

Nokia Research Center / Helsinki  
Dirk Trossen, Dana PavelFINAL  
24.10.06**N-RSA HIGH-LEVEL SYSTEM ARCHITECTURE: FUNCTIONALITY & INTERFACE  
DESCRIPTION**

Authors: Dirk Trossen, Dana Pavel  
Nokia Research Center / Helsinki

Editor: Elena Balandina  
Nokia Research Center / Helsinki

**ABSTRACT**

This document defines the interfaces used between the middleware components within the Nokia Remote Sensing Architecture (N-RSA). For each of the interfaces, the functionality and involved components are described in more detail.

Version:	1.0
Authors:	Dirk Trossen, Dana Pavel

Nokia Research Center / Helsinki  
Dirk Trossen, Dana Pavel

FINAL  
24.10.06

**GLOSSARY**

CORBA	Common Object Request Broker Architecture
fka	formerly known as
N-RSA	Nokia Remote Sensing Architecture
SIP	Session Initiation Protocol
XCAP	XML Configuration Access Protocol
XML	Extensible Markup Language

**TABLE OF CONTENTS**

**1. INTRODUCTION..... 4**

**2. FUNCTIONALITY OF EACH COMPONENT ..... 4**

2.1 EVENT COMMUNICATION PARADIGM IN N-RSA ..... 5

2.2 EVENT DELIVERY COMPONENT ..... 7

2.3 ACQUISITION COMPONENT ..... 8

2.4 QUERY RESOLVER (FORMERLY KNOWN AS REPRESENTATION) COMPONENT ..... 9

2.5 AGGREGATION COMPONENT ..... 10

2.6 ACCESS CONTROL COMPONENT ..... 10

2.7 STORAGE COMPONENT ..... 10

2.8 CODE REPOSITORY COMPONENT ..... 11

2.9 REGISTRATION & AVAILABILITY COMPONENT ..... 11

**3. INTERFACES BETWEEN THE COMPONENTS ..... 12**

3.1 THE RAPI INTERFACE (N-RSA API) ..... 12

    3.1.1 *The RAPI<sub>ARA</sub> interface* ..... 12

