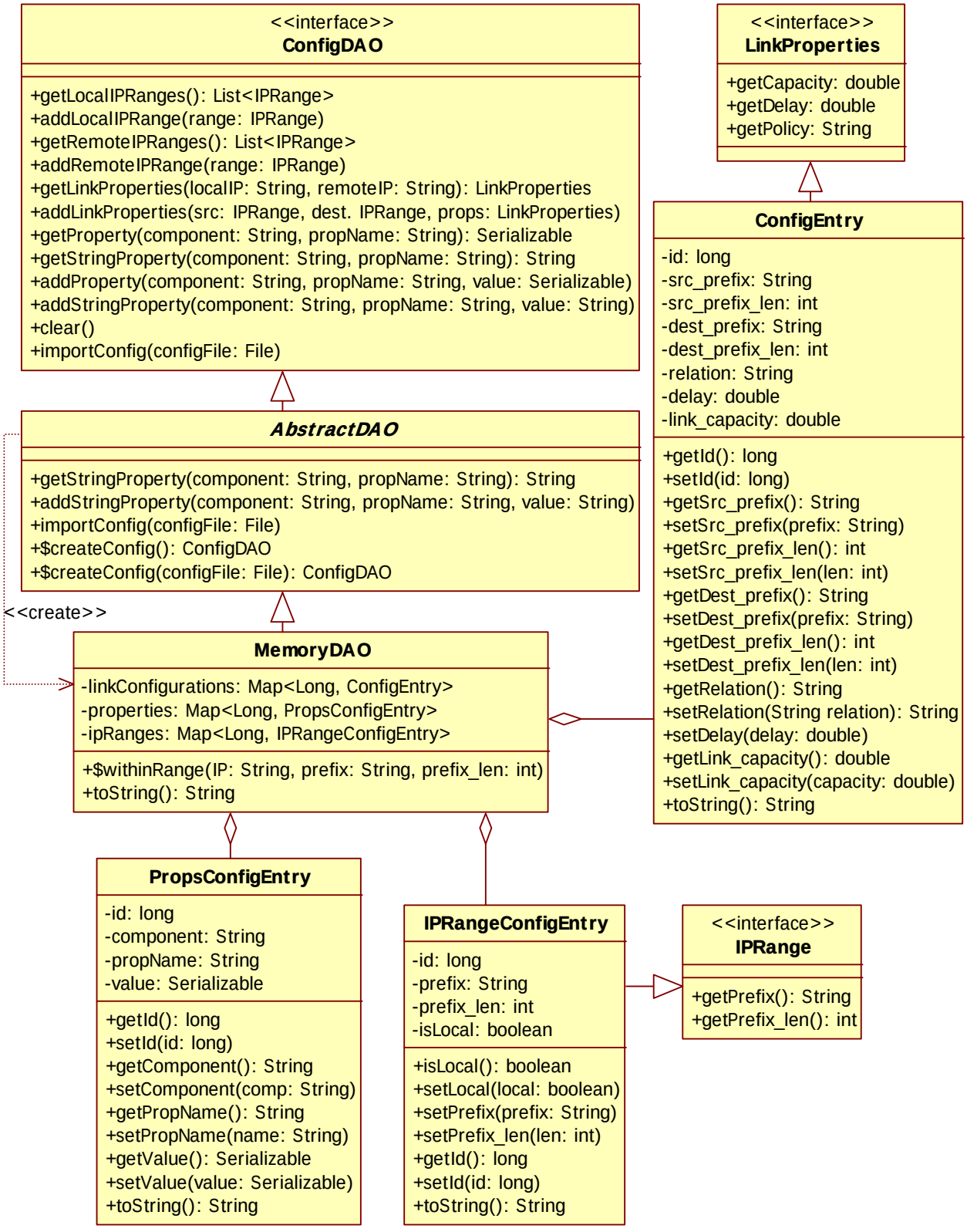


Figure 10: Class diagram of SIS DB

5.2.3.2 Behavioral View

Figure 11 shows the creation process of a concrete DAO implementation and the import of a configuration file.

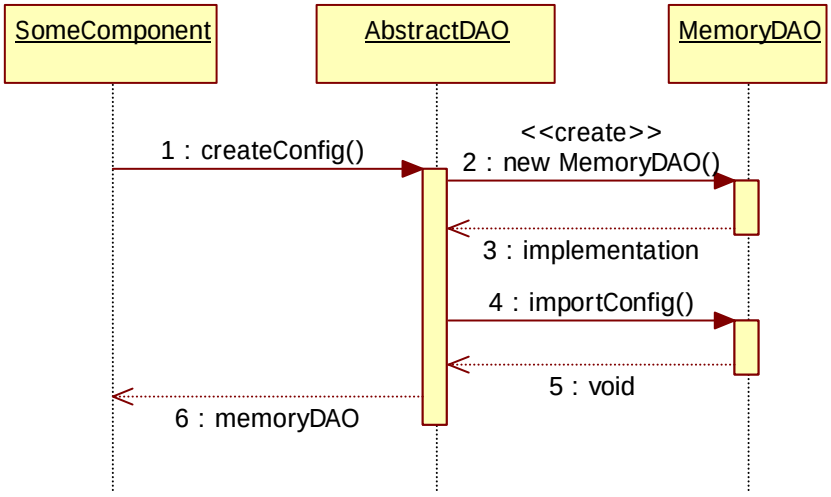


Figure 11: Sequence diagram of SIS DB

5.2.3.3 Technology

SIS DB uses Java Persistence API (JPA) to access the underlying SQL database. Hybernate [hybernate] is used as JPA Provider and HSQLDB [hsqldb] as SQL database. Because this database comes as part of JBoss, it can be easily used for developing, testing, and debugging purposes. If there are any scalability issues as a switch can be made to the more optimized MySQL DB, for the external trial and if necessary also for the internal trial. This switch will be transparent for other components.

The benefits of using JPA to access the database are:

- SQL database can reside on the same or different server (or even be replicated on several servers). This allows transparency for the client.
- Inside of the SIS server, the SIS DB can be accessed via Plain Old Java Objects (POJO) unifying the Data Transfer Object (DTO) and Data Access Object (DAO) design patterns, to further ease code organization.

There is an additional implementation for test purposes that reads configuration data from a configuration file and stores the DB content in memory only. It is further possible to populate the database with default content from a configuration file or set its content via setXY() methods (e.g. to be used by the Admin component). The flexible design makes the distinction evident to client components.

5.2.4 Parameters

The component can be parameterized as follows:

- It is possible to select the persistent or non-persistent mode (either `MemoryDAO` or `PersistentDAO`). This configuration is done during SIS initialization. The expected usage is that the SIS Controller will decide on the implementation to be used and provide the reference to other components.
- For persistent implementation, it is also possible to specify which SQL database backend is used (e.g. HSQLDB running as part of JBoss, or MySQL DB running on the same or remote machine). This configuration is done via configuration files (`persistence.xml`). Fine-grained parameterization can be used to adjust handling of transactions, cache setting etc.

5.2.5 Customization for trials

Since the topology of the internal test-bed will be known in advance, most of the information stored in the database will be static. Therefore, it will be sufficient to load content from configuration files. Additionally, it will be possible to setup the configuration using the setup API, e.g., by init scripts.

The extent of the integration with PrimeTel's equipment is not completely clear at this stage of the development. Therefore, the component offers a transparent API to the SIS server and residing ETM mechanism without bothering them with how the data is read into the database. The only expected modification is the potential usage of external SQL database already deployed by PrimeTel.

5.2.6 Testing

The tests can be run without JBoss server. They make use of the JUnit framework in version 4.0 [junit] and are invoked via ant files (same as for other SIS components).

The following JUnit tests are used to test the correct functioning of the component:

5.2.6.1 Scenario: Simple Memory DAO Tests

A simple tests that creates a `MemoryDAO`, inserts and deletes values.

Test case	Initial Condition	Input	Expected Result
1. Simple store-retrieve test	None.	IP ranges, component, and link properties to store	Properties are stored and retrieved successfully

5.2.6.2 Scenario: Extended Memory DAO Tests

Advanced tests of a DAO (file import, update, reset, etc.)

Test case	Initial Condition	Input	Expected Result
1. DAO creation	None	None	Checks if the creation was successfully, i.e. the database exists

2. Default values	None	None	Checks that getters return default values, if no configuration file is loaded and no fields are modified via `addXY`() methods.
3. Load values from file	None	Configuration file name	Creates a new `MemoryDAO` and imports initial setup from the `setup.conf` file. Also asserts that this data is actually contained afterwards.
4. Update existing values	None	New values for IP ranges and component properties	Check whether those new versions of values are returned by the component.
5. Non-persistence	None	None	Recreates configuration and asserts that the data is gone afterwards.
6. New value insertion	None	New values for IP ranges, component, and link properties	New values are added to the configuration. Asserts that this data is contained afterwards.
7. Reset	None	Clear command.	Asserts that all changes are gone.

5.3 Metering

The Metering component is responsible for collecting different kinds of information (i.e. metering data) from the network and providing metering data to other components of the SmoothIT architecture. Metering data can be used by the ETM mechanisms and implemented in the different SIS components, to assist overlay applications in their peer selection process. The Metering component provides BGP information to the SIS Controller in its first version. In later versions, the Metering component can include other functions as well, such as aggregating metering data, performing performance measurements, or metering the amount of data transferred between neighboring ISPs. The future functionality of the Metering component depends on the ETM mechanism to be integrated in the SmoothIT architecture.

The current version of the Metering component can provide BGP routing information to the SIS Controller. To this end, it accesses BGP routers and/or route reflectors in the ISP's network to read the BGP routing table and provides this information to the SIS Controller. The BGP routing information is required by the BGP-based Locality Promotion ETM mechanism. The routing table is read by the Metering component over SNMP. Additionally, the BGP Information Module can read BGP information from a file in case no real routers are available, e.g., in a test-bed or in the internal trial, where topologies will be emulated.

Future versions of the Metering component will be used to support other ETM mechanisms as well, such as Locality-aware, Tit-for-Tat/Unchoking, or IoP with locality-awareness. This can include, for example, functionality to meter traffic over inter-domain links or meter network performance metrics.

5.3.1 Requirements

The functional and non-functional requirements of the current version of the Metering component are listed in the following. Further requirements might be identified if new ETM mechanisms are integrated to the SmoothIT architecture.

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1.	Provide BGP information either from routers or from a file.	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Easy deployment: the Metering component shall be easy to be deployed in the network of an ISP.	X	
N.2	Standard compliance: the Metering component shall read routing tables from routers over SNMP.	X	
N.3	Scalability: The Metering shall be scalable in terms of number of routing entries.	X	

N.4	Efficiency: The operation of the Metering shall be efficient in terms of memory consumption, and processing requirement.	X	

5.3.2 Offered Interfaces

The interface offered by the Metering component is described in the following.

5.3.2.1 Interface meter.1

This interface enables the ability to read the BGP routing table from a router over SNMP or the ability to read BGP routing entries stored in a file. It allows setting the IP address and porting number of the router, the SNMP community, and the name of the file. After reading and loading the routing entries (either from a router or from a file) into memory, the interface permits the retrieval of the number of routing entries, all routing entries, or specific entries corresponding to an IP address.

The interface provides the following methods:

- `public void setRouterAddress(String a)`: Sets the IP address of the router where the BGP routing table is read from.
- `public void setRouterPort(int p)`: Sets the port number of the router where the BGP routing table is read from.
- `public void setSNMPCommunity(String c)`: Sets the SNMP community string for reading the BGP routing table from a router.
- `public void setFileName(String f)`: Sets the name of the file where the BGP routing table is read from.
- `public void useRouter()`: Specifies to use a router to read the BGP routing table from.
- `public void useFile()`: Specifies to use a file to read the BGP routing table from.
- `public boolean readRoutingTable()`: Loads the BGP routing table into memory. The routing table is either read from a router or from a file. Whenever this method is called, the routing table is reloaded. Returns true if the routing table was loaded successfully, false otherwise.
- `public Vector<BGPRoutingTableEntry> getAllEntries()`: Returns the complete BGP routing table.
- `public BGPRoutingTableEntry getEntryAtIndex(int i)`: Returns the routing entry at index i in the BGP routing table.
- `public int getNumberOfEntries()`: Returns the number of routing entries in the BGP routing table.
- `public BGPRoutingTableEntry getEntryForPrefix(String ip)`: Returns the routing entry that represents the route to the destination IP address specified in the argument.

- `public void print()`: Prints the complete BGP routing table to the logger. It can be used for debugging.

5.3.3 Utilized External Interfaces

5.3.3.1 Interface net.1

The Metering component uses the interface net.1 of the network to read the BGP routing table from a BGP router or route reflector. Interface net.1 is based on the SNMP protocol and uses the standard BGP4 MIB [RFC4273] to read the routing table.

5.3.4 Component Design

The Metering component gathers information from the network and provides this information to other components of the SmoothIT architecture. Its structural and behavioral views are described in the following.

5.3.4.1 Structural View

The class diagram of the Metering component is shown in Figure 12. It includes the following classes:

- The `BGPRoutingTable` interface class specifies the meter.1 interface.
- The `BGPRoutingTableReader` class implements the BGPRoutingTable interface and enables to read BGP routing entries either from a router or from a file.
- The `BGPRoutingTableEntry` class represents a single BGP routing entry and it is used by the `BGPRoutingTableReader` class to store the routing table.

The `BGPRoutingTable` interface class is in the `eu.smoothit.sis.metering.bgp` package, while the `BGPRoutingTableReader` and `BGPRoutingTableEntry` classes are in the `eu.smoothit.sis.metering.impl` package.

