

INFORMATION SOCIETY TECHNOLOGIES (IST) PROGRAMME



Large Scale Monitoring of BroadBand Internet Infrastructure *Contract No. 004336*

Deliverable D1.4: Integrated Architecture Definition

Abstract: In this document the integrated architecture for the Lobster infrastructure is described. Furthermore, the relation with the GN2 JRA-1 project is discussed and integration options are presented

Contractual Date of Delivery	M15
Actual Date of Delivery	September 30 th , 2005.
Deliverable Security Class	-
Editor	Pieter Meulenhoff
Contributors	Pieter Meulenhoff, Hans Hoogstraaten, Rutger Coolen, Panos Trimintzios, Herbert Bos, Baiba Kaskina, Arne Oslebo

The Lobster Consortium consists of:

FORTH-ICS	Coordinator	Greece
VU	Partner	Netherlands
CESNET	Partner	Czech Republic
UNINETT	Partner	Norway
ENDACE	Partner	United Kingdom
Alcatel	Partner	France
FORTHnet	Partner	Greece
TNO	Partner	Netherlands
TERENA	Partner	Netherlands



Document Revision History

Version number	Date	Description of change	People
0.1	16-3-2005	Summary and comparison of JRA-1 performance measurement framework	Pieter J. Meulenhoff Hans Hoogstraaten Rutger Coolen
0.8	12-8-2005	Insertion of various draft sections and figures. Structural additions and improvements. Inclusion of review comments	Pieter J. Meulenhoff Hans Hoogstraaten Rutger Coolen
0.9	15-8-2005	Inclusion of review comments	Rutger Coolen, Hans Hoogstraaten
0.95	23-9-2005	Inclusion of review comments and contributions from the consortium	Rutger Coolen
1.0 (final)	29-9-2005	Inclusion of final round review comments of the consortium	Rutger Coolen



Table of Contents

1	Introduction	5
1.1	What is Lobster?	5
1.2	Objectives	5
1.3	Target audiences	6
1.4	Contents	6
2	Requirements	7
2.1	Existing software and hardware	7
2.2	Using Lobster	7
2.3	WP0.1 “Requirement Collection and Analysis”	7
2.4	Other Lobster activities in WP1 “Monitoring Infrastructure Design”	8
2.4.1	WP1.1 “Anonymization framework definition”	8
2.4.2	WP1.2 “Common Access Platform”	8
2.4.3	WP1.3 “First-tier encryption definition”	8
2.5	Applications	8
2.6	Summary	11
3	Lobster architecture	13
3.1	Components	13
3.1.1	The user	13
3.1.2	The administrator	14
3.1.3	Lobster repository	14
3.1.4	MAPI application	14
3.1.5	Monitoring sensor	15
3.1.6	<i>mapid</i>	15
3.1.7	authd	15
3.1.8	Distributed Execution Environment	15
3.2	Lobster usage scenario’s	15
3.2.1	Getting access to Lobster	15
3.2.2	Starting a MAPI application	16
3.2.3	Starting a ‘Distributed Execution Environment’ application	17
3.2.4	Searching the policy repository	17
4	Integration options for Lobster and GN2 JRA-1	19
4.1	JRA-1 - General Framework Design	19
4.1.1	Security Infrastructure	19
4.1.2	Services	19
4.1.3	Architecture validation	19
4.1.4	Measurement point service	19
4.2	Monitoring scenario’s	20
4.2.1	Monitoring within the same domain	20
4.2.2	Monitoring in a related domain	21
4.2.3	Monitoring in a remote domain	22
4.3	Lobster integrated in GN2 JRA-1	22
4.3.1	JRA-1-MAPI	22
4.3.2	JRA-1-diMAPI	23



4.3.3	JRA-1-MP	24
4.4	Discussion and remarks	25
4.4.1	Lobster applications	25
4.4.2	Service orchestration	25
4.4.3	Control interface	25
4.5	Conclusion	25



1 Introduction

1.1 What is Lobster?

Lobster is a European project [1] that works at establishing a community of network providers, primarily National Research and Education Networks (NRENs), with its participants operating an infrastructure to:

1. Share network monitoring information with each other for purposes such as improvement of Internet security and end-to-end network performance.
2. Combine, analyze and present information for common purposes, such as overall statistics and/ or analysis and visualisation of spread of worms in the participants' networks.
3. Provide data to third parties for purposes agreed by the community (such as Internet Measurement Service or Early Warning Centre).

A participant of the Lobster community by definition runs one or more Lobster measurement points in his network (part of Internet). This Lobster measurement point is part of a larger distributed monitoring infrastructure: the Lobster monitoring infrastructure. The Lobster community is open to Network Providers, NRENs, and Internet Service Providers (ISPs). Also, there are third party users of Lobster such as CERTs, CSIRTs and government bodies. Each Lobster participant can decide what information from within their domain can be measured by other organisations (within the Lobster community or third parties), Lobster provides the means to achieve this. The community's initial policy on Acceptable Use for the Fair Sharing and Use of the Infrastructure is deliverable 0.2 of the Lobster project [2].

The key value of Lobster is that its participants have access to information of their own organisation and other network providers to be able to improve network management and to adequately handle security problems. In return participants provide data to the other participants, where control over who is allowed to see what data remains in the hand of the participant that provides the data. Another value of joining the Lobster community is that the community analyses data of its participants for common purposes, as long as this does not exceed privacy or confidentiality policies of the individual participants.

In summary: the participants of the Lobster community invest to monitor and share information that is beneficial for themselves, for other participants, and for common goals of the community such as a more secure Internet.

1.2 Objectives

The main objective of this document is to describe the integrated architecture of the Lobster monitoring framework. The integrated architecture is based on the integration and alignment of the other tasks within Work-Package 1 (WP1) of the Lobster project, and serves as guiding input for the realization phase of the Lobster project (WP2).

Besides the integration and alignment of activities within the project itself, we set the technical investigation of a possible integration of Lobster and GN2 JRA-1, as another goal.

The goal of Lobster WP1, Monitoring Infrastructure Design, is to identify and address the obstacles that impede the smooth dissemination of collaborative passive monitoring across Europe. This work-package has three dedicated tasks, each targeted at an obstacle. The fourth task, with this document as the resulting deliverable, is targeted at integrating and aligning the three tasks:

- **WP1.1 Anonymization Framework:** Lack of trust among collaborating parties may be the single most important obstacle to the wide dissemination of passive network monitoring across Europe. Indeed, although different parties want to collaborate in order to identify new security attacks, they may be reluctant to exchange their data in raw form. To address privacy concerns related to the



exchange of raw data, network traces need to be anonymized before they are presented to (local or remote) monitoring applications. In addition, different monitoring applications may have different anonymization needs. For example, during the time of a crisis, the chief network administrator of an ISP may have access to full header and payload information of all suspect packets in order to trace the origin of a cyber-attack. On the other hand, the average network administrator may have only access to encrypted data that allow him/her to monitor the network without jeopardizing the privacy of the users. To accommodate all needs, we need a flexible way to anonymize the network packets according to the privileges and requirements of user applications. Such a flexible sanitization can be provided by a Scripting Sanitization Language, SiSaL, and some functions to be used for anonymization by extending the packet processing framework in SCAMPI (and maybe FFPF). This framework will be adopted in collaboration with the proposed end users of the code (NRENs, ISPs, and the like).