    3.1.2 *The RAPI<sub>AA</sub> interface* ..... 13

3.2 THE E<sub>D</sub> INTERFACE ..... 14

3.3 THE R<sub>A</sub> INTERFACE ..... 15

3.4 THE A<sub>C</sub> INTERFACE ..... 15

3.5 THE SSI<sub>A</sub> INTERFACE ..... 16

3.6 THE SSI<sub>R</sub> INTERFACE ..... 16

3.7 THE E<sub>D(LOCAL)</sub> INTERFACE ..... 17

3.8 THE A<sub>RA(LOCAL)</sub> INTERFACE ..... 17

3.9 THE A<sub>CTRL(LOCAL)</sub> INTERFACE ..... 18

3.10 THE A<sub>A(LOCAL)</sub> INTERFACE ..... 18

3.11 THE A<sub>Q(LOCAL)</sub> INTERFACE ..... 19

3.12 THE A<sub>S(LOCAL)</sub> INTERFACE ..... 19

3.13 THE C<sub>R</sub> INTERFACE (OPTIONAL) ..... 20

3.14 THE R<sub>A(AS-RS)</sub> INTERFACE (OPTIONAL) ..... 20

3.15 THE C<sub>A(LOCAL)</sub> INTERFACE (OPTIONAL) ..... 21

3.16 THE C<sub>RA(LOCAL)</sub> INTERFACE (OPTIONAL) ..... 21

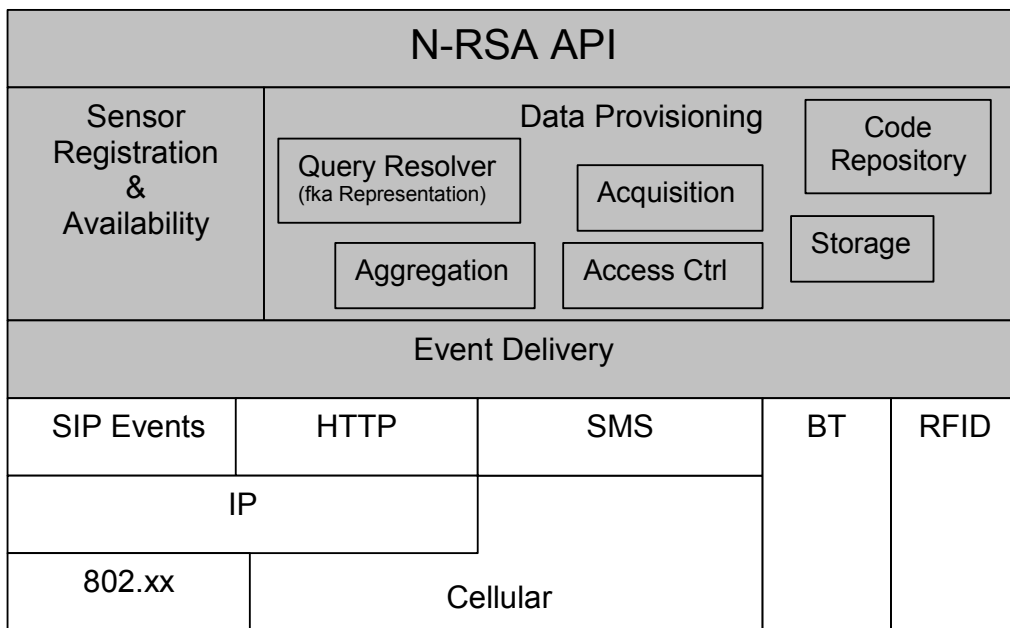
**4. REFERENCES ..... 22**

**1. INTRODUCTION**

This report outlines the functionality of the middleware components in the Nokia Remote Sensing Architecture (N-RSA), which was defined on high level in [3]. Furthermore, the interfaces between each component will be described in this report. We will further outline the candidates for the particular communication protocols to be used (note that, based on the requirements in [3] some of these candidates are optional only).

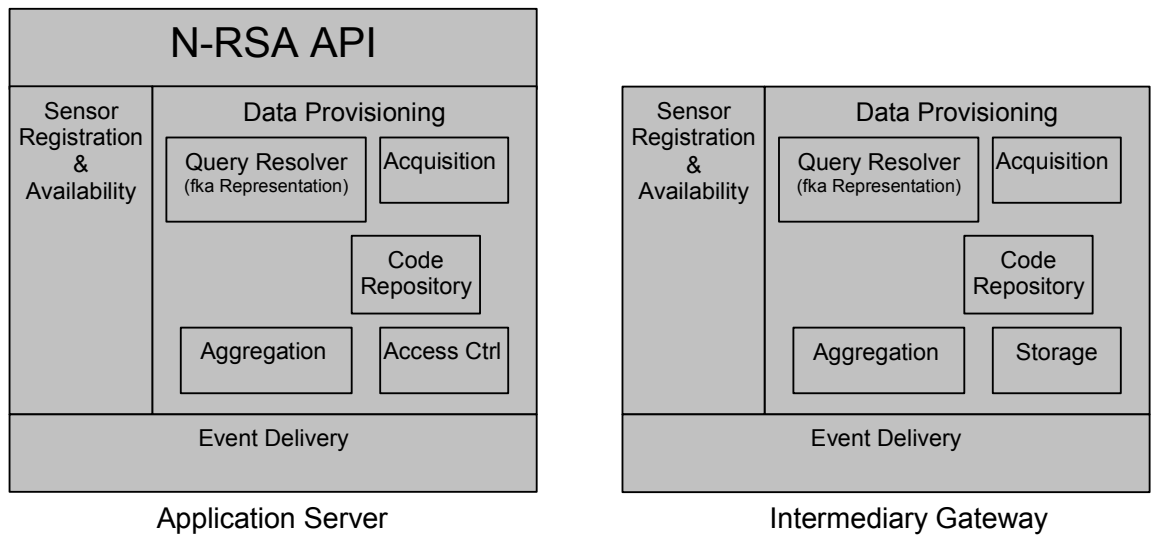
**2. FUNCTIONALITY OF EACH COMPONENT**

The functionality of each middleware component (see Figure 1 below) is specified through the requirements defined in [3].