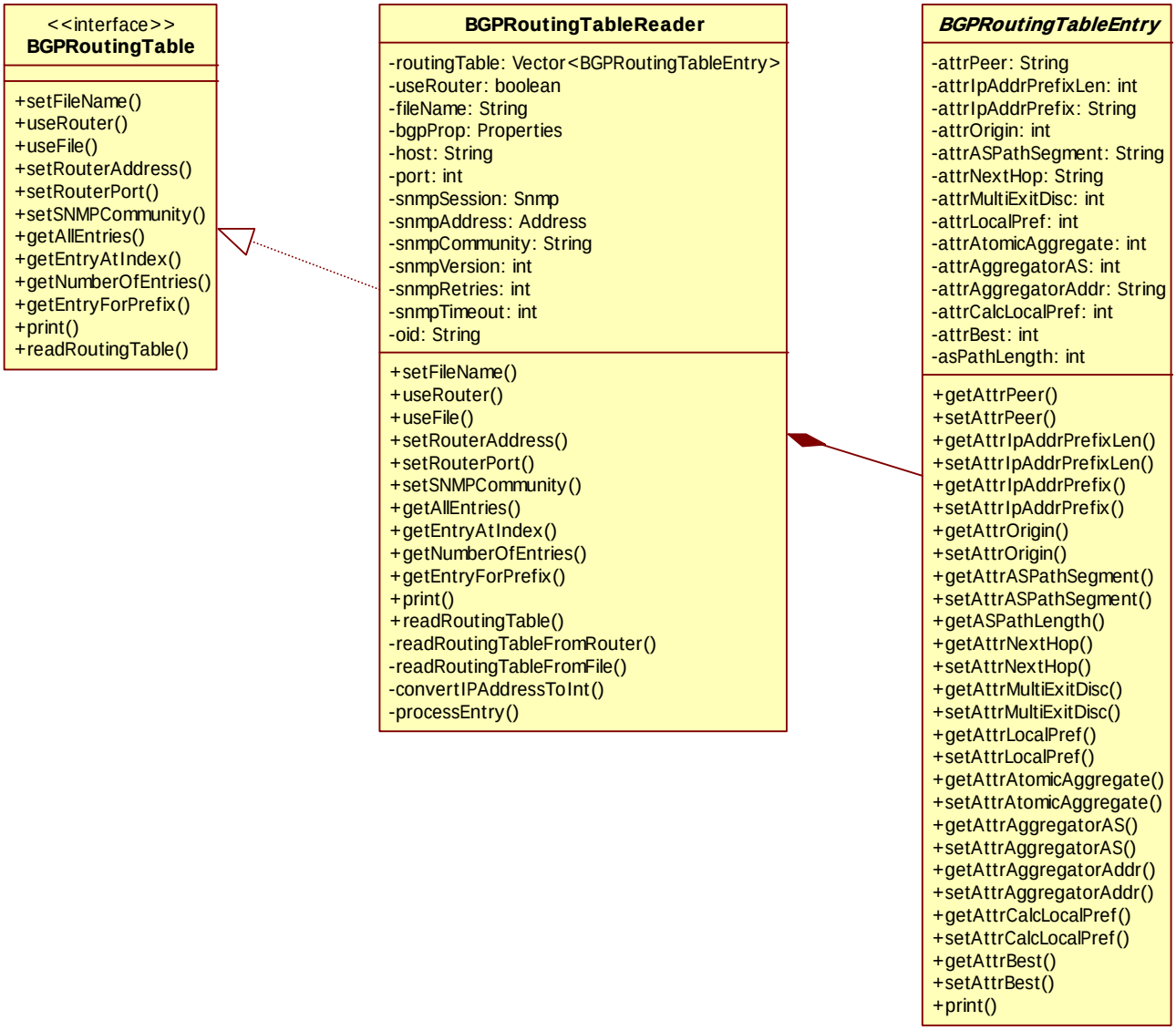


Figure 12: Class diagram of the Metering component

The `BGPRoutingTableReader` can read the routing entries from a BGP configuration file. The configuration file is a text file and contains a sequence of BGP routing entries. A single BGP routing entry is represented with the structure depicted in Figure 13, where `<index>` is an integer number starting from 1 and it indexes the routing entries.

```
<index>.bgp4PathAttrPeer = <bgp4PathAttrPeer>
<index>.bgp4PathAttrIpAddrPrefixLen = <bgp4PathAttrIpAddrPrefixLen>
<index>.bgp4PathAttrIpAddrPrefix = <bgp4PathAttrIpAddrPrefix>
<index>.bgp4PathAttrOrigin = <bgp4PathAttrOrigin>
<index>.bgp4PathAttrASPathSegment = <bgp4PathAttrASPathSegment>
<index>.bgp4PathAttrNextHop = <bgp4PathAttrNextHop>
<index>.bgp4PathAttrMultiExitDisc = <bgp4PathAttrMultiExitDisc>
<index>.bgp4PathAttrLocalPref = <bgp4PathAttrLocalPref>
<index>.bgp4PathAttrAtomicAggregate = <bgp4PathAttrAtomicAggregate>
<index>.bgp4PathAttrAggregatorAS = <bgp4PathAttrAggregatorAS>
<index>.bgp4PathAttrAggregatorAddr = <bgp4PathAttrAggregatorAddr>
<index>.bgp4PathAttrCalcLocalPref = <bgp4PathAttrCalcLocalPref>
<index>.bgp4PathAttrBest = <bgp4PathAttrBest>
<index>.bgp4PathAttrUnknown = <bgp4PathAttrUnknown>
```

Figure 13: BGP configuration file format

For a routing entry the attributes `bgp4PathAttrIpAddrPrefixLen`, `bgp4PathAttrIpAddrPrefix`, and `bgp4PathAttrASPathSegment` are mandatory. The other attributes are optional and do not have to be listed in the configuration file.

The following example (Figure 14) defines two routing entries, the first entry specifies all BGP attributes, while the second one specifies only the mandatory attributes:

```
1.bgp4PathAttrPeer = 192.168.1.1
1.bgp4PathAttrIpAddrPrefixLen = 16
1.bgp4PathAttrIpAddrPrefix = 192.168.0.0
1.bgp4PathAttrOrigin = 0
1.bgp4PathAttrASPathSegment = 02:02:00:cd:00:c8
1.bgp4PathAttrNextHop = 192.168.1.1
1.bgp4PathAttrMultiExitDisc = 100
1.bgp4PathAttrLocalPref = 200
1.bgp4PathAttrAtomicAggregate = 1
1.bgp4PathAttrAggregatorAS = 0
1.bgp4PathAttrAggregatorAddr = 0.0.0.0
1.bgp4PathAttrCalcLocalPref = -1
1.bgp4PathAttrBest = 2

2.bgp4PathAttrIpAddrPrefixLen = 16
2.bgp4PathAttrIpAddrPrefix = 192.100.0.0
2.bgp4PathAttrASPathSegment = 02:02:00:cd:00:c8
```

Figure 14: BGP configuration file example

5.3.4.2 Behavioral View

The sequence diagram for reading the routing table from a router is shown in Figure 15. The SIS Controller (or any other component using the interface meter.1) can set the IP address of the router, the port number, etc., according to the interface specification. Using the `readRoutingTable()` method the routing table is read from the router via SNMP. Afterwards, the SIS Controller can read routing entries from the Metering component using the methods specified for the interface meter.1.

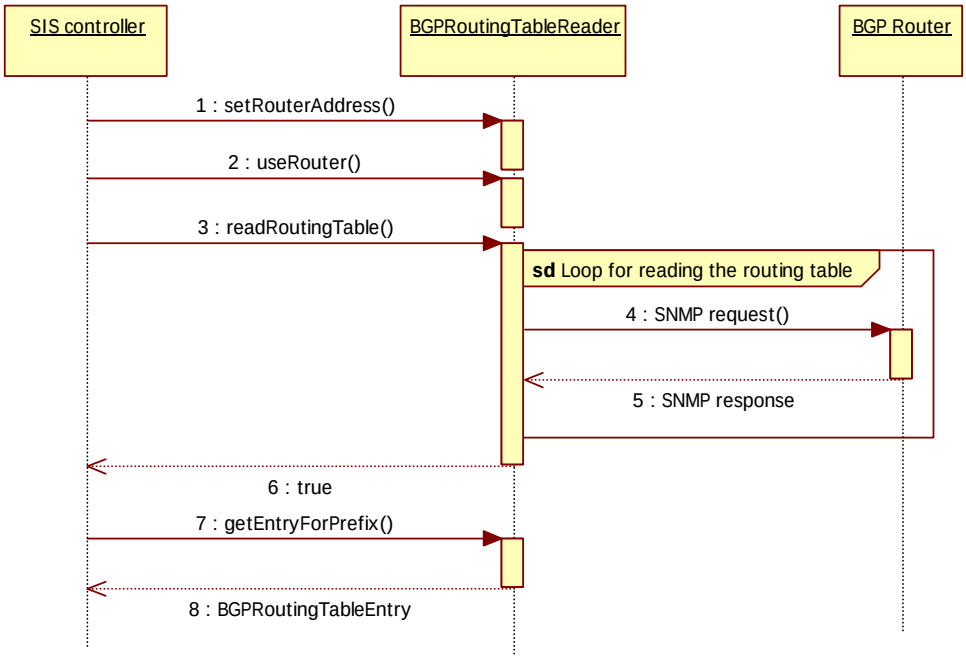


Figure 15: Reading routing table from a router

The sequence diagram for reading the routing table from a file is shown in Figure 16. The SIS Controller (or any other component using the interface meter.1) can set the name of the file according to the interface specification. Using the `readRoutingTable()` method the routing table is read from the file. Afterwards the SIS Controller can read routing entries from the Metering component using the methods specified for the interface meter.1.

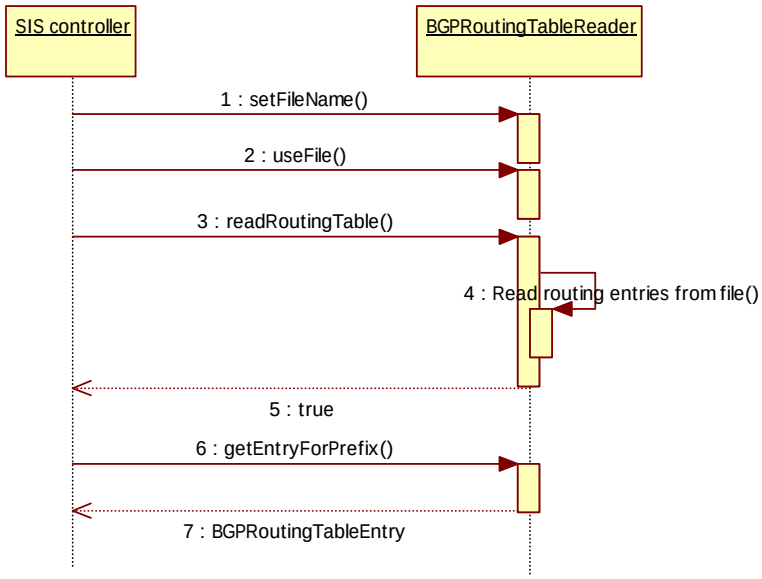


Figure 16: Reading routing table from a file

5.3.4.3 Technology

The Metering component is implemented in Java. It uses the SNMP4J library [SNMP4J] for the SNMP-based communication. The SNMP4J library provides an SNMP API to communicate over SNMP.

5.3.4.4 Parameters

The Metering component has the following parameters that are currently provided by the SIS Controller and will be stored in the SIS DB in the next releases:

- The IP address and port number of the router.
- The SNMP community string.
- The name of the BGP configuration file.
- Reading the routing table from a router or a file.

5.3.5 Testing

The tests can be run without JBoss server. They are invocated via ant files. To test the reading of the routing table from a router, a BGP router has to be available (e.g., based on quagga [quagga]) and it has to be configured to have some BGP routing entries.

5.3.5.1 Scenario: BGPRoutingTableEntry Test

The testing of the `BGPRoutingTableEntry` class includes the setting and getting all BGP attributes.

Test case	**Initial Condition**	**Input**	**Expected Result**
1. Set-get test	None.	One of the BGP attributes (set one of the BGP attribute with the corresponding set method).	The BGPRoutingTableEntry sets the value of the appropriate BGP attribute.

5.3.5.2 Scenario: BGPRoutingTableReader Test

The testing of the `BGPRoutingTableReader` class includes the reading of the routing table from a router and a file and different error conditions.

Test case	**Initial Condition**	**Input**	**Expected Result**
1. Non-existing conf. file	None.	Name of a non-existing BGP configuration file.	`readRoutingTable()` returns false and the routing table is empty.
2. Empty conf. file	Empty BGP configuration file.	Name of the empty configuration file.	`readRoutingTable()` returns true and the routing table is empty.
3. Conf. file with errors	BGP configuration file with syntax errors.	Name the configuration file.	`readRoutingTable()` returns false and the routing table is empty.
4. Read from file	BGP configuration file.	Name of the configuration file.	`readRoutingTable()` returns true and the routing table contains all entries from the file.
5. Get routing entry	Routing table is already read.	Get a certain routing entry from the routing table.	The routing entry.
6. Non-existing router	None.	IP address of a non-existing BGP router.	`readRoutingTable()` returns false and the routing table is empty.
7. Read from router	None.	IP address of an existing BGP router.	`readRoutingTable()` returns true and the routing table contains all entries from the router.

5.4 Admin Component

The Admin component is used by the system administrators to monitor the system operation and set relevant parameters. For example, an administrator may want to update the information on relationships with other ISPs or data related to the P2P application operation, such as maximum allowed peer request rate. Thus, it is an interface between the system administrator and the system.

The Admin component is connected with the SIS DB and the SIS Controller. Thus, it must be able to set and retrieve the relevant set of parameters of these two components.

5.4.1 Requirements

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1.	Serve as an interface between the administrator and the configuration DB. This means that the Admin component must provide means for the administrator to query and update the SIS DB.	X	
F.2.	Serve as an interface between the system administrator and the SIS Controller. This means that the Admin component should provide a means for the system administrator to monitor the operation of the SIS Controller and, if needed, set its relevant parameters.	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Ease of use: The Admin component should offer a GUI that makes it easy for the system administrator to set the parameters of the system and monitor its operation	X	
N.2	Extensibility: It shall be easy to extend the component with additional functionality and features	X	

5.4.2 Offered Interfaces

The Admin component will not offer any programmable interface to any other component in the system. It will use the interfaces offered by other components (the SIS Controller and the SIS DB) but will not be used by any other component in the system.