- **WP1.2 Common Access Platform:** The wide dissemination and early acceptance of a collaborative passive network monitoring infrastructure across a large number of geographically distributed and possibly heterogeneous sensors can only be based on the existence of a distributed uniform access platform. The purpose of this task is to expedite the widespread acceptance of passive network monitoring by providing such a common access platform. This platform will be realized mainly by customizing the Monitoring Application Programming Interface (MAPI) being developed within SCAMPI, to operate in a distributed world, which will allow different monitoring sensors to talk to each other and exchange information. MAPI provides a rich language which has been used to program only individual passive monitoring systems. DiMAPI will customize MAPI so that it will be able to facilitate the concurrent programming and coordination of several distributed passive monitoring systems.
- **WP1.3 Protection from Intruders:** A main obstacle to the widespread dissemination of passive network monitoring is the reluctance of networking organizations to collect and store large amounts of sensitive data such as those that may be contained in network packet payloads. Indeed, if the computer where the sensitive data are stored is compromised by an attacker, all these data may be accessed by the attacker, and the organization maybe liable for this incident. The purpose of this task is to address these concerns and pave the road towards the easier, wider, and safer dissemination of passive network monitoring across Europe. Indeed, this task advocates that all network traces which will be stored in a computer should be encrypted. Even more, this task suggests that all traces should be encrypted before they reach the computer where they will be processed or stored. Therefore, even if attackers break into the computer they will not be able to access the original (private) information - they will only be able to access the encrypted data

1.3 Target audiences

This document is targeted at the participants of the Lobster project for use as a reference architecture for the “Monitoring Infrastructure Realization” work-package. Other audiences include professionals working on related projects, such as GN2 JRA-1, that have common interest in monitoring and infrastructure realization, as well as potential Lobster community participants to understand the added-value and characteristics of the Lobster infrastructure.

1.4 Contents

This document contains the following chapters:

- **Requirements (Chapter 2):** Requirements serve as input for architecture design, and therefore have to be identified first. Here, input for the architecture requirements comes from the “requirement collection and analysis” (WP0.1) result [3], other Lobster deliverables [4,5,6], and the analysis of monitoring use-cases.
- **Architecture (Chapter 3):** The integrated architecture is presented in Chapter 3. Firstly by describing all the components that exist within the Lobster architecture, and secondly by describing several usage scenarios.
- **GN2 JRA-1 (Chapter 4):** During the start of the Lobster project a related monitoring project GN2 JRA-1 came into view. There is a strong relation between GN2 JRA-1 and Lobster. Both projects are targeted towards monitoring for NRENs; however the work program of the two projects is largely complementary. In this chapter we discuss the technical synergy between the two projects.



We present various integration options for the Lobster and JRA-1 infrastructures. The result is an integrated monitoring architecture based on the building blocks from the GN2 and Lobster projects. Furthermore we discuss some of the integration consequences for the Lobster project and infrastructure.

2 Requirements

The requirements for the Lobster infrastructure are needed because they serve as input for the integrated architecture. In the validation phase, the architecture is checked against the requirements.

The requirements summarised in this chapter originate from several sources:

1. Lobster is based on existing software/ hardware. The MAPI libraries have been developed in the course of the SCAMPI project. Within Lobster we plan to use already existing monitoring boards that have computing capabilities, which can be used to perform various forms of data processing, including packet anonymization. Such boards include the SCAMPI adapter, and the DAG monitoring cards from Endace.
2. The “Requirements Analysis” work-package (WP0) of Lobster has delivered D0.1 (Requirements Collection and Analysis) which identifies the monitoring needs of potential or future Lobster users. The result of this task serves as input for the requirements for the integrated architecture.
3. The other activities: “anonymization framework” [4], “common access platform” [5] and “first-tier encryption definition” [6] within Lobster WP1 each have their requirements for the integrated architecture.
4. A number of potential Lobster applications or use-cases have been identified in order to identify integrated architecture requirements.

In the following sections the requirements from each of these sources are analysed.

2.1 Existing software and hardware

The software to be created should be compatible with or a continuation of earlier work such as MAPI, developed in the course of the SCAMPI project, and the Combo6 monitoring boards.

2.2 Using Lobster

In order to use Lobster, a user should have some knowledge about the sensors. In general a user wants to answer a specific question by measuring network traffic. In order to select the appropriate sensor he must know a subset of the following information:

- The existence of the sensor;
- the administrator of the sensor;
- how to communicate with the sensor in the network;
- some general description of the traffic passing the sensor;
- the anonymization policy for this sensor.

This information should be stored in the Lobster Repository. This repository should be a distributed repository with no central component. Each monitoring site should have a repository which contains a link to other repositories. The Lobster Repository can also be used for other administrative purposes like resource control or statistics about the usage of the sensors.

2.3 WP0.1 “Requirement Collection and Analysis”

The following list shows the relevant highlights from Lobster deliverable D0.1 “Requirement collection and analysis” [3]:

- “In other words, a monitoring system must be general and flexible enough to satisfy different monitoring goals. A similar observation can be drawn about metrics to be monitored [...]. People are interested in different metrics.”
- “Highest level of interest was most frequently indicated for detecting denial of service attacks and for packet loss monitoring.”



- “[...] many people said that they are not willing to share any data and those who are willing to share data prefer to give out only filtered data, packet headers, anonymized data and statistics.”
- “Most partners prefer to make installation themselves, but some partners also want to have it done remotely. Almost all organisations want to operate their monitoring node themselves. However, in case of a failure, most organisations prefer to have a failure helpdesk available, even though people are also willing to co-operate in case of failure.”
- “Regarding anonymization, people prefer to configure the kind of performed anonymization and to choose parties to whom the anonymized traces can be provided and they prefer anonymization to be done directly on the monitoring adapter. The kind of preferred anonymization was the most varying factor in obtained responses and should therefore be highly configurable.”

From these highlights we deduct the following list of requirements

- Generic and Flexible infrastructure that can be fitted to a wide area of applications
- Architecture should support anonymization and access control to data.
- Installation, operation and administration of monitoring sites by and under responsibility of the hosting site in a decentralized environment.

2.4 Other Lobster activities in WP1 “Monitoring Infrastructure Design”

The following paragraphs list the requirements that come out of the three tasks in the Lobster WP1 work-package.

2.4.1 WP1.1 “Anonymization framework definition”

The following list of requirements was derived from the work package “anonymization framework definition” (Work-Package 1.1, [4]):

- To be able to use the data from a sensor authorization must be given to a user for each sensor.
- The authorization for the data can be at different anonymization levels
- In order to do this an authentication mechanism must be added.

2.4.2 WP1.2 “Common Access Platform”

The following list of requirements was derived from the work package “The Common Access Platform” (Work-Package 1.2, [5]):

- For the applications only one application program interface exists.
- There are three different ways application can access a sensor:
 - The application uses one sensor on same machine (communication via UNIX sockets). This is the classic MAPI interface.
 - The application uses one sensor on a different machine; the remote MAPI.
 - The application uses multiple sensors on different machines; the distributed MAPI or DiMAPI.
- The DiMAPI interface has additional logic for combining data from several sensors.
- Instead of using the DiMAPI interface it should also be possible to custom build a sensor by executing code near or on the sensor. This is the so called Distributed Execution Environment (DEE).