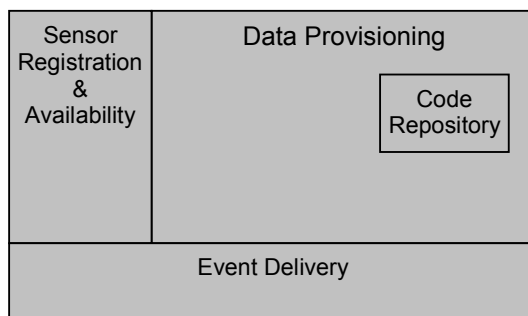


**Figure 1** Middleware Components

This functionality is realized within a distributed architecture, as outlined in Figure 1 of [3]. The following figures show at which architectural component of the overall N-RSA the different middleware components need to be implemented, based on the functionality and interfaces described in this report.



**Figure 2** Middleware Components at Application Server & Intermediary Gateway



**Figure 3** Middleware Components at Repository Server

The following sections will give some more details on which functionality needs to be provided by each of the components, in addition to implementing the appropriate interfaces as defined in Section 3.

**2.1 Event Communication Paradigm in N-RSA**

Before we introduce the functionality of each component of the N-RSA middleware, some words on the underlying communication paradigm in N-RSA.

Nokia Research Center / Helsinki  
Dirk Trossen, Dana Pavel

FINAL  
24.10.06

Many interfaces within the N-RSA are based on a common event delivery paradigm with the following characteristics:

- The event delivery is based on individual *subscriptions* to so-called *events* (implementing requirement 8.a in [3]). These events usually represent state information associated to certain resources.
- In order to allow for proper identification of the particular subscription, allowing for several subscriptions at any given time, there is a *dialog identifier* provided with each confirmation of the subscription.
- The semantic of each event is application-specific and based on a certain semantic description.
- The event delivery allows for arbitrary payload for each event subscription and notification (implementing requirement 8.a in [3])
- For each subscription, there exists a subscriber (client) and an event server
- There exists five methods (messages)
  - SUBSCRIBE allows for initiating a subscription. It contains
    - *event name*,
    - *event-specific content* (such as query or filter descriptions)
    - *lifetime* of the subscription. If the lifetime equals zero, the subscription serves as a “one-shot” subscription or poll of information, i.e., there will be only a single notification and there is no subscription dialog established. Otherwise, the lifetime parameter specifies the duration of the subscription in seconds.
    - *FROM* and *TO* parameters, identifying the subscriber and event server. These identifiers depend on the particular bearer of information (such as SMS, IP, SIP events, HTTP URIs) on which the event communication paradigm will be implemented.
  - PUBLISH allows for publishing relevant information for a particular event, such as state information or information used in the state determination by the event server. The sender of the PUBLISH method is usually not the subscriber to the event (but could be), the receiver is the event server that hosts the event. The method contains
    - the *event name* to which the information relates