5.4.3 Component Design

Implementation of the Admin component will be provided later. Therefore, full details with respect to its design are not available yet. At this point, it is believed that a simple JSP-based web page will suffice. It should contain a set of fields through which the administrator can set relevant parameters, such as a list of ISPs with peering agreements between the current ISP or properties of the current ISP's network (e.g., changes in the topology, addition of new access links and so on).

5.5 Security Component

The Security component is responsible for security assurance in the SmoothIT architecture. It is connected directly to the SIS Controller through the security interface which provides the following services:

- authentication
- authorization
- accounting

Other services that could be implemented in SmoothIT architecture are the encryption of messages (confidentiality assurance) and messages signing (data integrity assurance). It is worth considering the efficiency of the SIS server, which can be affected by the deployment of an encryption service. If the performance and the functionality of the SIS system is sufficient, the encryption of messages will be implemented.

5.5.1 Requirements

This section describes the security requirements of the SIS server.

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1	Inter-domain support: The SIS deployed in different ISPs shall be able to communicate with each other in a secure way. SIS elements in different ASes may communicate with each other after successful authentication. Additionally, the communication between SIS servers will be protected by means of data integrity algorithms.	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Security: Only authorized entities should be able to access and modify the configuration data. There should be at least two different levels of security: the manager, who is able to modify the configuration; and the user, who is able to access the configuration data. Also, message integrity service will be deployed (against masquerade attack).	X	
N.2	Extensibility: The Security component shall be extendible to support new algorithms and security services.		X
N.3	Robustness: The SIS server shall be robust against malicious behavior and against active and passive attacks.	X	
N.4	Standard compliance: The Security component shall use and be based on standard protocols where applicable.	X	
N.5	Efficiency: The operation of the Security component	X	

	shall be efficient in terms of communication (bandwidth) overhead, storage consumption, and processing requirements. Only efficient algorithms (hash functions, authentication methods) will be implemented.		

5.5.2 Offered Interfaces

Security component offers the interface sec.1. It provides authentication, authorization and accounting services for other SIS components.

5.5.2.1 Interface sec.1

The interface between the SIS Controller and Security component is Java-based and offers the following methods:

- `public boolean checkRequestorID(String userName, String password)`: Check the password of the requestor (authentication service)
- `public boolean isCallerInRole(String userName, String roleName)`: Retrieve the access control rights of the user (authorization service)
- `public void addLogRecord(String userName, String event, String result)`: Add a new log record to the log file (accounting service)

5.5.3 Component Design

The ITU-T Recommendation X.805 “defines the general security-related architectural elements that when appropriately applied can provide end-to-end network security” [ITU-X.805]. This document includes the definition of eight security dimensions, which should be considered during a new network element design. Table 2, presents the security dimensions (also called: requirements) and SmoothIT external interfaces. Many possible solutions to protect external interfaces against threats are presented in detail. Sometimes more than one solution is possible — in that situation the propositions are numbered.

5.5.3.1 Security Dimensions

The ITU-T Recommendation X.805 “defines the general security-related architectural elements that when appropriately applied can provide end-to-end network security” [ITU-X.805]. This document includes the definition of eight security dimensions which should be considered during a new network element design. In Table 2 we present the security dimensions (we can also call them: requirements) and SmoothIT external interfaces. In detail, we present many possible solutions to protect external interfaces against threats. Sometimes more than one solution is possible — in that situation the propositions are numbered.

Table 2: Security requirements for external interfaces

		EXTERNAL INTERFACES			
		Client-SIS (sis.1)	**Admin-SIS (sis.2)**	**Network-SIS (net.1 and net.2)**	**SIS-SIS (sis.3)**
SECURITY DIMENSIONS / REQUIREMENTS	**Authentication**	1. Without authentication of client *(SIS will be an open service)* 2. Authentication of SIS server by means of SSL (verification by client) 3. RADIUS authentication of the clients	1. Login and password based on JBoss methods 2. List of admitted IP addresses 3. RADIUS authentication	1. Connection to NGN is based on certificates and list of accepted IP 2. Connection to BGP router is based on login and password	1. List of admitted IP addresses 2. RADIUS authentication
	Access control	1. Client does not have direct access to SIS server *(Client can only send the requests to SIS)* 2. Access control based on the IP address of the client (only clients from the ISP's network can query the SIS server of the ISP) 3. RADIUS authorization of the clients	1. Authorization based on JBoss – access without restrictions 2. RADIUS authorization	Authorization depends on used connections	1. Authorization based on JBoss – access with some restrictions 2. RADIUS authorization

	Non-repudiation	1. Accounting for all clients based on JBoss log files (username, IP, time/date, event name, result of action) 2. RADIUS accounting (username, time/date, event name, result of action)	1. Accounting service which records the admin identity and date of authorization 2. RADIUS accounting (username, time/date, event name, result of action)	Accounting depends on used connections	1. Accounting service which records the SIS identity and date of authorization 2. RADIUS accounting (username, time/date, event name, result of action)
	Data confidentiality, Privacy, Communication security	1. Without protection 2. Security based on SSL (messages from SIS server) 3. Encryption service will be based on WS-security services delivered by JBoss	Encryption service will be based on WS-security services delivered by JBoss	Encryption of messages to/from NGN is based on asymmetric ciphers (https)	Encryption service will be based on WS-security services delivered by JBoss
	Data integrity	1. Without protection 2. Data integrity based on SSL (messages from SIS server)	Data integrity service will be based on JBoss services	Integrity of messages to/from NGN is based on hash functions used in https	Data integrity service will be based on JBoss services
	Availability	Reliability assurance	Reliability assurance	Reliability assurance	Reliability assurance

The three first security requirements — authentication, access control, and non-repudiation — will be met by means of AAA (Authentication, Authorization, Accounting) server services. The question is: which solution will be better for SIS architecture? Below we present advantages and disadvantages of two AAA solutions.

The other security requirements — data confidentiality, privacy, communication security, and data integrity — will be met by means of encryption and signing provided by the WS-security services of JBoss. We consider the asymmetric cipher to protect the data: PKI (*Public Key Infrastructure*) and X.509 certificates. This solution is based on the mechanisms implemented with the SSL (*Secure Socket Layer*) technique.

The last security requirement — availability — will be considered from the reliability point of view. Consideration about reliability of SIS is the one special task of WP2 and will be presented in deliverable D2.4.

The main objective of the Security component is protection of the SmoothIT architecture against active and passive threats. Generally, outside interfaces are the sources of threats. With reference to the SmoothIT architecture, we can indicate the following outside interfaces:

- admin interface,
- inter-SIS interface,
- network interface,
- overlay application interface.

Presented mechanisms will be used by two external interfaces: sis.2 and sis.3. Some of presented security services will be implemented for sis.1 interface. The last interfaces — net.1 and net.2 — will be protected by means of other mechanisms (not by means of implemented services in Security component).

The admin interface and the inter-SIS interface will be protected against threats by means of services implemented in Security component. The network interface does not need ‘special’ protection from Security component because it will use ‘default’ mechanisms, i.e. community strings in SNMPv2 or encryption and message integrity in SNMPv3. Lastly, the overlay application interface also does not provide ‘special’ protection, i.e. the authentication of overlay clients. These additional security services worsen the efficiency of SIS. Therefore, additional security services will not be implemented in the overlay security interface.

As described below, the three first services (authentication, authorization, and accounting) can be implemented by means of an AAA server based on JBoss services or an AAA RADIUS server. In the following considers both solutions in detail.

5.5.3.2 AAA Server Based on JBoss services

This solution provides all services of an AAA server:

- Authentication

  We have to modify the configuration of some files: web.xml, jboss-web.xml, and login-config.xml. Additionally, we have to put the usernames and passwords into jmx-console-users.properties, jbossws-users.properties and messaging-users.properties files.

- Authorization

  In the jmx-console-roles.properties, jbossws-roles.properties and messaging-roles.properties files, the access control groups are assigned to the users.

- Accounting

  This service will be created specially for SIS. The java code will be localized in the package: `eu.smoothit.sis.security.account`. The information in the log-file consists at least of data about each event, username and effect of the event. The name of this file will be based on the date of the events and will have the following format: *yyyy-mm-dd* (where *yyyy* is year, mm is month and *dd* is day).

5.5.3.3 AAA RADIUS Server

If an AAA RADIUS (Remote Authentication Dial In User Service) server is implemented, the following will be taken into account:

- **JRadius**, which is licensed under the LGPL (GNU Lesser General Public License) and partially GPL (GNU General Public License),
- **TinyRadius**, which is released under the terms of the LGPL (GNU Lesser General Public License).

Because both solutions have advantages (AAA server based on JBoss services and AAA RADIUS server), we have decided that the protection of SIS system will be twofold. The basic security will be provided by an AAA server, which is based on JBoss mechanisms: authentication, authorization, and accounting. The second protection will be an AAA RADIUS server which also provides authentication, authorization, and accounting services, but the services and communications will be standardized with RFC documents (i.e. RFC 2865). These two different solutions are motivated by performance and scalability of the SIS system. Because the higher security level always degrades the system performance, these two different security models are the best way to meet the user requirements regarding the functionality of the system. The section 'Appendix A.3 Security' includes more details about AAA servers.

When a RADIUS server was considered, it was decided that the JRadius would be the better solution, because it provides all AAA RADIUS server functionalities defined in the RFC documents. The JRadius is based on the FreeRADIUS implementation, so both servers have to be deployed (JRadius is not a stand-alone RADIUS server). The overall JRadius architecture is presented in Figure 17.

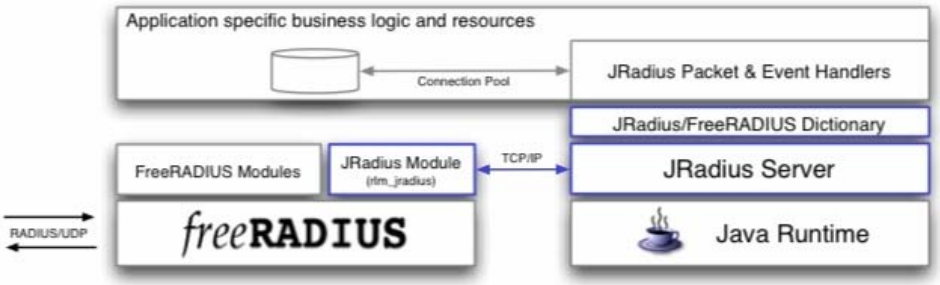


Figure 17: The overall architecture of JRadius AAA server.

source: http://coova.org/wiki/index.php/JRadius/WithFreeRADIUS

The RADIUS standardized protocol describes how to ensure the authentication, authorization, and accounting service. The general RADIUS architecture consists of two elements: RADIUS client and RADIUS server. When the network component (in this case it is a SIS server) receives the access request, then the RADIUS client sends the message `AccessRequest` to the RADIUS server. After that the RADIUS server sends to the RADIUS client the message `AccessResponse`, which contains the result of authentication and the access control rights. The `AccountingRequest` and `AccountingResponse` are the messages, which provide the accounting service. In Figure 18, we present the example UML sequence diagram of an AAA RADIUS server.

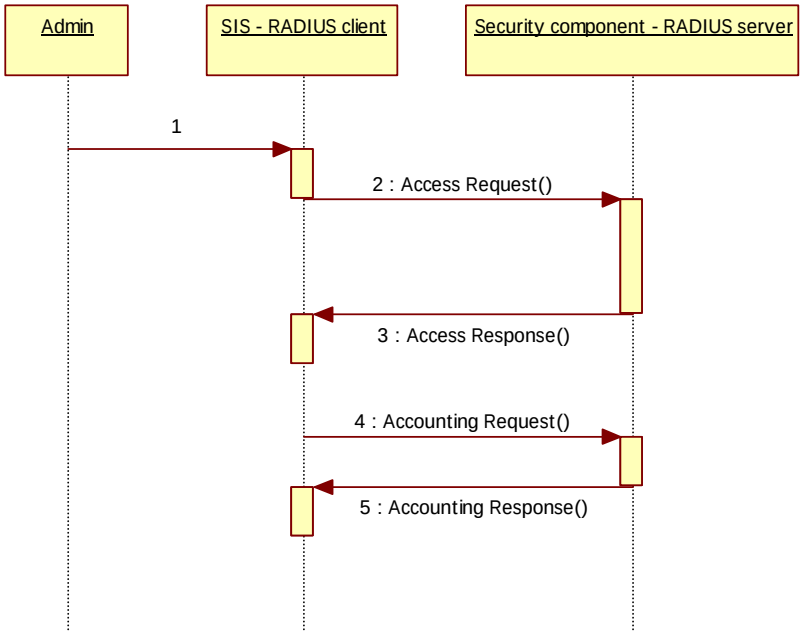


Figure 18: Example sequence diagram of AAA RADIUS server

It is worth mentioning that the Security component is independent of ETM approaches (communication through the admin interface and the inter-SIS interface does not depend on the implemented ETM mechanism).

5.6 QoS Manager and NGN Integration

The QoS Manager aims to check the availability of network resources and to guarantee resources requested by the end user as well as to enforce the QoS policies in the network.

This is a complex task, since QoS mechanisms are technology dependent and usually require ad-hoc solutions. The mechanisms proposed next consider the most general solution where NGN transport control capabilities are available in the ISP network.

For other networks, where these capabilities are not available, the QoS Manager will have to interface the different network equipment (i.e. BRAS in a xDSL access) in order to enforce the different policies.

The QoS Manager is in charge of enforcing specific policies in the network according to:

- specific operator rules (relationship with the Admin interface),
- end user requests (that can, e.g., request some QoS guarantees for specific connections).

Therefore, this module must interface with the network equipment. In order to provide a common solution, an initial approach is to interface the transport control functionalities of the NGN equipment [Y.2111]; in particular, the QoS Manager will use the Gq interface provided by the RACS [RACS]. This would allow the SmoothIT architecture to provide a standard compliant solution easier to deploy.