2.4.3 WP1.3 “First-tier encryption definition”

No requirements that are specific for the integrated architecture are derived from the work-package “First-tier encryption definition” [6].

2.5 Applications

This section contains a description of several potential Lobster applications, which serve as use-cases for the Lobster architecture. Applications making use of Lobster have different target audiences and may have entirely different functionalities. Therefore we use a systematic approach to classify the described applications. All applications are matched against each of the listed properties in the following list.



- The amount of data transported (to a central application)
- The transported data can be either a continual stream or in bursts
- The need for data to be exchanged across domain borders
- The possibility for decentralised data processing
- The possible reuse of the sensor data for other applications
- The effect of data anonymization
- The location of sensors
- Storage of data for later use

Accurate traffic characterization for programs using dynamic ports

This application will provide accurate distribution of traffic to applications and will work even for applications that use dynamic ports to communicate, such as peer-to-peer systems. In contrast to our application, traditional traffic characterization methods, such as netflow and IPFIX, are based on static ports and thus are not able to categorize packets belonging to applications that use dynamically generated ports.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Possibly streaming (partly centralised analyses).
- Streaming data: No.
- Inter domain data exchange: None.
- Decentralised data processing: Yes.
- Reuse of the sensor data: No. The sensor produces reporting signals only.
- Anonymization: Yes. The data can be anonymized.
- Sensors locations: Specific. All the traffic of the target must pass through one sensor.
- Data storage: None.

*) This depends on the setup of the system. When the traffic of interest (the internet sessions) is not captured by only one sensor, huge amounts of data must be analysed centrally or some form of storage must be used.

Spread of zero-day worms

Based on traffic captures at several different sensors, this security-related application will focus on finding worms as soon as they start to spread on the Internet, so as to provide an early-warning system.

The requirements for the applications depend on the detection method used. Further studies are required. Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: The amount of data varies and depends on the method used and can be minimal (packet characterization only) to huge (complete packet streams).
- Streaming data: Yes.
- Inter domain data exchange: The inter domain data exchange can be minimal, except when a non-MAPI function is required for the analysis.
- Decentralised data processing: Yes.
- Reuse of the sensor data: Specific traffic is measured, data probably can't be used for other means.
- Anonymization: The data can often be anonymized, however for some purposes raw data will be required.
- Sensors locations: Several key sensors possibly across domains.
- Data storage: Storage of data, for anomaly detection (can also be handled by the application).

European Internet measurement service



Virtually any kind of network statistics can be measured at a European scale. This would include information about the kind of services that are most-used by citizens, network security information, quality-of-service, social-cultural and behavioural information, and the use of encrypted channels. The information could benefit member states policy-development, various areas of research, as well as network planning within Europe.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Limited amount of data (statistics only)
- Streaming data: Continues stream of data to the application
- Inter domain data exchange: All the data.
- Decentralised data processing: Yes.
- Reuse of the sensor data: The application on the sensor can be used for other means as well
- Anonymization: The data can be anonymized.
- Sensors locations: Several key sensors must be in place to get the best picture
- Data storage: No.

End-to-end performance debugging

Debugging end-to-end performance is a difficult task. Using the Lobster infrastructure to look at test traffic in different location in the path it will be possible to identify where packet loss, packet reordering, packet delay etc. occurs.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Amount of data can vary from minimal to huge (depending on the performance test)
- Streaming data: Continues stream of data for limited period (during debugging)
- Inter domain data exchange: Sensors across multiple domains should be used.
- Decentralised data processing: Possibly
- Reuse of the sensor data: No, specific traffic is measured.
- Anonymization: The data can be anonymized.
- Sensors locations: Several sensors across the path
- Data storage: No.

Applications performance measurement*

Passive measurement can be used for measuring the performance of some applications or services on the Internet. One example of this is DNS measurements. The Lobster infrastructure can give a distributed view of the performance for such services.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Amount of data can vary from minimal (ex. DNS request only) to huge (ex. video streaming application)
- Streaming data: Continues stream of data to the application, for real time detection
- Inter domain data exchange: The inter domain data can be intensive (depended on the application under test).
- Decentralised data processing: Possibly.
- Reuse of the sensor data: The application on the sensor can probably be used for other means as well.
- Anonymization: The data can be anonymized in most cases.
- Sensors locations: Each sensor should be close to the server in order to detect all traffic with the server. For each specific performance measurement an different set of sensors should be used.
- Data storage:

*) This type of performance measurement can probably be done more efficiently with an active measurement method, although these influence network performance.



Detect and Trace DOS attacks

DOS attacks often have spoofed source IP addresses. The Lobster infrastructure can help trace the origin such attacks.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Limited amount of data (e.g. ip header and sensor address).
- Streaming data: Continues stream of data to the application, for real time detection.
- Inter domain data exchange: The inter domain data exchange is minimal (attack information only).
- Decentralised data processing:
- Reuse of the sensor data: The application on the sensor can be used for other means as well.
- Anonymization: The data can be partially anonymized.
- Sensors locations: Closed sensor network on the edge of the own network, for backtracking the attacker.
- Data storage: Storage of data, for backtracking (can also be handled by the application).

Test platform for new IPFIX attributes

The IPFIX protocol allows for new attributes to be added to a flow record. One problem is that getting new attributes implemented in routers will be difficult and time consuming. The Lobster platform will be a good platform for large scale testing and proof of concept for new attributes.

Some requirements of the Lobster infrastructure for this application:

- Amount of data transported: Similar to NetFlow exports from routers.
- Streaming data: Yes.
- Inter domain data exchange: No inter domain data exchange in general. Only for specific attributes.
- Decentralised data processing: No.
- Reuse of the sensor data: No, sensor application is very specific.
- Anonymization: The data can be anonymized.
- Sensors locations: Multiple sensors will usually be needed. One sensor can be sufficient. Application should be close to (or in) the sensor.
- Data storage: to Disk for later processing.

2.6 Summary

From the various sources for requirements, as described in the previous sections, we attempt to extract a set of requirements that should be met by the integrated architecture of the Lobster infrastructure. The following requirements for the Lobster architecture are recognized:

- R1. Lobster should be general and flexible to satisfy different monitoring goals and generate different metrics.
- R2. Information about Lobster sensors should be available for potential users.
- R3. Participants who share data to Lobster must be able to authorise individual users for each sensor.
- R4. Participants who share data to Lobster must be able to anonymize data to all or specified users using the proper policy.
- R5. Unauthorized persons cannot make use of Lobster data.
- R6. The installation, operation and administration of monitoring sites in a decentralized environment must be possible.
- R7. There should be only one programming interface to Lobster sensor(s) namely diMAPI
- R8. A possibility should exist to execute custom build code on or near a Lobster sensor.
- R9. A sensor should be able to continuous send large amounts of data to an application.
- R10. It should be possible to collect data from Lobster sensors in various different domains.
- R11. It should be possible for a sensor to send different data to different users at the same time



- R12. It should be possible for an application to have un-anonymized data at its disposal. However, this data must only be available to authorised users and applications.