- *event-specific content*, i.e., the information relating to the event.
- *FROM* and *TO* parameters, identifying the publisher and event server. These identifiers depend on the particular bearer of information (such as SMS, IP, SIP events, HTTP URIs) on which the event communication paradigm will be implemented.
- CONFIRM always confirms each publication to the publisher and each subscription to the subscriber. The confirmation can either be positive (subscription granted) or negative. In the latter case, a reason code is provided. In the positive case for a subscription, the *dialog identifier* is provided to the subscriber to identify the subscription dialog. For a publication, there is no dialog established.
- NOTIFY allows for sending notifications for a particular subscription. These notifications are triggered depending on the particular event semantic. Examples are state changes in the event or other reasons. There is always an initial NOTIFY, conveying the initial state of the subscription right after the subscription has been granted (note that this initial NOTIFY is not sent if the submission has not been granted, indicated through a negative CONFIRM). If the lifetime of the subscription is non-zero, future NOTIFY messages are possible, depending on the semantic of the event subscription. The NOTIFY method includes *FROM*, *TO*, and *event name* fields from the original subscription. It further includes the *dialog identifier* to properly relate notification and subscription.
- BYE explicitly terminates a subscription. The dialog identifier needs to be given in order to relate the BYE method to the particular subscription. A BYE method can be sent from either the subscriber or the event server. In the latter case, a reason code is provided.

The above outlined event delivery paradigm is very similar to the SIP event framework [1]. Interfaces are defined throughout the N-RSA some of which are based on the outlined event communication paradigm. These interfaces implement the functionality as described in this section. Only additional, interface-specific functionality will be described for these interfaces. We will refer appropriately to the current section in such cases.

## 2.2 Event Delivery Component

The event delivery component implements the requirement 8 (and its sub-requirements) as defined in [3]. In other words, this component implements the event communication paradigm of Section 2.1 in order to provide a general purpose component for all remote communication between application server, repository server and intermediary gateway.

Nokia Research Center / Helsinki  
Dirk Trossen, Dana PavelFINAL  
24.10.06

At the application server and intermediary gateway, there exist subscriber/publisher as well as event server functionality, while the code repository merely acts as a publisher of information.

The functionality is provided through interface  $E_D$  (see Section 3.2 for remote communication between event delivery components) and interface  $E_{D(local)}$  (see Section 3.7 for local communication from other components towards the local event delivery component). With respect to the actual functionality of this component, see also Section 2.1.

Incoming requests, such as SUBSCRIBE at the event server side or NOTIFY at the subscriber side, are dispatched to the appropriate local component, based on the event information given in the request (see interface descriptions in Section 3 for the event information for each middleware component). In this dispatching, the particular middleware component is also provided with the information included in the request, i.e., event name, event body, dialog identifier, and FROM/TO information.

In order to implement requirement 8.b and 8.c in [3], the event delivery component implements bearer-specific functionality to provide the interface  $E_D$  to upper layer components.

### 2.3 Acquisition Component

The acquisition component implements requirement 7.c (and its sub-requirements) as defined in [3].

On the application server side, the acquisition component is one of the two components to be accessed by the application(s). This access might happen locally (i.e., application resides locally on the application server) or remotely (e.g., through providing CORBA-enabled access to the component) and is defined through Interface  $RAP_{AA}$  (see Section 3.1).

Acquisition requests are pre-processed at the application server by

- checking the access rights of the particular application with respect to the requested data (interface  $A_{ctrl(local)}$  as defined in Section 3.9) and
- verifying the availability of particular sensors that are required to fulfill the request (interface  $A_{RA(local)}$  as defined in Section 3.8)
- verifying the availability of code for particular aggregation functionality, if required in the request, (interface  $A_{RA(local)}$  as defined in Section 3.8)

Hence, in this case the application acts as a subscriber while the application server implements the event server, the event being the acquisition request.

If these verifications return positively, the acquisition request is made remotely at the appropriate intermediary gateway through the interface  $A_C$  as defined in Section 3.4. In this

Nokia Research Center / Helsinki  
Dirk Trossen, Dana Pavel

FINAL  
24.10.06

case the application server acts as a subscriber for an event (being the acquisition request) at the intermediary gateway, the gateway's acquisition component acting as a event server.

Acquisition requests at the intermediary gateway are received remotely (through interface  $A_C$ ). The query of the request is forwarded to the query resolver component (see Section 2.4) through interface  $A_{Q(local)}$  as defined in Section 3.11. The query resolver returns pointers to sensor objects and (if required) aggregation objects required to satisfy the query.