5.6.1 Requirements

The QoS Manager and its interaction with the NGN Transport Control Functionalities support the following functional and non-functional requirements.

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1	Improve the performance of the overlay application by guaranteeing network performance parameters for a selected set of connections. This could also support the reduction of inter-domain traffic if the guarantees are applied to intra-domain connection.	X	
F.2	Incentive-compatibility: The E2E-QoS mechanism aims to provide incentives to all the players: - improve the performance for the end users' connections - guarantee the performance for the overlay service providers' main connections (i.e. connections from the server).	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Easy deployment: In the current specification (based on the usage of NGN transport control capabilities), the mechanism will reuse the facilities provided by the networks	X	
N.2	Scalability: The SIS shall be scalable to support a large end-user population. For the scenario where end users can request specific QoS guarantees, maybe some scalability problems can be foreseen (this needs some performance tests). On the other hand for the scenario based on an SLA with a service provider, no scalability problem is foreseen	X	
N.3	Standard compliance: The SIS shall use and based on standard protocols where applicable. This has been considered in the specification of the interface to NGN components.	X	

5.6.2 Offered Interfaces (Commercial)

This section remains confidential for the time being.

5.6.3 Utilized External Interfaces (Commercial)

This section remains confidential for the time being.

5.6.4 Component Design (Commercial)

This section remains confidential for the time being.

5.6.4.1 Parameters (Commercial)

This section remains confidential for the time being.

5.6.5 Customization for trials (Commercial)

This section remains confidential for the time being.

5.6.6 Testing

5.6.6.1 Scenario: QoS Manager Tests

Test case	*Pre-condition, Input*	*Expected result*
1. interface to the NGN test	No reservation is done in the RACS.	The RACS sends the affirmative response and the resources are

	The QoS Manager interface is tested: a script to test the interface ./qos_install.sh is run	properly configured. The RACS web interface provided by the vendor has also the information about the reservations.
2. SLA installation	The admin uses the GUI interface provided by the QoS Manager to test the installation of a new policy.	The admin sees that the policy has been enforced in the network as displayed in the GUI. In the Web interface provided by the RACS, the policies are enforced as expected.

5.7 Summary

This section describes the components being developed for the SIS framework in order to support ETM mechanisms proposed in SmoothIT. Its main focus is the requirements and design of single components. It further includes the APIs offered between components and implementation details. The focus is on components residing in the ISP-side of the framework; meaning the components, SIS Controller, SIS DB, Metering, Security, and QoS Manager, from Figure 1. The client-side integration of SIS is discussed in the next section.

6 Extension of a P2P Client with SIS Functionality

This section describes the integration of SIS functionality in a peer-to-peer client. It, first describes the required modifications in a general way, assuming an abstract client used for content distribution. The next subsection presents an overview of the client application used within SmoothIT (as selected in WP1), focusing on its relevant technical aspects.

The main part of the section describes the extension of the client based on the work done within WP2 (Task 2.2) that deals with the ETM mechanism specification. The modifications were performed in the NextShare client in order to support SIS-based ETM mechanisms and to demonstrate their effectiveness in the upcoming trials. Our goal is to keep modifications, of the client, to as little as possible since the ETM approaches developed are not intended to be tailored to a specific overlay application.

6.1 Generic SIS Client

The general architecture enhances an abstract P2P client with SIS related functionalities. We assume that such a client comprises at least the following functions:

- **Service discovery**: mechanism to search for a specific service requested by the user and to identify the remote peers carrying that service. For example in BitTorrent [BT], searching for a movie on a tracker listing site, retrieving the associated torrent file, contacting a tracker, and getting the list of remote peers that have the movie.
- **Service selection**: mechanism to select the remote peers from which to receive the discovered service or whom to grant the requested service. For example, in a file sharing application, deciding out of a list of retrieved remote peers from which specific peers to download the movie.
- **Service management**: mechanism to maintain the performance level of the received service. For example in BitTorrent, this applies to peer choking and optimistic unchocking.
- **Monitoring**: component to assess the application's performance and to provide information necessary to rank peers and resources according to used metrics. For example in BitTorrent, peers are ranked according to their upload or download speed.

The SIS Client itself is an additional module that is used by the application to communicate with other SIS entities (via the SIS Client interface). Its exact functionality depends on the ETM mechanisms applied and will be elaborated on. At the very least, it communicates with the SIS to provide requirements (e.g. remote peers about to be selected) and receive policies (e.g. remote peers preferred from the ISP and client point of view).

Figure 24 shows the new component, called SIS Client, to be integrated in the client application. It can offer the following functions: peer ranking functionality, security support, and QoS support. Each of them can be used by the aforementioned client functionalities (also depicted in the figure) to modify a client's behavior.

The first version of the SIS prototype modifies the behavior of the monitoring mechanism (by obtaining additional monitoring information from SIS) and the service selection mechanism (by modifying peer selection mechanism based on additional ranking from

SIS). The security feature allows protecting SIS-related communication from malicious parties. Finally, the QoS component allows modifying client's behavior according to NGN facilities (see Section 5.1, NGN integration).

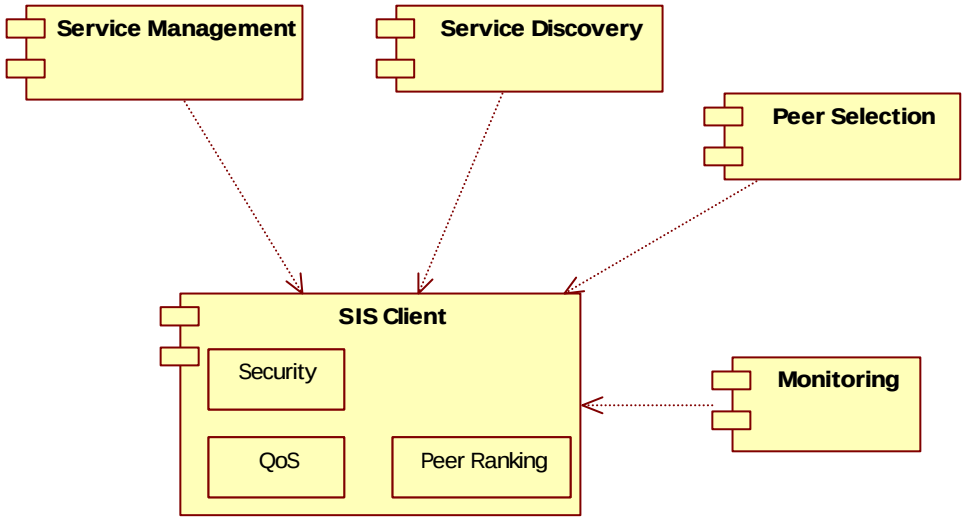


Figure 24: General client architecture

6.2 Overview of Tribler and NextShare

Tribler is an extension of the ABC BitTorrent client (ABC = yet Another BitTorrent Client) that in turn was an extension of BitTornado client. Tribler is written in Python and was developed at the University of Vrije and Delft University of Technology. Currently, the client is actively being developed within the 7th FP project P2P-NEXT [p2p-next]. The version developed within P2P-Next is called NextShare (its user interface is the so-called *SwarmPlayer* [swarmplayer*]),* its most obvious difference to Tribler is a new user interface that concentrates on streaming capabilities and hides non streaming related functionalities, such as a recommendation service and collaborative (social) downloads. The open-source releases of the NextShare client are still branded with the name Tribler. Therefore, through the remainder of this section will use the term Tribler and NextShare interchangeably, since they share the same core functionalities relevant for SmoothIT.

Besides the common BitTorrent functionality (based on Tit-for-Tat [t4t]) NextShare offers streaming capabilities (live and video on demand) based on the Give-to-Get mechanism [g2g], which is better suitable for streaming than Tit-for-Tat. Note that Give-to-Get is only applied between two Tribler/NextShare instances, while Tit-for-Tat is applied if the remote peer is not using NextShare/Tribler.

In fact, the client can act as both a BitTorrent and Streaming client simultaneously and even combine both mechanisms while downloading the same file.

6.3 Integration of SIS in NextShare

This subsection describes the extensions within NextShare that have been performed up until now in order to support SIS. The initial focus was on the BGP-Loc ETM mechanism

and test-bed support. In future ***releases*** we will further focus on the trial support and integration of more ETM mechanisms.

After initial experiments with the NextShare client, the SIS integration was performed based on the NextShare release made available by the P2P-Next team. Modifications were done to the NextShare core and an additional functionality was implemented to communicate with SIS components residing at the server. The whole communication happens via the ***sis.1*** interface provided by the SIS Controller.

6.3.1 Requirements

Since NextShare is a live project under active development the extensions have to be as modular as possible to allow integration with subsequent versions. Detailed requirements are:

Functional Requirements

ID	Requirement	Core req.	Add. req.
F.1.	Communicate with SIS server to receive SIS peer ranking.	X	
F.2.	Use SIS-provided ranking information for peer selection.	X	

Non-functional Requirements

ID	Requirement	Core req.	Add. req.
N.1	Extensibility: It shall be easy to integrate the implementation with the overlay application.	X	
N.2	Efficiency: Communication with the SIS server should not introduce additional delay and throttle the download performance.	X	
N.3	Security: Communication with SIS server can be authenticated and verified using the Security component. This feature is optional and only relevant for the external trial.		X

6.3.2 Relevant Mechanisms (Commercial)

This section remains confidential for the time being.

6.3.3 Extensions/Modifications to NextShare (Commercial)

This section remains confidential for the time being.

6.3.4 Offered Interfaces

The SIS Client does not offer any programmable interfaces to other components. It uses the sis.1 of the SIS Controller to communicate with the SIS.

6.3.5 Structural View (Commercial)

This section remains confidential for the time being.

6.3.6 Behavioral View (Commercial)

This section remains confidential for the time being.

6.3.7 Bipartite Sorting Algorithm

The first SIS-compatible modification to NextShare algorithms is done similar to the proposal of the simulation taskforce. It divides the potential unchoke candidates in two groups (hence, the algorithm name): “good” and “bad”. The distinction is perfomed based on the SIS server reply, which ranks peer IPs either with 0 (bad peer) or a positive integer number (good peers). Later on, we can treat peers with different positive rankings differently (e.g. reuse the order while assigning unchoke slots). The overall algorithm is presented below.

```
Input: list of potential unchoke connections
(including IP addresses)

1. Send request to SIS server, containing the IP
  addresses of candidates
2. init empty unchoke_list
3. For each IP in the server reply:
4.   If ranking(IP) > 0 add the connection to the
unchoke_list
5. Unchoke the resulting filtered unchoke_list as
usual
```

Algorithm 3: Bipartite sorting algorithm

In the next release, modifications to the algorithm will be performed in order to:

- Distinguish peers with different positive rankings (combine the ranking with T4T/G2G)
- Allow the server to insert additional IP addresses into the list

6.3.8 Tests

Invocation: The client is executed through the “*linuxRun”* script (it also works on windows using cygwin) with the supported parameters “*nofilter*”, “*sis,*” “*simple*”, or “*local*”. Then the client is started with the given parameterization.

All scenarios were tested with this torrent:

`Big_Buck_Bunny_1080p_surround_frostclick.com_frostwire.com.torrent`

This torrent points to the open source video “Big Bug Bunny” [BBB] published under Creative Commons Attribution 3.0 license.

6.3.8.1 Scenario 1: No SIS running

Test case	*Initial condition*	*Input*	*Expected Result*
1. No filtering (vanilla NextShare)	SIS is not running	Run swarmplayer in a filter-less mode with: `linuxRun nofilter`	NextShare download runs in an unbiased mode. In file `output.log` one can read lines of content `<timestamp> NOT USING ANY FILTERING SERVICE` `<timestamp> UNCHOKING <IP addresses>`
2. SIS-based peer ranking	SIS is not running	Run swarmplayer in SIS mode with default SIS URL: `linuxRun sis`	Player does not startup. Last two lines of the output: `Failed to generate wsdl bindings.` `Are you sure SIS server is running at http://127.0.0.1:8080/sis/ClientEndpoint?wsdl ?`
3. Non-default SIS URL	SIS is not running	Run swarmplayer in SIS mode with custom SIS URL <YOUR-URL>: `linuxRun sis <URL>`	Expected behavior: player does not startup. Last two lines of the output: `Failed to generate wsdl bindings.` `Are you sure SIS server is running at <YOUR-URL> ?`
4. Local ranking of peers (emulate SIS)	SIS is not running	Run swarmplayer in local-filtering mode with: `linuxRun simple`	Expected behavior: same as above

6.3.8.2 Scenario 2: SIS running

Test case	*Initial condition*	*Input*	*Expected result*
	SIS is	Run swarmplayer in a	Normal NextShare download. In

1. No filtering (vanilla NextShare)	running at http://127.0.0.1:8080	filter-less mode with:
		`linuxRun nofilter`
			file `output.log` you can read lines of content
			`<timestamp> NOT USING ANY FILTERING SERVICE`
			`<timestamp> UNCHOKING ['212.46.229.84']`

2. SIS-based peer ranking	SIS is running at http://127.0.0.1:8080	SIS Client Endpoint is running at http://127.0.0.1:8080/sis/ClientEndpoint?wsdl
		Run swarmplayer in SIS mode with default SIS URL:
		`linuxRun simple`
			NextShare download with biased unchoking. In file `output.log` you can read something like:
			`<timestamp> USE SIS-FILTERING`
			`<timestamp> REQUESTED ['118.90.128.11', '212.46.229.84']`
			`<timestamp> SIS-RESPONSE {'118.90.128.11' : 0,'212.46.229.84' : 100}`
			`<timestamp> UNCHOKING ['212.46.229.84']`

3. Non-default SIS URL	SIS is running at http://127.0.0.1:8080	Run swarmplayer in SIS mode with custom SIS URL <YOUR-URL> `linuxRun sis <YOUR-URL>`	Same as above
4. Local ranking of peers (emulate SIS)	SIS is running at http://127.0.0.1:8080	SIS Client Endpoint is running at *http://127.0.0.1:8080/sis/ClientEndpoint?wsdl* Run swarmplayer in SIS mode with default SIS URL: `linuxRun sis`	NextShare download with biased unchoking. The result will depend on the implemented `getRankedPeerList` function of SIS.