3 Lobster architecture

This Chapter describes the integrated Lobster architecture. In this Chapter we first present an overview of the architecture, and a description of each of the components. Then, in the form of sequence diagrams, the use of the Lobster architecture is described. We must underline here that we do not intend to give a detailed description of the internal structure of each component in this chapter. For the detailed component descriptions we refer to the relevant documentation of the components [4,5,6].

Figure 1 presents an overview of the Lobster architecture and its constituent components. The figure presents two administratively distinct domains, organizations or domains: A and B, that operate together to form a Lobster measurement (this set-up is chosen to address R6). From now on we name A and B as domains, although this may not be valid in the actual application of Lobster. A is the domain where the “monitoring sensor” resides. The user, located in domain B wants to make use of the “monitoring sensor” by using his/her monitoring application i.e.: “MAPI application” or “DEE Application”.

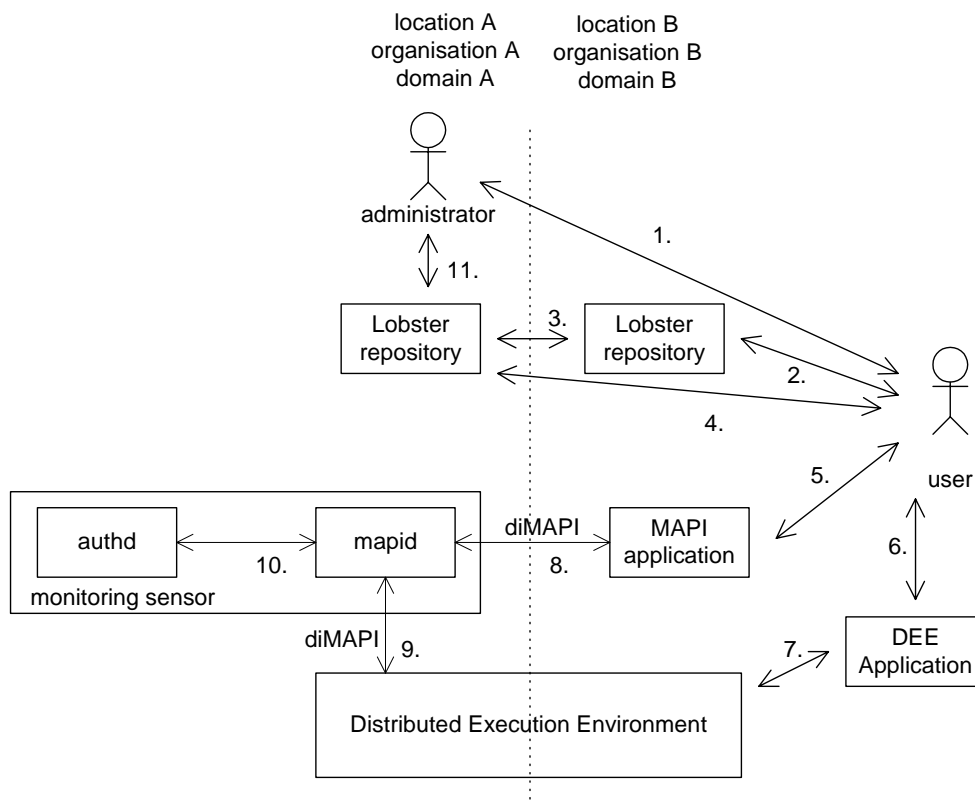


Figure 1. Lobster Integrated Architecture

3.1 Components

We now proceed by describing the components in more detail.

3.1.1 The user

The user, located in domain B, who wants to make use of the Lobster infrastructure, is typically a network manager, network engineer, or university researcher. The user wants to run his monitoring application and through that start a measurement at one or more “monitoring sensors” in his or another domain. To make this possible, the user must be able to execute the following tasks presented by the arrows in Figure 1:



- Query the “Lobster Repository” (2.) in order to find the location of a “Monitoring Sensor” that has the required capabilities to deliver the required monitoring data, and could be used by the user’s application. The user contacts a known repository to execute his, possibly located in his own domain and, through that (2), query other repositories in order to find the wanted “monitoring sensor”.
- Contact the administrator responsible for a monitoring sensor (1.), in order to request credentials for the monitoring sensor that the user wants to use in his application. At this moment of writing the protocol for this communication is not defined and thus could be performed in various ways: (e)mail, telephone, vis-à-vis,... Upon success the credentials to get access to the “monitoring sensor” contain the monitoring policy that defines how the monitoring sensor must be used: i.e. which anonymization functions are required.
- The credentials can be retrieved from the Lobster repository located in the domain that is responsible for the monitoring sensor. After the administrator has granted access to the user for a monitoring sensor, the credentials are stored in the “Lobster repository” (11.). The user can now retrieve the credentials from the “Lobster Repository” (4.).
- Start a measurement application (5. and 6.) that makes use of a remote “monitoring sensor”. Such an application can either be using the (Di)MAPI libraries, forming a MAPI application(8.), or it can be built on top of a distributed application framework, that is supported by the monitoring sensor (7.). The latter case allows moving an instance of the distributed application, for example for efficient processing of measurement data, closer to the monitoring sensor.

3.1.2 The administrator

The administrator, located in domain A, is responsible for managing the monitoring sensor(s) and repositories in his/her domain. The administrator has the following tasks

- Configure monitoring policies, which are stored in the Lobster repository. Upon user request, the administrator configures a policy in such a way that the user is allowed to use it. The created credentials are stored in the Lobster repository (11.) so that they can be retrieved by the user (4).
- Store information about the available monitoring sensors in the Lobster repository (11.). This information contains the capabilities of the “monitoring sensor”: network(s), capacities, and so forth.
- The location of other Lobster repositories, in the form of a reference, is stored by the administrator in his Lobster repository (11.).

3.1.3 Lobster repository

The Lobster repository (requirements R2, R3, R4) is a service that allows storage (by administrator) and retrieval of monitoring policies, information about the location and capabilities of monitoring sensors in the domain, and locations of Lobster repositories in other domains. All information in the “Lobster repository” is publicly available. This is also the case for monitoring policies, the specification which operations are allowed by a user. Because these are digitally signed using a public key system, only the applicable user has the ability to use the policy. Creating, changing and deleting of information in the policy repository are restricted to the administrator.

3.1.4 MAPI application

The MAPI application (requirements R7) uses the diMAPI libraries and protocols to communicate with a monitoring sensor (8.). Through the MAPI application, the user sends the credentials, and a digitally signed message using the user’s private key, to the monitoring sensor. Besides this, the MAPI application should be configured in such a way that all the monitoring policies are satisfied. This means that the correct anonymization functions have to be applied. Before starting the application, the “monitoring sensor” validates the applied anonymization functions against the monitoring policy, part of the credentials. Upon success, the measurement is started.

The diMAPI libraries and protocols enable the application to collect data from various Lobster sensors (R10).