Based on the return results of the query resolver component, the required sensor data is acquired via interface  $SS|_A$  (see Section 3.5), using the returned sensor objects.

If required, obtained sensor data is aggregated according to the query (implemented through communicating with the aggregation component via interface  $A_{A(local)}$  as defined in Section 3.10). If the request is fulfilled, appropriate notifications are fired via interface  $A_C$ .

Note that several acquisition requests may be pending on either the application server but also on the intermediary gateway. Hence, there is functionality required to support such multiple requests.

## 2.4 Query Resolver (formerly known as Representation) Component

Incoming acquisition requests contain queries for sensor data (or aggregations of sensor data). These queries are based on a defined abstraction model for the sensor data (and its aggregations), implementing requirement 7b (and its sub-requirements) in [3]. Such abstraction model will be defined in task 2.2a of the N-RSA project.

The query resolver component (formerly known as the representation component in [3]) implements functionality to

- parse the incoming request, based on the abstraction model and the query language syntax
- determine the sensor data to be acquired by the acquisition component
- determine the aggregation functionality that is required in case when aggregated data is requested.

While the first bullet items merely aims at the validity of the request, the last two ones aim at instructing the acquisition component as to what sensor data to acquire and what aggregation component functionality to use in order to fulfill the request.

For that, the query is received and the results are returned via the interface  $A_{Q(local)}$  (as defined in Section 3.11).

Nokia Research Center / Helsinki  
Dirk Trossen, Dana Pavel

FINAL  
24.10.06

## 2.5 Aggregation Component

The aggregation component implements requirement 7.d in [3] by providing the functionality for the acquisition component to support aggregated data.

Based on the acquisition query, the query resolver determines the required aggregation functionality (see Section 2.4). The acquisition component then requests the determined aggregation functionality from the aggregation component (via interface  $A_{A(local)}$  as defined in Section 3.10). If the required functionality does not exist, the aggregation component attempts to download the code for implementing the requested functionality from the code repository component (via interface  $C_{A(local)}$  as defined in Section 3.15), see also Section 2.8. In case this functionality of code download is optional or an error occurred in the code download, the acquisition is rejected.

During the acquisition process, the acquisition component provides the aggregation component with the appropriate sensor data to perform the desired aggregation functionality (via interface  $A_{A(local)}$  as defined in Section 3.10). If obtained sensor data is required for future use in order to perform the aggregation, it can be stored with the local storage component (via interface  $A_{S(local)}$  as defined in Section 3.12), implementing requirement 7.f in [3].

## 2.6 Access Control Component

The access control component implements requirement 7.e in [3]. This component only resides on the application server side. As explained in Section 2.3, the access control component is consulted in case of an incoming acquisition request in order to determine appropriate access rights for the particular application with respect to the particular data requested (using interface  $A_{ctrl(local)}$  as defined in Section 3.9).

Although depicted as a local component in Figure 1, it is possible to implement the access control component in an outside server. This server might host other access control policies as well. For instance, the additional server might be compliant to XCAP [3], i.e., the interface  $A_{ctrl(local)}$  as defined in Section 3.9 would be compliant to XCAP, the policies would be defined as *XCAP usages*.

## 2.7 Storage Component

The storage component implements requirements 7.f in [3], i.e., the temporary storage of acquired data, e.g., for implementing certain aggregation functionality.

This storage functionality uses simple memory heap functionality in the intermediary gateway, assuming no existing hard-drive functionality. However, one could also envision the usage of externally available memory cards (e.g., MMC).

Nokia Research Center / Helsinki  
Dirk Trossen, Dana PavelFINAL  
24.10.06

## 2.8 Code Repository Component

The code repository component implements requirement 9.b (and also requirement 9.a if therein new sensor types are used in aggregation functionality) in [3]. The code repository component resides on the repository server as well as on the intermediary gateway.