7 Summary and Conclusions

This deliverable presents the initial implementation of the SmoothIT architecture. It re-engineers the original architecture presented in SmoothIT deliverable D3.1, in order to support the detailed interfaces offered and used by single components.

The realization of the ETM framework covers the technology used to implement single components, interface specifications, and protocols used to allow inter-component communication. The framework implementation uses the JBoss application server to host the core functionality of SmoothIT information services and communicates to external entities via standardized protocols, such as SOAP. For each component, the requirements, offered interfaces, design (both structural and behavioral), and most relevant algorithms are presented.

Inside of the framework, three main components have been implemented: SIS Controller, Metering, and SIS database (SIS DB). They are required to support the initial ETM mechanism, called BGP-based locality and already specified by SmoothIT consortium. Further server-side components (Admin interface, Security, and QoS Manager) are designed to match the framework implementation and will be integrated in the next software release.

On the client side, the cooperation with the EU project P2P-Next resulted in the initial integration of SIS functionality in the NextShare client, done by SmoothIT consortium. The client's ability to communicate with the SIS server as well as to receive ETM information and to update its behavior accordingly, was extended.

The current implementation of the framework supports the basic ETM mechanism and the ongoing specification of further ETM mechanisms will result in the extension of the involved components. This applies mostly to the SIS Controller, but for specific ETM mechanisms also to the client, QoS Manager, and to the attainment of Inter-SIS communication.

The deployment aspects of the SIS architecture are presented to show the system setup for trials. Here the current focus lies on the internal trial, but is kept extensible to cover the external trial in future releases.

Additionally, the document appendix summarizes the results of the technology scouting that served as basis for relevant design decisions, regarding the utilized technology and protocols. Furthermore, the SmoothIT software integration and release process are part of the appendix.

This deliverable serves as the basis for future extensions of the ETM framework and to WP4 and will integrate the framework with test-bed environment.

References

[ant] Apache Ant, http://ant.apache.org/

[apsw] APSW wrapper for Sqlite DB, http://code.google.com/p/apsw/

[coberture] Cobertura, http://cobertura.sourceforge.net/

[D2.2] SmoothIT Project : *ETM Model and Components (Initial Version),* Deliverable D2.2, December 23, 2008

[D3.1] SmoothIT Project : *Economic Traffic Management Systems Architecture Design (Initial Version)*, Deliverable D3.1, October 12, 2008

[dao] Data Access Object (DAO) http://java.sun.com/blueprints/corej2eepatterns /Patterns/DataAccessObject.html

[dto] Data Transfer Object (DTO) http://java.sun.com/blueprints /corej2eepatterns/Patterns/TransferObject.html

[eclipse] Eclipse IDE, http://www.eclipse.org

[euqos] EuQoS FP6 EU Project, http://www.euqos.eu

[findbugs] FindBugs, http://findbugs.sourceforge.net/

[ITU-X.805] ITU-T Recommendation X.805: *Security architecture for systems providing end-to-end communications*, 2003

[jaxws] Sun JAX-WS, https://jax-ws.dev.java.net/

[jboss] JBoss Application Server, http://www.jboss.org/jbossas/

[jbossws] JBossWS web service stack, http://www.jboss.org/jbossws/

[jdk] Java Development Kit (JDK), http://java.sun.com/

[junit] JUnit, http://www.junit.org/

[log4j] log4j, http://logging.apache.org/log4j/1.2/index.html

[m2crypto] M2Crypto library, http://wiki.osafoundation.org/bin/view/Projects /MeTooCrypto and http://chandlerproject.org/Projects/MeTooCrypto

[modelnet] ModelNet Network Emulator, http://modelnet.ucsd.edu/

[oracle] V. Aggarwal, O. Akonjang, A. Feldmann. *Improving User and ISP Experience through ISP-aided P2P Locality*. In Proceedings of 11th IEEE Global Internet Symposium 2008

[p2p-next] P2P Next EU/ICT project, http://www.p2p-next.org/

[P4P] P4P - Proactive network Provider Participation for P2P, http://www.openp4p.net/

[py2exe] py2exe, http://sourceforge.net/project/showfiles.php?group_id=15583

[pydev] PyDev for Eclipse IDE, http://pydev.sourceforge.net/

[pysqlite] PySqlite wrapper for Sqlite DB, http://oss.itsystementwicklung.de/trac/pysqlite

[python] Python, http://www.python.org/

[pyxml] PyXML, http://sourceforge.net/projects/pyxml/

[quagga] Quagga Routing Suite, http://www.quagga.net/

[RACS] ETSI ES 282 003. Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); *Resource and Admission Control Sub-System (RACS)*: Functional Architecture

[RFC4273] J. Haas, S. Hares, Ed., Definitions of Managed Objects for BGP-4. IETF RFC 4273, January 2006.

[snmp4j] SNMP4J, http://www.snmp4j.org/

[staruml] StarUML, http://staruml.sourceforge.net/en/

[svn] SVN, http://subversion.tigris.org/

[swarmplayer] P2P-Next Consortium, *SwarmPlayer Streaming Client*, http://trial.p2p-next.org/

[wxpython] wxPython wrapper for the wxWidgets library, http://www.wxpython.org/

[xdocklet] Xdoclet, http://xdoclet.sourceforge.net/

[Y.1541] ITU-T Y.1541, Network Performance objectives for IP-Based services.

[Y.2111] ITU-T Y.2111, Resource and Admission Control functions in Next Generation Networks.

[zsi] The Zolera Soap Infrastructure, http://pywebsvcs.sourceforge.net/

Abbreviations

AAA	Authentication, Authorization, Accounting
AD	Active Directory
API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
BGP	Border Gateway Protocol
BRAS	Broadband Remote Access Server
BRAS	Broadband Remote access Aggregation Service
BT	BitTorrent
CPU	Central Processing Unit
DB	Database
DNS	Domain Name System
DoS	Denial-of-Service
DPI	Deep Packet Inspection
DSL	Digital Subscriber Loop
DSLAM	Digital Subscriber Line Access Multiplexer
ETM	Economic Traffic Management
FTTH	Fiber to the Home
G2G	Give-to-Get
GNU	GNU Not Unix
HTTP	Hyper-text Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
ID	Identifier
IETF	Internet Engineering Task Force
ISP	Internet Service Provider
ITU	International Telecommunication Union
JPA	Java Persistence API
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
NASS	Network Attachment Sub-system
NGN	Next Generation Networks
P2P	Peer-to-peer
QoS	Quality of Service

RACS	Resource and Admission Control Sub-system
RADIUS	Remote Authentication Dial In User Service
RPC	Remote Procedure Call
SIS	SmoothIT Information Service
SLA	Service Level Agreement
SmoothIT	Simple Economic Management Approaches of Overlay Traffic in Heterogeneous Internet Topologies
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
STREP	Specific Targeted Research Project
SVN	Subversion Version Control System
T4T	Tit-for-Tat
UML	Unified Modeling Language
VDSL	Very high bit-rate Digital Subscriber Line
xDSL	x-type Digital Subscriber Line
XML	Extensible Markup Language
YAML	YAML ain't Markup Language

Acknowledgements

This deliverable was made possible due to the large and open help of the WP3 team of the SmoothIT project within this STREP, which includes besides the deliverable authors as indicated in the document control, a number of additional persons as well. The authors would like to thank especially the SmoothIT-internal reviewers Maria Angeles Callejo Rodriguez and Frank Lehrieder for their valuable comments.

Appendix A. Technology Scouting

The goal of SmoothIT is to provide an interconnection between overlay and underlay networks and do this through an evolutionary, rather than revolutionary approach. The reason for technology scouting was to reuse existing technologies and to follow established standards whenever they are applicable.

The functionality of network equipment is diverse, especially between different manufacturers. Furthermore, there exist freely available platforms that are typically employed in test-beds and are also heavily used in production environments. Therefore, within SmoothIT it is crucial to understand which network equipment functionality can be presumed to be available, even across several ISPs.

This appendix summarizes the available functions of different network equipment platforms and technologies and concludes which ones are advisable for application in SmoothIT project.

The studies were carried out in four main areas:

1. Control Data Protocols
2. Measurements
3. Security
4. Traffic Management Mechanisms

A.1 Control Data Protocols

The protocol will be used for communication between clients and the SIS server as well as between different SIS servers. To be overlay and client agnostic and allow for easy integration of SmoothIT technology into other clients, implementations of this protocol should be available for multiple platforms and programming languages.

Other requirements imposed on the protocol are performance and maturity. Furthermore, the ability of the protocol for self-validation is highly desirable. Lastly, the protocol should be human readable without any additional tools to help in debugging.

Based on the above criteria four protocols were selected:

1. YAML and its subset JSON
2. SOAP
3. REST
4. Google Protocol Buffers

All of the above protocols were found to be mature enough with their implementations existing on virtually any combination of platform/language. The main differences were in performance and readability criteria.

A.1.1 YAML

YAML [1], [2] stands for *Yet Ain't Markup Language* as opposed to XML. It is a serialization format borrowing ideas from XML, C, Python and Perl. It was first introduced by Clark Evans in 2001.

YAML operates with data structures found in most modern languages such as hashes, arrays, and scalars, presenting them in a human readable format. Implementations of YAML exist for a large range of programming languages. JSON is a widely known subset of YAML using its inline notation.

Table 3: SWOT - YAML

Strengths	**Weaknesses**
• Human readable • Multiple implementations exist • Fast parsing	• No validation mechanism exists
Opportunities	**Threats**
• High performance implementations can be easily created. • Implementation, especially in scripting languages is trivial.	• Implementation might be somewhat problematic as custom validation will need to be written.

A.1.2 SOAP

SOAP is a very widely used XML-based RPC and serialization protocol. Using XML as the data format implies that it is human readable and can be easily validated. Though implementations exist for most systems, XML parsers might lack performance under heavy load.

Table 4: SWOT - SOAP

Strengths	**Weaknesses**
• RPC standard de-facto • Multiple implementations exist • Human readable • Allows parameter validation	• XML parsers are generally rather slow
Opportunities	**Threats**
• Solution can be implemented easily thanks to large number of libraries and tools available. • Maintaining is not an issue as the standard is well tested over time.	• Potential problems exist when scaling implementation to a large number of simultaneous requests.

A.1.3 REST

REST is an RPC protocol and the readability and automatic validation parameters depend on serialization format of choice. It can easily be used with any of the described formats.

A.1.4 Google Protocol Buffers

Google Protocol Buffers is a new but very fast protocol with excellent validation capabilities and great performance metrics. Unfortunately, it is a binary protocol and debugging it might not be an easy task. As being human readable was identified as one of the core requirements for the protocol, Google Protocol Buffers were excluded from the evaluation.

A.1.5 Conclusion

After detailed evaluation, where each protocol was graded in a number of areas relevant to SIS implementation, SOAP scored highest and was selected as the protocol of the choice for SIS communication. Some additional factors that contributed to this decision are the facts that SOAP is the industry standard and is well supported by the J2EE framework, which is used throughout the server side implementation.

A.2 Measurements

A number of techniques and protocols are available to collect measurements from the network including, but not limited to NetFlow, RADIUS, BGP etc.

NetFlow is a proprietary protocol developed by Cisco, but its specification is completely open. It allows equipment to collect traffic statistics at IP level and its implementations exist for IOS, FreeBSD, and OpenBSD. It is also important to mention that it is an emerging IETF standard known as IPFIX - Internet Protocol Flow Information eXport.

RADIUS and its successor Diameter are two very well known AAA protocols used in a variety of cases in different networks. Even though Diamater is more advanced, as far as measurement requirements of SmoothIT are concerned, they can be considered as one protocol.

RADIUS records contain basic information about users sessions, which includes user and sessions ids, network address, point of attachment recorded in the very beginning, periodical updates on session duration and data usage, and the final update on session duration and termination reason.

Finally, BGP (Boarder Gateway Protocol) routing information was found extremely useful for the purposes of SmoothIT project as it can be directly used to evaluate the peer's locality.

Choice of measurement technology is not subject to SWOT analysis, but depends on the ETM techniques implemented.

A.3 Security

While SIS service is meant to be used by everyone it is important to ensure that appropriate security technology is in place to avoid abuse. This is especially important for inter-SIS communication, where large amounts of network critical information would be exchanged.

Security enforcement can be broken down into authorization and authentication of users on one side and encryption of communication channels on the other side.

A.3.1 SSL

For encryption of communication channels, the only feasible solution found was Secure Socket Layer (SSL). It is very widely used and supported by virtually all platforms. By encrypting communication on transport layer it stays protocol agnostic. Furthermore, SSL allows authentication of both server and client. The only real downside to be considered is that SSL, just as any other encryption technique, is rather CPU intensive and is therefore not to be overused.

Table 5: SWOT - SSL

Strengths	**Weaknesses**
• Industry standard • Supported on all platforms • Both client and server can be authenticated	• Increases request processing time
Opportunities	**Threats**
• Communication with third party SIS can be easily established and both parties can be authenticated with certificates. • Customers can be provided with premium service without jeopardizing security.	• If overused can significantly slow down the SIS server when high load is reached.

A.3.2 RADIUS/Diameter

RADIUS (Remote Authentication Dial-In User Server) is a highly popular AAA server, which became the de-facto standard in many networks and is a good candidate for SIS cases. Authentication can be done both using username/password pairs, in which case they would be encrypted before transmission, and certificates. The storage engine for RADIUS can be selected from a large number of choices such as SQL, Kerberos, LDAP and AD.

Diameter protocol is an improved version of RADIUS, which comes at the cost of lacking full backward compatibility. As opposed to RADIUS, which works via UDP, Diameter uses TCP and supports application acknowledgements, error notification and better accounting.