3.1.5 Monitoring sensor

The monitoring sensor, located in domain A is a system that exposes a service, responsible for making passive network measurements with small or large volumes of measurement data as a result, either continuous or in bursts (R9). According to R1 and R11 the monitoring sensor should be capable of supporting different monitoring goals and metrics in parallel. The monitoring sensor consists primarily of two components: *mapid* and *authd*. While the component *mapid* is responsible for executing a measurement, the component *authd* is responsible for checking the validity of the user's credentials. This together with the administration of Lobster credentials in the repository addresses R5. Finally, the monitoring sensor is capable of anonymizing data when required, or it can ensure secure transport of unanonymised data when required (R12).

3.1.6 *mapid*

Mapid is the MAPI daemon that is responsible for executing the network measurement on the available hardware monitoring system and communication with the MAPI application (8.), and running the specified network measurements. When *mapid* passes the credentials received from the MAPI application, to *authd* in order to be validated. *mapid* itself checks the configured anonymization functions, as instructed by the MAPI application, against those specified in the monitoring policy (part of credentials). If the applied anonymization adheres to the specified policy, and *authd* successfully validates the credentials, the measurement can be started, and data is returned to the MAPI application.

3.1.7 *authd*

authd is the component that validates the identity of the user using a public key system, such that only authorised users can use a sensor. The result is passed to *mapid*.

3.1.8 Distributed Execution Environment

The Distributed Execution environment (DEE) is targeted at applications that require significant processing close to the "monitoring sensor" that goes beyond filtering or anonymization which can be carried out by *mapid*. Therefore an instance of DEE is located in the vicinity of the sensor in order to facilitate fast communication (this addresses R8). The communication between this DEE instance and *mapid* takes place in the same way as a standard DiMAPI application. Another instance of the DEE application is located in the domain of the user (A).

3.2 Lobster usage scenario's

In this section, the process flow of several scenarios is described. These scenario's are: getting access to Lobster, starting a MAPI application, starting a Distributed Execution Environment application and searching the policy repository.

3.2.1 Getting access to Lobster

In this scenario, a user contacts the administrator in order to get access to a monitoring sensor. The process flow of this scenario is depicted in figure 2.

1. The administrator is responsible for specifying the anonymization policies which are stored in a policy repository. The administrator issues credentials for potential users of a monitoring sensor, which enforce the use of certain anonymization functions. A user is allowed to use the monitoring sensor after application of all anonymization functions specified in their credentials.
2. A user that wants to use the monitoring sensor must first acquire the necessary credentials for that sensor. Credentials delegate authority to a user (or a user group) identified by a public key (or a set of public keys). Thus, the user first has to deliver his/her public key to the administrator. The administrator then signs the credentials and stores them in the Policy Repository (2a). Since the credentials are digitally signed, they can be easily distributed over untrusted networks, the user can safely download them from the Policy Repository (2b, 2c). Thus the credentials include the public key of the user and the anonymization policy that the user must adhere to in order to get access to the monitoring sensor.

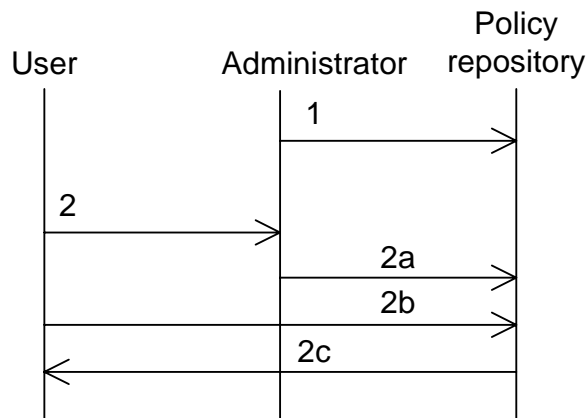


Figure 2. Getting access to Lobster

3.2.2 Starting a MAPI application

In this scenario, the user starts a MAPI application that makes use of a monitoring sensor. the application can only be started when all steps of ‘Getting access to Lobster’ are successfully completed. The process flow of the scenario is depicted in figure 3.

1. The user then configures his/her MAPI application by applying the required functions, resulting in a flow specification that contains all information about the requested flow, and which anonymization functions should be applied (1). The active MAPI application delivers the flow specification, and the credentials of the user to the mapid component of the monitoring sensor (1a). Mapid subsequently passes this information to the authd component (1b). Authd checks the authentication of the user and evaluates the specification of the flow for compliance with the supplied credentials (i.e., which include the anonymization policy). Authd then returns the result of the evaluation to mapid (1c). If the flow specification complies with the supplied credentials and the authentication is successful, mapid activates the flow and returns a handle to the MAPI application (1d). Otherwise the flow is not activated, and the request is rejected. Depending on the functionality, the MAPI application processes the incoming flow and represents this to the user (1e).
2. Data exchange between MAPI application and mapid.
3. Starting a MAPI application that makes use of more than one monitoring sensor.

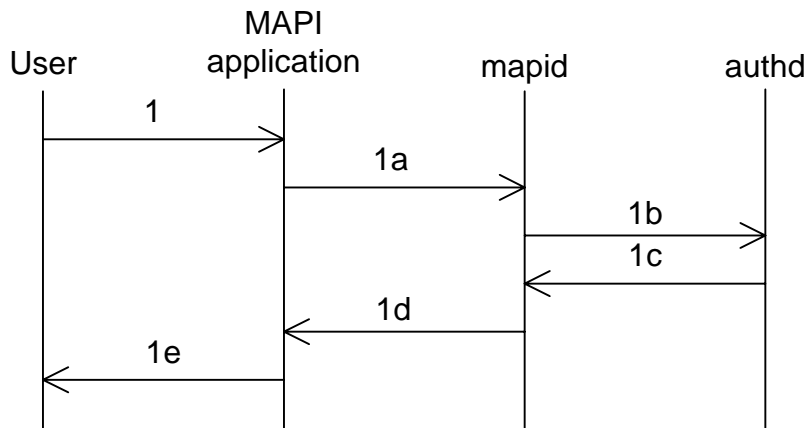


Figure 3. Managing policies and starting a Lobster application

3.2.3 Starting a ‘Distributed Execution Environment’ application

In this scenario, the user starts a distributed application that makes use of a monitoring sensor. the application can only be started when all steps of ‘Getting access to Lobster’ are successfully completed. The process flow of the scenario is depicted in figure 4.

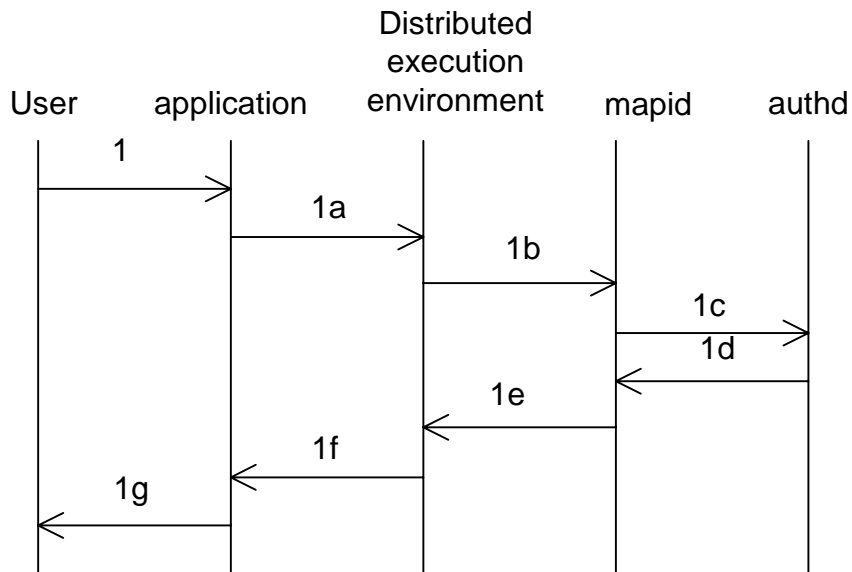


Figure 4. Starting a distributed application

3.2.4 Searching the policy repository

This scenario the user queries the policy repository in order to find the location of other policy repositories, the location of monitoring sensors, and the user credentials for getting access to a monitoring sensor.