As both protocols share same basic architecture, the following table presents a joint SWOT analysis.

Table 6: SWOT – RADIUS-based AA

Strengths	Weaknesses
• Is standardized protocol (RFC 2865 and RFC 2866) • Highly scalable	• additional messages (Access-Request, Access-Response etc.) - additional operation cost
Opportunities	**Threats**
• can be used for future services (more universal)	• Potential increase in system complexity as RADIUS requires for client and server in SmoothIT system.

A.3.3 JBoss-based AA

JBoss server provides mechanisms for authentication and authorization that support username/password authorization and are easy to integrate into J2EE base solutions. From the point of view of SmoothIT project it is a viable alternative to RADIUS, which can make implementation much easier and cleaner.

Table 7: SWOT – JBoss-based AA

Strengths	Weaknesses
• Authentication and access control are implemented in JBoss • Authentication and access control have better performance (they are integrated with JBoss environment)	• By default stores password as plaintext
Opportunities	**Threats**
• Easy integration with SIS architecture .	• Depends on JBoss environment (problem with moving SIS to other platform) • Accounting services will need to be implemented separately

A.3.4 Conclusion

The AAA server based on JBoss services will be the best solution when the performance of the system is a crucial requirement and we are sure that JBoss will be deployed. The RADIUS server services should be used in order to implement a scalable system, which can be deployed in different environments and, additionally, with the higher security level. As far as encrypting communication channels is concerned, SSL is the only feasible choice.

A.4 Traffic Management Mechanisms

Traffic management is an integral part of a number of possible SmoothIT ETM approaches; therefore, various available possibilities were studied. Technologies in this area can be broken down to proprietary ones, which include Cisco, and open source, mainly Linux-based TC.

A.4.1 CISCO

Cisco-based solutions use class based queuing and traffic shaping with QoS mechanisms. They include limited capabilities of Network Based Application Recognition (NBAR), which cannot be extended. Furthermore, performance of the solutions was found to be rather low and inadvisable to be installed on core routers.

Table 8: SWOT – Cisco-based traffic management

Strengths	**Weaknesses**
• Well established and supported industry standard solution • Supports DPI • Documentation is good.	• Proprietary and not extensible • limited performance
Opportunities	**Threats**
• The test-bed can be setup with PCs (easier to deploy in a reduced environment) • TID has expertise in managing it • If we add IMQ patch to the kernel we can configure QoS policies.	• Support of new P2P protocols might be problematic unless CISCO releases an appropriate NBAR module • Scalability problems can arise

A.4.2 Linux-based traffic shaping

Linux-based Traffic Shaping was found to have medium performance, being able to process 500mbps traffic on an average DualCore x86 box. Layer 7 protocol matching is available but somewhat limited. However, being open source, this solution is extremely flexible and can be extended and customized. Among other things this allows for extensions to be written for replication of metrics between multiple nodes to form a distributed traffic management overlay. This technique is not supported by any of the above-mentioned commercial solutions.

Table 9: SWOT - Linux-based traffic shaping

Strengths	Weaknesses
• Open Source • Runs on wide range of HW • Supports DPI and can be extended	• Medium level performance on highly loaded links
Opportunities	**Threats**
• Can be extended to support distributed traffic shaping mechanisms • PrimeTel team is well familiar with the solution	• None identified

A.4.3 Zebra

GNU Zebra is free software that manages TCP/IP-based routing protocols. It is released as part of the GNU Project, and it is distributed under the GNU General Public License. It supports BGP-4 protocol as described in RFC1771 (A Border Gateway Protocol 4) as well as RIPv1, RIPv2, and OSPFv2. Unlike traditional, monolithic architectures and even the so-called “new modular architectures” that remove the burden of processing routing functions from the CPU and utilize special ASIC chips instead, Zebra software offers true modularity which simplifies the extension of the protocols for prototyping. Zebra is unique in its design in that it has a process for each protocol.

One of its multiple features is its modularity. Due to the multi-process nature of the Zebra software, it is easily upgraded and maintained, so it is feasible to add further protocols or mechanisms. Each protocol can be upgraded separately, leaving the other protocols and the router online.

Zebra also features a high reliability. In the event of failure of any of the software modules, the router can remain online and the other protocol daemons will continue to operate. The failure can then be diagnosed and corrected without taking the router offline.

With the use of the libraries Linux IMQ it is possible to configure QoS in the networks and implement different mechanisms. In the EuQoS [euqos] project, there was an implementation of interdomain links using the software of Zebra to support qBGP and the IMQ queues in Linux routers.

The SWOT analysis of Zebra is shown in Table 10.

Table 10: SWOT - Zebra

Strengths	Weaknesses
• Open Source • It can run in several platforms (GNU/Linux 2.2.X and 2.4.X, FreeBSD 4.X, FreeBSD 5.X, NetBSD 1.6.X, OpenBSD 3.X) • Most IP protocols are supported • BGP is supported (important for interconnection tests) to work as router and router reflector • Documentation is good.	• MPLS is not working properly • No possibility for testing LSP
Opportunities	**Threats**
• The test-bed can be setup with PCs (easier to deploy in a reduced environment) • TID has expertise in managing it • If we add IMQ patch to the kernel we can configure QoS policies.	• Not recommendable for the external trial due to the usage of not dedicated HW in routing

A.4.4 XORP

XORP is the industry's only extensible open source routing platform. It is in broad use worldwide, with thousands of downloads by companies and educational institutions and an active international developer community. Designed for extensibility from the start, XORP provides a fully featured platform that implements IPv4 and IPv6 routing protocols as BGP4, RIP, OSPF, and IS-IS as well as a unified platform to configure them. It is the only open source platform to offer integrated multicast capability. The supported routing protocols for multicast are PIM-SM and IGMP/MLD. XORP's modular architecture allows rapid introduction of new protocols, features, and functionalities, including support for custom hardware and software forwarding.

XORP has similar features to Zebra, but experience has shown that it has more capabilities than XORP. While the latter is thought to build commercial systems on the top, Zebra is more focused on providing full and reliable routing solutions.

As it is quoted on the web page, "XORP and Zebra/Quagga compete in the same space. In particular, we both provide open-source implementations of major Internet routing protocols. However, we differ in a number of ways. XORP's emphasis is more clearly on extensibility, and this is reflected in the XORP architecture in many ways."

The SWOT analysis is represented in Table 11:

Table 11: XORP SWOT Analysis

Strengths	Weaknesses
• Open Source • Platforms • Most IP protocols are supported • BGP is supported (important for interconnection tests) to work as router and router reflector • Documentation is good.	• Virtual interfaces are not supported (e.g. VLAN)
Opportunities	**Threats**
• The test-bed can be setup with PCs (easier to deploy in a reduced environment). • Maybe we can add an IMQ patch.	• Not recommendable for the external trial due to the usage of not dedicated HW in routing • Probably we will have to perform more actions to set up a test-bed with XORP-based routers.

A.4.5 Conclusion

Since the currently specified and supported ETM mechanisms do not require traffic shaping or similar mechanisms, no final decision has been made up to now.

Appendix B. NGN Transport Control Functionalities

The main objectives of the NGN are the following:

- to provide better access (high bandwidth, QoS),
- to be able to efficiently carry different services and
- to integrate mobile and fixed architectures and services.

According to the ITU, the NGN is composed of the following strata:

- The **Transport Stratum** includes:
  - The transport functions (in the access, metro and core network) include new optical solutions for the core networks, FTTH, new wireless mechanisms, etc.
  - The transport control functions (resource and admission control functions and network attachment control functions). Currently, there are two standardization bodies in charge of leading the evolution of this control plane: TISPAN and ITU-T.
- The **Service Stratum** which includes the service control functions including service user profile functions; and the application support functions and service support functions. E.g. IMS

According to ETSI/TISPAN, the following picture shows the strata:

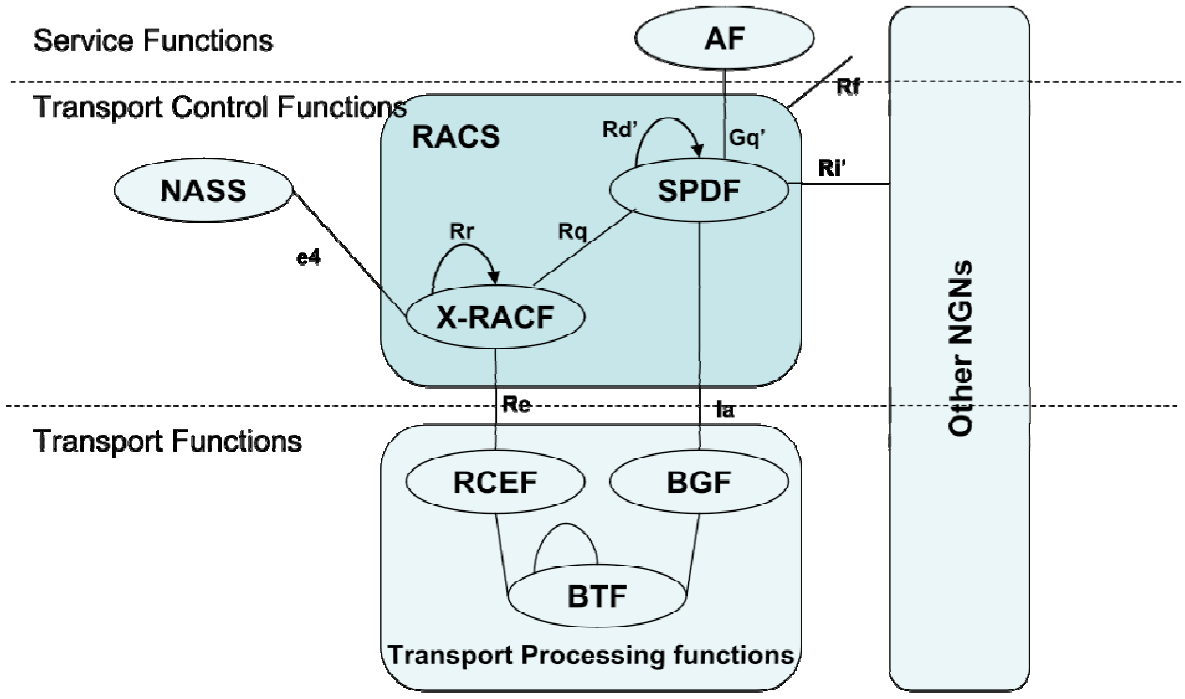


Figure 27: NGN Strata

According to this general description, the network equipment available for the internal trial is described in Table 32.

This figure remains confidential for the time being.

Figure 28: NGN Internal Test-bed

In this test-bed the following capabilities are available:

Transport Functionalities

- DSLAM and BRAS with beta version of the SW to support new QoS configurations and services.
- New multicast capabilities
- Offer SNMP interfaces to monitor different parameters (e.g. traffic in a specific port)

Transport Control Functions

- RACS (that also includes some NASS capabilities, since the users must be provisioned there)
- An implementation of the Gq' interface in SOAP to configure new user policies.

Appendix C. WSDL Interface Definitions

C.1 Interface sis.1 Definition (Commercial)

This section remains confidential for the time being.

C.2 Interface sis.3 Definition (Commercial)

This section remains confidential for the time being.

Appendix D. Integration and Release Procedure

This appendix describes the release and integration procedure applied in WP 3. The procedure is separated into the component release procedure done by component owner, and the system release procedure performed by the system integrator. The procedures are accompanied by the component and system release reports, respectively. Since they are based on the same best practices the both types of procedures and reports have much in common, but still differ in detail.

D.1 Component Release Procedure

Here we describe the necessary infrastructure as well as the procedures for releasing software components intended for integration into the SmoothIT system. These procedures apply to the software component suppliers of the SmoothIT project.

This is an effort to achieve homogeneity in the delivery of software components for integration. The procedures defined herein are intended to assist the smooth integration of software components emanating from different suppliers, under the time constraints set by the project schedule.

D.1.1 Prerequisites

D.1.1.1 Contact Persons

The initial SmoothIT components must be identified and assigned to partners that would function as component suppliers.

Each component supplier shall designate a primary contact person. This person will be responsible for delivering the component release items to the integrator as well as for any other interaction regarding the releases. From here on, unless otherwise stated, all references to “contact person” are to the primary contact person.

The component supplier shall also designate a deputy contact person. This person should be capable of assuming the primary contact person’s responsibility in case of the latter’s absence.

Any change of contact details must be communicated by e-mail to the contact person of the integration team.

D.1.1.2 Tool Set

All component suppliers must use the same toolset for building the components. The toolset description is maintained at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/Tools_and_Libraries.

The integrator contact person is responsible for distribution of the toolset to the component suppliers.

D.1.1.3 Tree

The SmoothIT system will consist of one or more source trees. The Integrator shall create and release the initial tree. Component suppliers will then use the tree to develop their components.

System tree structure is described at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/System_tree_layout.

D.1.1.4 Environment

A uniform build environment has proven crucial in reproducing builds and investigating defects. The libraries description is maintained at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/Tools_and_Libraries.

Compiler options

Build environment uniformity implies uniformity of compiler options as well. The integration team as well as all component suppliers shall use the Ant scripts as distributed from the integration source tree. These scripts include all tool options.

Deviation from these options is strictly unacceptable at all times.

Build process control files (Makefiles)

The following build process control files exist:

/build.xml: The main Ant build file. Builds, deploys and tests all components in the system

/sis/build.xml: The main SIS Ant build file. Builds and tests the SIS

/sis/compile-build.xml: Builds the SIS

/sis/junit-build.xml: Tests the SIS

/sis/packaging-build.xml: Packages the SIS

/sis/xdoclet-build.xml: Runs xdoclet tasks for the SIS

/peer/makedist.bat: Builds the peer for Windows

Versioning

Each component release, including debug versions, shall be identified with a unique version number. Component suppliers are expected to define a versioning scheme, for their individual component, which will remain consistent throughout the project life cycle.

Engineering releases shall be clearly identified. The table below gives an example of a versioning scheme.