1. contact a policy repository and make a query for credentials, location of a monitoring sensor.
2. contact a policy repository and search another repository, subsequently contact the remote repository and make query.

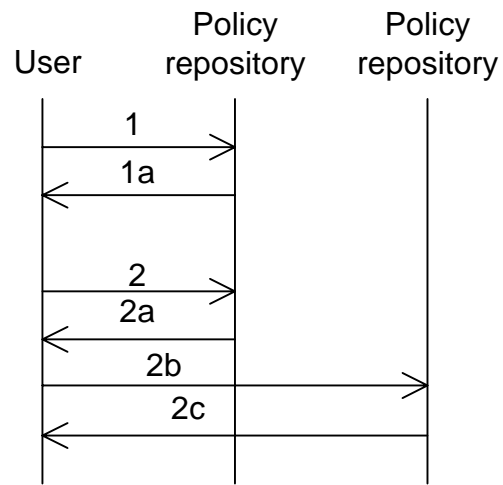


Figure 5. Searching the policy repository



4 Integration options for Lobster and GN2 JRA-1

During the start of the Lobster project a related monitoring project GN2 JRA-1 came into view. For a successful dissemination of Lobster it would be beneficial if the project's result are interoperable with the GN2 JRA-1 results and preferably form an integral part of the JRA-1 infrastructure.

The joint research activity #1 (JRA-1) document "General Framework Design" [7] describes the architecture of a performance measurement framework for the GN2 and Internet2 community. This architecture is based on a service-oriented architecture (SOA). SOA is an architectural style with the goal to achieve loose coupling among interacting services. A service is a unit of work done by a service provider to achieve desired end-results for a service consumer.

4.1 JRA-1 - General Framework Design

In this section various aspects of the General Framework Design of the JRA-1 monitoring infrastructure are described.

4.1.1 Security Infrastructure

The security infrastructure is a GN2 JRA5 activity. This security infrastructure seems to be targeted at solving inter-domain related problems. The following list is a summary of the most important 'features':

- A trust relationship can be made between domains which facilitates the use of services between domains.
- Upon authentication, a user receives a token that is valid for the use for a specific service. The service uses this token to retrieve user attributes (rights/roles). Such a mechanism ensures user privacy.

4.1.2 Services

The architecture of JRA-1, as already mentioned in the introduction, is based on a service-oriented architecture. The following list is a summary of the available services:

- **Measurement Point Service (MP)**. The Measurement Point (MP) service creates measurement data either by initiating active measurement tests or querying passive measurement devices.
- **Measurement Archive Service (MA)**. A Measurement Archive (MA) service is used to store and publish historical monitoring data produced by MP services and/or Transformation services.
- **Transformation Service (TrS)**. The Transformation service is used to pipeline data between the other services within the framework. It can be used to perform any function upon data
- **Lookup Service (LS)** The Lookup service enables users to discover other services (MP, MA, Authentication service, etc.) that can satisfy their needs. The lookup service is the entry point for the framework.
- **Authentication Service (AS)**. The Authentication service provides the authentication functionality for the framework as well as an attribute authority (attribute authority is a component of the authentication server that decide which attribute can be disclosed to a resource)
- **Resource Protector Service (RP)** The Resource Protector (RP) service is used to arbitrate the consumption of limited resources, such as network bandwidth.
- **Topology Service (TS)** The Topology service is used to make topological information about the network available to the framework.

4.1.3 Architecture validation

In the "General Framework Design" document, the proposed architecture is validated using a list of existing tools. The idea behind this is that the framework serves as a sort of wrapper around the existing tool. The main focus during validation is in the Measurement Point service (MP) in the case of active and passive measurements. At this moment the validation is an inventarisation of existing tools.

4.1.4 Measurement point service

This section describes the details of the Measurement Point service (MP). The MP consists of several components (depicted in **Error! Reference source not found.**). These components are:



1. The Request Manager (RM) is responsible for communication with the other services. The RM distributes measurement data to the subscribed clients.
2. The Information Service (IS) component contains configuration and descriptive information of the service.
3. The Authentication Manager (AM) component is responsible for authenticating request. It accepts tokens and ensures that they are valid.
4. The Resource Manager (RsM) is responsible for accepting/rejecting requests. Based on a policy, or reservation schedule. The RsM may contact the Resource Protector service (RP).
5. The Measurement Executor (ME) is responsible for controlling the measurement and redirecting data to the request manager.

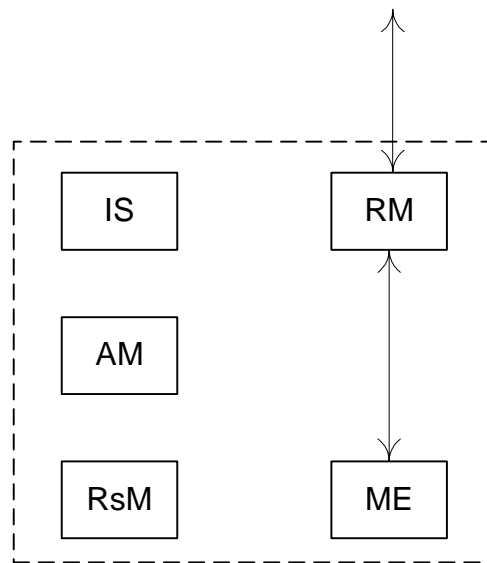


Figure 6. Measurement Point Service

At this moment, a set of six different measurements, each with its own interface are defined.

- Active delay, loss, and jitter measurements
- Processed information retrieved from network equipment
- Flow-based measurements
- Active stress-type achievable bandwidth measurements
- Active probe-type available bandwidth measurements
- Passive packet traces capturing

4.2 Monitoring scenario's

Here we describe two monitoring scenarios: Monitoring within the same domain, and monitoring in a related domain. A related domain has a trust relationship with the originating domain.

4.2.1 Monitoring within the same domain

In this case, the application, and all required services are located in the same domain. In the monitoring process, the following steps will be passed

1. Client application contacts a known lookup service in order to find a measurement point service (MP). The lookup service returns a location for the MP and a location of an authentication service that is valid for the MP.
2. The client application goes through an authentication and authorization phase, and finally receives a token, that is valid for the MP.
3. The client application connects to the MP, using the received token, in order to start the measurement.