Component	Version	Release type
Component X	1.0.12	Official
Component X	1.0.13E	Engineering
Component X	1.0.14	Official

Table 12: Versioning Scheme

Please note that version overlapping between engineering and official releases is not permitted. For instance in the above example we cannot have an official version 1.0.13.

In the RedMine integration project, each uploaded component/release must be assigned to a *version.* These versions will be created and maintained by the project maintainer.

D.1.1.5 RedMine Integration Project

Component releases shall only be made through the official RedMine collaboration site of the project:

- URL: https://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/list_files/smoothit-integration.

In accordance to these categories, the integration team will upload system releases tagged with the description “System Release” and assign it to the according release version.

D.1.1.6 Version Control System

The integration team shall maintain a version control system that contains the source code of the project and tracks the changes made between releases. This system is independent of potential version control systems used internally by each development team. All development teams have read access to this system in order to get the latest version of the source code tree, but only the integration team has write access to this system. Once a development team submits a component to the integrator for release, and this component passes QA (Quality Assurance) tests, then this component is committed by the integration team to this version control system.

The Subversion repository bundled with the RedMine server is used. For this purpose a distinct subproject has been created. The integrators have write access to the repository; the component developers only read access.

The SVN URL for the integration repository is:

https://dev.kom.e-technik.tu-darmstadt.de/svn/smoothit-integration/

In the RedMine integration project, each uploaded component release must be assigned to a *version.* These versions will be created and maintained by the project maintainer.

If the component developer uses the SmoothIT project at RedMine for development (http://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/show/smoothit) the source code should be labeled with release versions.

D.1.2 Issue Tracking

The development and integration teams use an issue tracking system in order to track, assign, review and manage issues regarding the SmoothIT system. This includes, but is not limited to, bugs of the software. The issues can be tracked by their status (for instance, “New”, “Assigned”, “Closed”, “Duplicate”, etc.) as well as their projected completion time. The issue tracking system can categorize each issue based on the corresponding component (module) and can assign the issue to developers. It can also automatically notify the developers via email should their attention be required (for example, if they are assigned a new issue, or if the status of an issue changes).

URL of the issue tracking system:

https://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/smoothit-integration/issues

D.1.3 Test-bed Access

In order to test the components, a common, controlled system build-up is used by all development teams for development, deployment and testing.

D.1.4 Documentation

Documentation for each component will be included in Deliverable D3.3 “Documentation of Engineering and Implementation”. This document shall be updated upon each component release by the component development team.

For Java-based components, complete Javadoc documentation is expected, and documentation files are automatically generated from the source code by the build system.

D.1.5 Release Procedure

As part of the component development process, the component developer team periodically releases a new version of a component to the integrator. The new version of the component may include bug fixes, new features, or both.

D.1.5.1 Trial Build

Before releasing the component, the component supplier team creates a trial build of the whole tree with the new component. If this fails the component should not be released. The integration team will also perform a trial build and if a component release is found to fail the build, it will be returned to the component supplier.

It is expected that the component supplier will check out from the SVN and use the latest official release of the tree for the trial build, with the new version of the component replacing the old one.

D.1.5.2 Tests

Following a successful trial build, a sanity test must be performed to verify the validity of the bug fixes and/or new features that are implemented in the new component.

Partial failure of the sanity test should be clearly stated in the relevant section of the Component Release Report along with the method used for testing.

Full failure of the sanity test constitutes the component unacceptable for release.

Tests may or may not be performed in the test-bed environment. Issues for failed tests should be logged and tracked at the issue tracking system.

Note that Java-based component should provide Junit tests. These tests should cover at least 85% of the component source code or (ideally) more. JUnit failures or code coverage less than 85% constitutes the component unacceptable for release.

D.1.6 Component Deliverables

D.1.6.1 Component Software

The primary element a component release is the component source code. The expected format of the released files differs depending on the component. Please find below the table of the associations.

Component	Supplier	Extension	Description
SmoothIT Client	TUD	.zip	Compressed file containing changed files from base version
InterSIS	TUD	.zip	Compressed file containing changed files from base version
DB	TUD	.zip	Compressed file containing changed files from base version
Admin	DoCoMo	.zip	Compressed file containing changed files from base version
QoS	TID	.zip	Compressed file containing changed files from base version
Security	AGH	.zip	Compressed file containing changed files from base version
Metering	UZH	.zip	Compressed file containing changed files from base version
Controller	ICOM	.zip	Compressed file containing changed files from base version
System Test	ICOM	.zip	Compressed file containing changed files from base version

Table 13: Components and File formats

D.1.6.2 Component Release Report

The component release report is an indispensable part of the component release. It is composed by the component supplier whenever a new version of the component is released for integration.

All component suppliers are expected to use the “SmoothIT project: Component Release Report” document template which is designed specifically for this purpose. The integrator is responsible for distributing the template to the component suppliers. This report should include relevant issue IDs from the issue tracking system for this component release.

D.1.6.3 Executable Code

The executable code produced in the trial build of paragraph 3.1 shall be included in the component release.

D.1.7 Delivery

Only the appointed contact person of the component supplier shall make component releases. Furthermore, the recipient of the component release must always be the contact person of the integration team.

The component release shall consist of the following items:

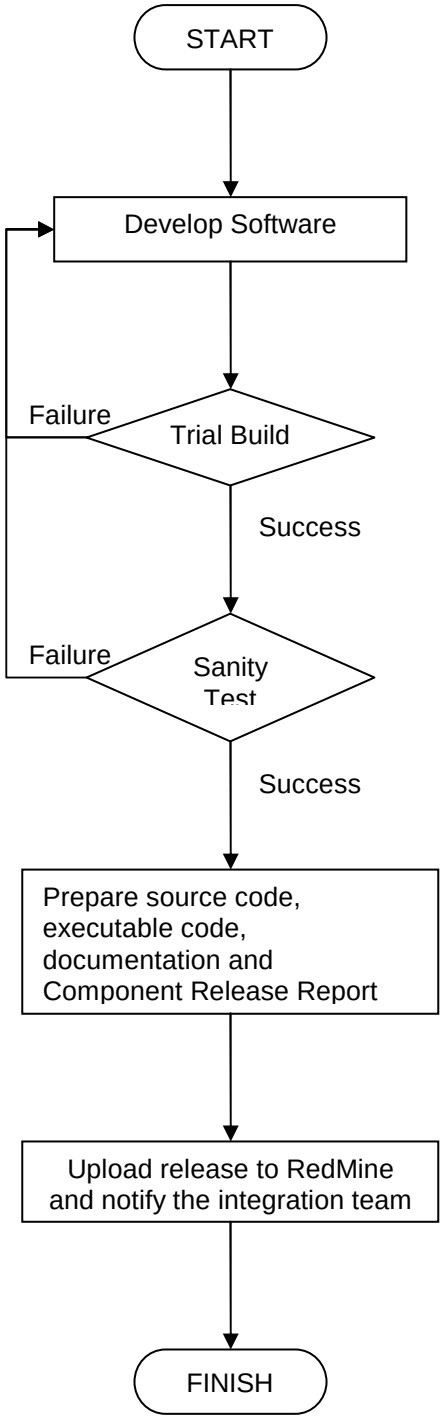
- Component software source code
- Component software executable code

- Component Release Report

Any component release missing (wholly or partially) one or more of these items shall be deemed incomplete and will be returned to the sender unless the component supplier provides sufficient causes.

All deliverables shall be uploaded to the designated location of paragraph 2.5 clearly marked. A single compressed file containing all items is recommended. The contact person of the integration team shall be informed by e-mail about the new release submission. For each release, the documentation for this component should be updated (see paragraph 2.9).

D.1.8 Component Release Flow Chart



D.2 Component Release Report

This is the template for the Component Release Report that accompanies each component release and describes the new features and any problems of that release.

D.2.1 Component Owner

D.2.1.1 Institution and Department Name

[e.g.: Intracom Telecom – Digital Media & Internet Services Dept.]

D.2.1.2 Contact Information

Contact Name

[First Name, Last Name]

Contact E-mail

[someone@partner.com]

Contact Telephone Number

[+30 210 1234567]

D.2.2 Version Description

D.2.2.1 Identification

Component Name & Abbreviation

[e.g. Metering - MET]

Version Number

[e.g. 1.16]

Date Submitted

[e.g. 2008/10/02]

D.2.2.2 Purpose

[Explain why this version was prepared]

D.2.2.3 List of Major Changes

[Include major new features and changes requests. Use the Issue Tracking System reference number (ID) if it exists.]

D.2.2.4 List of Minor Changes

[Include minor new features and changes requests. Use the Issue Tracking System reference number (ID) if it exists.]

D.2.2.5 Known Defects and Problems

[List all known errors in this version that have not yet been fixed. Use the Issue Tracking System reference number (ID) if it exists. Include workarounds and projected release in which they will be fixed.]

D.2.3 Source Code

D.2.3.1 Changed Files

[Include all the changed, added or deleted files of this component in this release]

[e.g.:

/src/foo/bar1.java

/src/foo/bar2.java

/src/foobar/foo.java

/conf/log4j.properties]

D.2.3.2 Location of Software Release

[Upload location of this release and how it is called]

D.2.4 Release Build

D.2.4.1 Unique Build ID

[unique number created using checksum method]

D.2.4.2 Compatibility & Library Dependencies

[Dependency on and required changes to other components or libraries]

[e.g. depends on

- [SmoothIT Component X version Y]
- [Library X, Version 1.0, http://libraryurl.com]
- ..

D.2.4.3 Environment

Build Requirements

[List the environment options required for building this component release]

[e.g.: Set the environment variable SMOOTHIT_HOME=/smoothit/test]

Software Tools

[List the environment tools required for building this component release]

[e.g.:]

- [Sun Microsystems Java Development Kit version 1.6.0_05]
- [Apache Ant version X.XX]

D.2.4.4 Version Build Instructions

[If applicable, explain how to compile the source files into the executable object files. Mention any special configuration options required.]

D.2.4.5 Build Process Output

[Build process screen output from the trial build of the component]

D.2.5 Tests

[List all tests performed by the developer before the component release, including potential JUnit test cases. Mention their outcome and all relevant Issue Tracking reference numbers (IDs)]

D.2.6 Comments

[Insert any other comments here]

D.3 Component Release Procedure: Component Integration and System Release Procedures

Here we describe the necessary infrastructure as well as the procedures for releasing integrated software intended to be used by component suppliers in order to develop software. These procedures apply to the software integration team of the SmoothIT project.

This is an effort to achieve homogeneity in the delivery of integrated software to all component suppliers involved in the SmoothIT project. The procedures defined herein are intended to assist software component suppliers as well as software integrators.

D.3.1 Prerequisites

D.3.1.1 Contact Persons

The integration team shall designate a primary contact person. This person will be responsible for delivering system release items to component suppliers as well as for any other interaction regarding the releases. From here on, unless otherwise stated, all references to “contact person” are to the primary contact person.

The integration team shall also designate a deputy contact person. This person should be capable of assuming the primary contact person’s responsibility in case of the latter’s absence.

Finally, the integration team shall designate an escalation contact person. This person will be contacted only in the cases of unresolved issues or otherwise conflicting interests between the contact person and any outside parties.

Any change of contact details must be communicated by e-mail to the contact person of every component supplier.

D.3.1.2 RedMine Integration Project

Component releases shall only be made through the official RedMine collaboration site of the project:

- URL: https://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/list_files/smoothit-integration

In accordance to these categories, the integration team will upload system releases tagged with the description “System Release” and assign it to the according release version.

D.3.1.3 Tool Set

All component suppliers must use the same toolset for building the components. The toolset is described at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/Tools_and_Libraries.

The integrator contact person is responsible for distribution of the toolset to the component suppliers.

D.3.1.4 Tree

The SmoothIT system will consist of one or more source trees. The Integrator shall create and release the initial tree. Component suppliers will then use the tree to develop their components.

System tree structure is described at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/System_tree_layout

D.3.1.5 Environment

A uniform build environment has proven crucial in reproducing builds and investigating defects. The libraries are described at https://dev.kom.e-technik.tu-darmstadt.de/redmine/wiki/smoothit-integration/Tools_and_Libraries.

Compiler Options

Build environment uniformity implies uniformity of compiler options as well. The integration team as well as all component suppliers shall use the Ant scripts as distributed from the integration source tree. These scripts include all tool options.

Deviation from these options is strictly forbidden at all times.

Build process control files (Makefiles)

The following build process control files exist:

/build.xml: The main Ant build file. Builds, deploys and tests all components in the system

/sis/build.xml: The main SIS Ant build file. Builds and tests the SIS

/sis/compile-build.xml: Builds the SIS

/sis/junit-build.xml: Tests the SIS

/sis/packaging-build.xml: Packages the SIS

/sis/xdoclet-build.xml: Runs xdoclet tasks for the SIS

/peer/makedist.bat: Builds the peer for Windows

Versioning

Each system release, including debug versions, shall be identified with a unique version number. Engineering releases shall be clearly identified. The table below gives an example of a versioning scheme.

System Version	Release type
1.0.12	Official
1.0.13E	Engineering
1.0.14	Official

Table 14: Versioning Scheme

Please note that version overlapping between engineering and official releases is not permitted. For instance in the above example we cannot have an official version 1.0.13.

Version information is displayed by the SIS upon initialization.

D.3.1.6 Version Control System

The integration team shall maintain a “release” version control system tree that contains the source code of the project and tracks the changes made between component and system releases. This system is independent of potential version control systems used internally by each development team. All development teams have read access to this system in order to get the latest version of the source code tree, but only the integration team has write access to this system. Once a development team submits a component to the integrator for release, and this component passes QA (Quality Assurance) tests, then this component is committed by the integration team to this version control system.

The Subversion repository bundled with the RedMine server will be used. For this purpose a distinct subproject has been created. The integrators have write access to the repository; the component developers only read access.