4. The MP connects to a resource protector service (RP) in order to check if the requested operation/usage of the MP is allowed.
5. If needed, the MP can contact the AS to request more information (rights, roles, attributes) in order to determine if a certain operation on the MP is allowed.

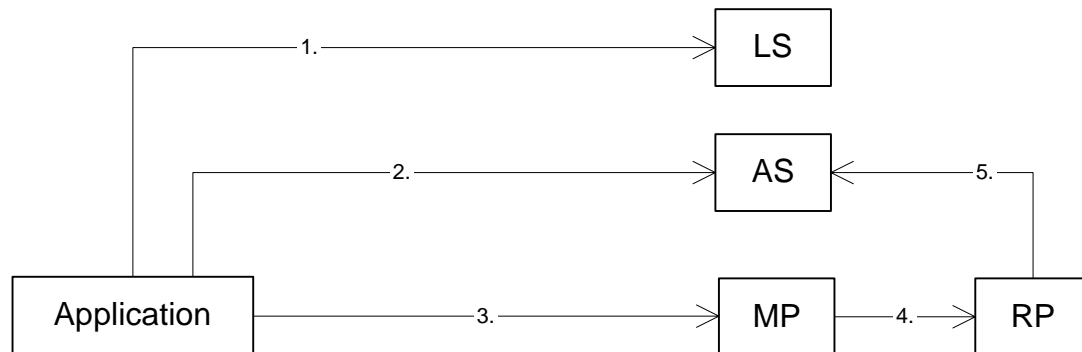


Figure 7. Monitoring within the same domain

4.2.2 Monitoring in a related domain

In this case, the application and the required services are located in different domains. The authentication servers in both domains have trust relationship. In the monitoring process, the following steps will be passed.

1. The client application contacts a known lookup service in order to find a measurement point service (MP), The lookup service returns a location for the MP and a location of an authentication service, that is valid for the MP.
2. The client application goes through an authentication and authorization phase on the local AS, that has a trust relationship with a remote AS, and finally receives a token, that is valid for the MP.
3. The client application connects to the MP, using the received token, in order to start the measurement.
4. The MP connects to a resource protector service (RP) in order to check if the requested operation/usage of the MP is allowed.
5. If needed, the MP can contact the AS to request more information (rights, roles, attributes) in order to determine if a certain operation on the MP is allowed.
6. All user information is stored in the local AS, thus requests are forwarded to the local AS.

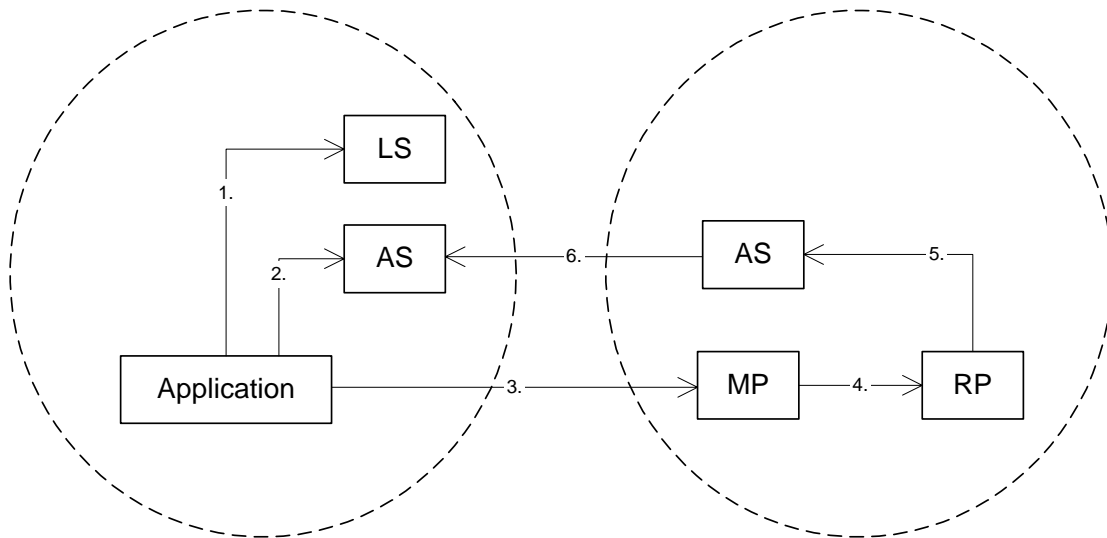


Figure 8. Monitoring in a related domain

4.2.3 Monitoring in a remote domain

Now there is no relation between the local and remote domain. In this case, the user, using the application, has to be known in the remote AS. This makes this case more or less comparable to monitoring in the same domain.

4.3 Lobster integrated in GN2 JRA-1

As GN2-JRA-1 and Lobster can operate in the same domain (network management) and have common needs, it is logical to compare both architectures and see where integration or re-use of components is possible.

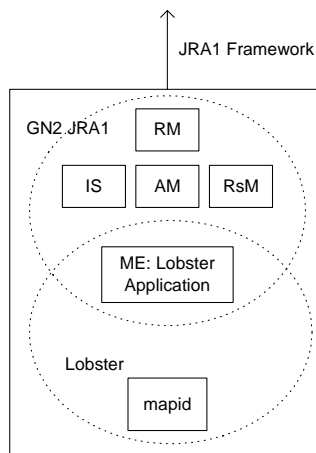
We describe the integration between GN2 JRA-1 and Lobster in the form of a JRA-1 measurement point (MP) and by exploring three different scenarios where each scenario has a different integration level.

1. **JRA-1-MAPI:** GN2 JRA-1 Measurement Point is an application or service based on the standard MAPI (non-distributed) libraries. One Lobster node is mapped to one JRA-1 MP. The Lobster measurements are conducted on the same machine as where the JRA-1 MP is active.
2. **JRA-1-diMAPI:** GN2 JRA-1 Measurement Point is an application or service based on the distributed/network MAPId libraries. One JRA-1 MP can map to one or more Lobster nodes.
3. **JRA-1-MP:** The MAPId communication protocol is designed in such a way that it is compatible with JRA-1 MP communication protocols. This means that a Lobster node becomes a JRA-1 MP.

The following sections discuss these three integration scenarios in more detail.

4.3.1 JRA-1-MAPI

In this case, Lobster is integrated into JRA-1 by constructing a JRA-1 MP based on the MAPI library without support for remote network monitoring. Basically this means that the JRA-1 MP and the Lobster monitoring node are active on the same system. This scenario is graphically depicted in the following figure.



In this figure, the two dotted circles present the GN2 JRA-1, and the Lobster environments. The boxes: RM, IS, AM, and RsM represent the same components as defined within a JRA-1 MP. The measurement executor (ME) is in fact a Lobster application which is responsible for instructing *mapid* to conduct a network measurement.

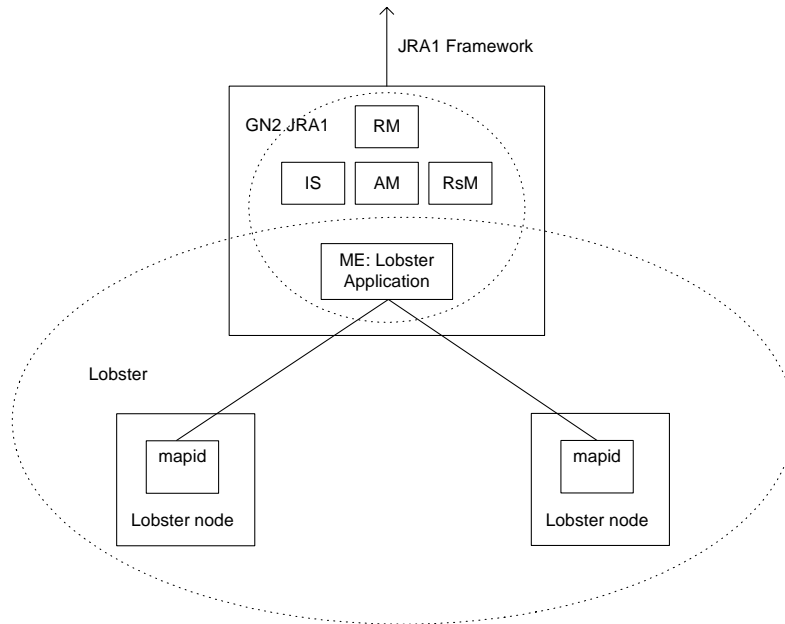
The major advantage, of this scenario, is that is built using existing components¹. Implementation of JRA-1 MP based on this scenario should be relatively straightforward.

We expect a major drawback in the case of executing a 'distributed monitoring measurement' where multiple Lobster enabled MP's work together to generate data, the integration of data needs to be done by the application making use of the JRA-1 framework, and thus takes place at a relatively high level making the tackling of these integration problems more difficult.

4.3.2 JRA-1-diMAPI

In this case, Lobster is integrated into JRA-1 by constructing a JRA-1 MP based on the diMAPI library, which includes support for remote network monitoring. Basically this means that the JRA-1 MP can connect to multiple Lobster monitoring nodes at the same time. This scenario is graphically depicted in the following figure.

¹ This is valid for Lobster. At the time of writing there is no JRA-1 implementation.

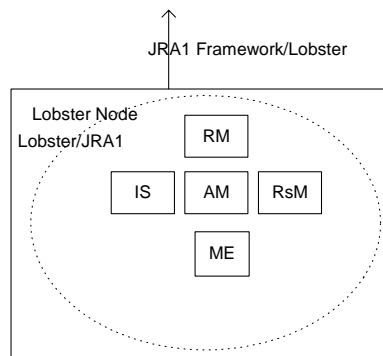


In this figure, the two dotted circles present the GN2 JRA-1, and the Lobster environments. The boxes: RM, IS, AM, and RsM represent the same components as defined within a JRA-1 MP. The measurement executor (ME) is in fact a Lobster application which is responsible for instructing *mapid* to conduct a network measurement. Major difference compared to the previous scenario is that now multiple Lobster nodes can be attached to the same JRA-1 MP.

Where integration of measurement data was difficult in the previous scenario, where one Lobster monitoring node is mapped to one MP, it has now become an advantage, since Lobster has complete control over the setup and proceeding of a distributed monitoring application. Disadvantage of this scenario is that such an environment has to be planned very carefully. For example resource control at JRA-1 level may not be valid anymore when other Lobster/JRA-1 applications are making use of a Lobster node at the same time, ultimately resulting in a conflict.

4.3.3 JRA-1-MP

In this case, the Lobster framework is integrated into JRA-1 by constructing a Lobster node, that is completely JRA-1 compatible. This means that the protocols used for diMAPI are valid within JRA-1, and that the support functions like authentication, resource planning and control are compatible with JRA-1, running under responsibility of the *mapid*. This scenario is graphically depicted in the following figure.





As can be seen in this figure, the Lobster architecture is completely dissolved within GN2 JRA-1. The boxes: RM, IS, AM, ME, and RsM represent the same components as defined within a JRA-1 MP. Difference is now that all these components have to be built by Lobster, and thus providing the necessary functionality to be compatible with GN2 JRA-1. Lobster applications on the other hand, make use of JRA-1 network and protocols to make use of a monitoring node.

The scenario is the most generic, since both frameworks are completely integrated, but this will have several consequences. As is the case with the JRA-1-MAPI scenario, the integration of data coming out of several independent 'Lobster' MP's is expected to integrate more difficult than at a lower level. Also, the implementation of typically Lobster functionality like: anonymization, advanced filtering, and remote execution of code may be difficult to implement on the basis of just the GN2 JRA-1 framework.

4.4 Discussion and remarks

4.4.1 Lobster applications

The majority of applications described in Chapter 2 require the usage of multiple measurement points and multiple domains. We want to address some issues

- Applications must be able to use multiple identities at the same time, when using MP's from different domains.
- In JRA-1 MP's are expected to be 'dumb' devices that generate a limited set of performance metrics. The majority of data-processing is expected to run in Transformation Services (TrS), not in the MP. Some Lobster applications on the other hand require processing as close to the data-stream as possible (in hardware).

4.4.2 Service orchestration

The service oriented architecture (SOA) as presented in the JRA-1 performance monitoring framework document introduces a number of services that need to be integrated within the client application. The client application is responsible for controlling the interoperation of the different services. E.g. client application instructs MP1 to send data to TrS2, TrS2 is instructed to do some calculation and send result to MA3. In an environment that is targeted at solving problems that need distributed applications, the interoperation between the different services can become complex. This expresses the wish to use service orchestration in the form of BPEL (business process execution language) processes. BPEL is an XML specification for web-services orchestration, specifying the workflow and routing between involved services.

4.4.3 Control interface

The architecture presented in JRA-1 could be used as a starting point for future changes in the implementation DiMAPI. We suggest that distributed MAPI could evolve to two interfaces: control and data. The control interface could be implemented according to the specifications of JRA-1. Probably a XML over HTTP based service. The data interface could be specifically designed in such a way that is targeted at efficient packet trace transfer

4.5 Conclusion

In this report the components of Lobster have been integrated into a single architecture. Requirements for the component integration are analysed in Chapter 2. And the relation between the components is described in Chapter 3 by using usage scenarios. Finally the Integrated Architecture is analysed with respect to possible integration with JRA-1. The conclusion is that there are 3 options for integration, with varying complexity in the required changes to either LOBSTER or JRA-1.

The Integrated Architecture Definition (this document) forms the baseline for the further design and implementation of the Lobster infrastructure within the Lobster project. Parts of the architecture might evolve in time and updates of this document might be necessary after the pilot stages of Lobster.

Any contributions, remarks and suggestions with respect to the Lobster project are welcomed by the consortium. Contact information can be found in reference [1].



References

- [1] www.ist-Lobster.org
- [2] D0.2 - Acceptable Use Policy for the Fair Sharing and Use of the Infrastructure, www.ist-Lobster.org
- [3] D0.1 - Requirement collection and analysis, www.ist-Lobster.org
- [4] D1.1a - Anonymization Framework Definition - 25 July 2005, www.ist-Lobster.org
- [5] [D1.2 - Common Access Platform Definition](#) - 25 July 2005, www.ist-Lobster.org.
- [6] [D1.3 - First-tier Encryption Definition](#) - 21 July 2005, www.ist-Lobster.org
- [7] General Framework Design, draft document, JRA-1 project.