SVN address:

https://dev.kom.e-technik.tu-darmstadt.de/svn/smoothit-integration/

D.3.1.7 Issue Tracking

The development and integration teams use an issue tracking system in order to track, assign, review and manage issues regarding the SmoothIT system. This includes, but is not limited to, bugs of the software. The issues can be tracked by their status (for instance, “New”, “Assigned”, “Closed”, “Duplicate”, etc.) as well as their projected completion time. The issue tracking system can categorize each issue based on the corresponding component (module) and can assign the issue to developers. It can also automatically notify the developers via email should their attention be required (for example, if they are assigned a new issue, or if the status of an issue changes).

URL of the issue tracking system:

https://dev.kom.e-technik.tu-darmstadt.de/redmine/projects/smoothit-integration/issues

D.3.1.8 Test-bed Access

In order to test the system, a common, controlled system build-up is used by all development teams for development, deployment and testing.

D.3.1.9 Documentation

Documentation for each component will be included in Deliverable D3.3 “Documentation of Engineering and Implementation”. This document shall be updated upon each component release by the component development team.

D.3.2 Component Integration Procedure

Upon receiving a new component release from a component supplier, the integration team must integrate this component to the existing tree. As part of the integration process, the

integration team releases a new version of the system to all component suppliers. The new system may include bug fixes, new features, or both.

D.3.2.1 Examine Component Release Report

The integration team examines the Component Release Report and the build output of the component as well as the source and binary code to make sure that all deliverables are acceptable.

D.3.2.2 Trial Build

Before integrating the component, the integration team creates a trial build of the whole tree with the new component. If this fails, the component is rejected and should be returned to the component supplier.

It is expected that the integration team will use the latest official release of the tree for the trial build, in order to integrate a new component.

D.3.2.3 Tests

Following a successful trial build, a sanity test must be performed to verify the validity of the bug fixes and/or new features that are implemented in the new component.

Partial failure of the sanity test should be clearly stated in the relevant section of the System Release Report along with the method used for testing.

Full failure of the sanity test constitutes the component unacceptable for integration. The integration team may perform bug analysis and create an engineering release with the new component. This release is returned to the component supplier in order to provide assistance in resolving the bugs that caused the tests to fail.

Tests may or may not be performed in the test-bed environment. Issues for failed tests should be logged and tracked at the issue tracking system.

Repair Build Environment

If the sanity test described in the previous section fails, the integrator must first check the building environment to ensure that all parameters being used, such as configuration files, toolset etc., are correct.

Repair SVN Settings

The sanity test could also fail due to incorrect usage of the SVN. The integrator should check that it uses the proper SVN tool settings and try again.

D.3.2.4 Checking into the Integration SVN Tree

If the tests are successful, the component is integrated into the integration SVN tree under a new label. The component is now part of the integration SVN tree.

D.3.3 System Release Procedure

Periodically or at specific events (e.g. the integration of a new component release), the integration team releases a new version of the system to all component suppliers. The new system may include bug fixes, new features, or both.

D.3.3.1 Labeling

Before releasing the system, the integration team gives the tree a unique label in order to be able to identify it and retrieve it in the future.

D.3.3.2 Checking out of the Integration SVN Tree

The integrator checks out the entire tree from scratch in order to perform the trial build.

D.3.3.3 Trial Build

Before releasing the system, the integration team creates a trial build of the whole tree with the new component as in the integration procedure.

D.3.3.4 Sanity Test

Following a successful trial build, a sanity test must be performed to verify the validity of the bug fixes and/or new features for this release. This test is performed in the same way as during the integration procedure.

D.3.3.5 MD5 Checksums

After the successful sanity test, the MD5 checksum of each file is computed.

D.3.3.6 Deliverables

Integrated Software

The primary element of the system release is the source code tree in a compressed .zip file.

Executable Code

With each release of the system, the integrator releases the executable code that was used in the trial build in a compressed .zip file.

System Release Report

The system release report is an indispensable part of the system release. It is composed by the integration team whenever a new version of the tree is released to component suppliers.

The integration team is expected to use the “SmoothIT project: System Release Report” document template which is designed specifically for this purpose.

Delivery

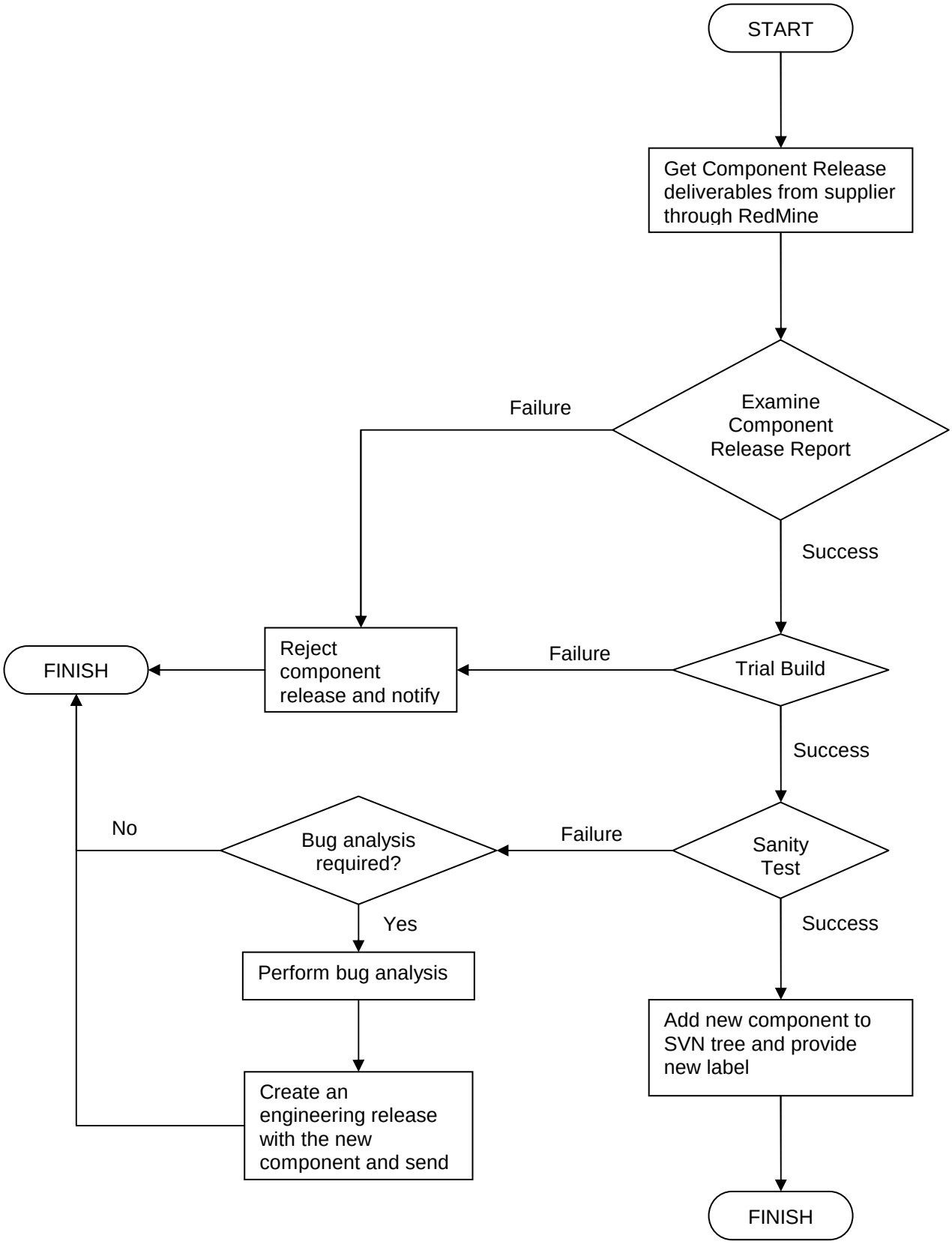
Only the appointed contact person of the integration team shall make system releases. Furthermore, the recipient of the system release must always be the contact person of every component supplier.

The system release shall consist of the following items:

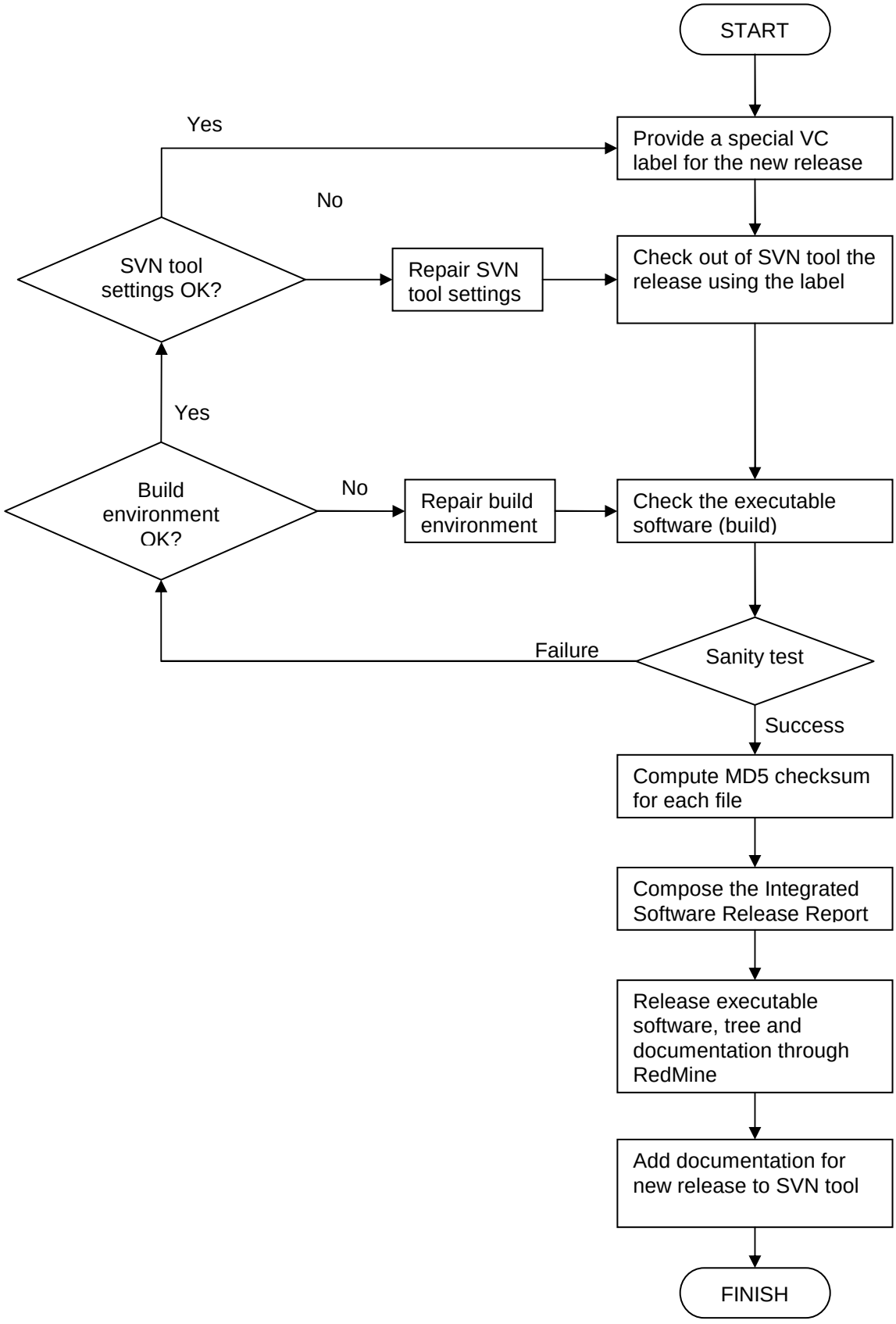
- Source Code Tree
- Executable Code
- System release report

All deliverables shall be uploaded to the designated location of paragraph 2.2 clearly marked. The contact person of every component supplier shall be informed by e-mail about the new release.

D.3.4 Component Integration Flow Chart



D.3.5 System Release Flow Chart



D.4 SmoothIT System Release Report

This is the template for the System Release Report that accompanies each system release and describes the new features and any problems of that release.

D.4.1 Build Identification

D.4.1.1 VCS Labels

[from VCS tool]

D.4.1.2 Date

[Date the release was frozen]

[e.g. 2008/10/02]

D.4.1.3 Contact Information

Contact Name

[Name of the person responsible for the release]

Contact E-mail

[E-mail of the person responsible for the release]

Contact Telephone Number

[Phone of the person responsible for the release]

D.4.2 Issue Tracking Report Document

[report from Issue Tracking system about the problems of this release]

D.4.3 Component List

[the following subsections are replicated for each component]

D.4.3.1 [Component Name; e.g. Metering]

Component Version

[e.g. 1.13]

Component Owner

[Name of the partner & contact person]

[e.g. ICOM – John Doe]

Test Results

[Results from the tests performed for this component]

Known Bugs

[list of open bugs for this component from the issue tracking system]

Notes

[Any other notes for this component]

D.4.4 Implementation Framework

D.4.4.1 Build Requirements

[List the environment options required for building this component release]

[e.g.: Set the environment variable SMOOTHIT_HOME=/smoothit/test]

D.4.4.2 Software Tools

[List the environment tools required for building this component release]

[e.g.:]

- [Sun Microsystems Java Development Kit version 1.6.0_05]
- [Apache Ant version X.XX]

D.4.4.3 Version Build Instructions

[If applicable, explain how to compile the source files into the executable object files. Mention any special configuration options required.]

D.4.4.4 Build Process Output

[the build output of the System]

D.4.4.5 Notes

[Any other notes for this component]

D.4.5 Change Summary

D.4.5.1 Major Changes

[Include major new features and changes requests. Use the Issue Tracking System reference number (ID) if it exists.]

D.4.5.2 Minor Changes

[Include minor new features and changes requests. Use the Issue Tracking System reference number (ID) if it exists.]

D.4.5.3 Known Defects and Problems

[List all known errors in this version that have not yet been fixed. Use the Issue Tracking System reference number (ID) if it exists. Include workarounds and projected release in which they will be fixed.]

D.4.6 Comments

[Insert any other comments here]