The code repository component at the repository server registers every available aggregation functionality with the local registration & availability component (using interface  $C_{RA(local)}$  as defined in Section 3.16), which in turn registers the functionality with the application server (see Section 2.9) in order to allow for verifying the required aggregation functionality for each acquisition request at the application server (see Section 2.3).

At the intermediary gateway, the aggregation module may request new aggregation code functionality from the code repository component (as explained in Section 2.5) via interface  $C_{A(local)}$  as defined in Section 3.15. The code repository component, as a result of such request, downloads the requested aggregation functionality from the repository server to the intermediary gateway for use in aggregation operations (using interface  $C_R$  as defined in Section 3.13).

## 2.9 Registration & Availability Component

The registration & availability (R&A) component implements the requirement 6 (and its sub-requirements) in [3].

The application server's R&A component serves as a central registry for this purpose, implementing an event server, used by the local acquisition module. The following information is published to the R&A component at the application server:

- Available sensors at the intermediary gateway. For this, the R&A component at the intermediary gateway subscribes to the sensor capabilities using interface  $SSI_R$  as defined in Section 3.6. Obtained sensor information is then published at the application server's R&A component using interface  $R_A$  as defined in Section 3.3.
- Available aggregation functionality at the repository server. For this, the R&A component at the repository server registers all available functionality with the application server's R&A component (using interface  $R_{A(IAS-RS)}$  as defined in Section 3.14) after receiving the registrations of aggregation functionality from the local code repository component (see Section 2.8).

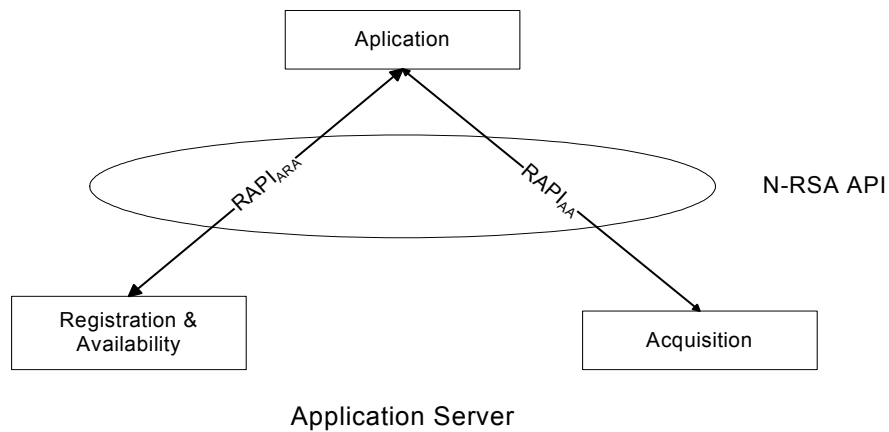
The available information at the R&A component at the application server can either be queried by the application directly, using interface  $RAPI_{ARA}$  as defined in Section 3.1, or is queried by the acquisition component, using interface  $A_{RA(local)}$  as defined in Section 3.8, before sending an acquisition request to the intermediary gateway.

It is important to note that the R&A component also provides functionality to subscribe to the availability of sensors (or aggregation functionality), implementing requirement 6.c in [3].

**3. INTERFACES BETWEEN THE COMPONENTS**

The following chapter describes the interfaces of the N-RSA middleware. The interfaces implement the requirements as defined in [3].

**3.1 The RAPI Interface (N-RSA API)**



**Figure 4** The RAPI Interface

The RAPI interface constitutes the N-RSA API in Figure 1. It is divided into two parts, namely the RAPI<sub>ARA</sub> and the RAPI<sub>AA</sub> interfaces.

**3.1.1 The RAPI<sub>ARA</sub> interface**

The RAPI<sub>ARA</sub> interface allows for subscribing to the availability of sensors or aggregation functionality within the N-RSA system. This interface is based on the event delivery paradigm outlined in Section 2.1:

- The application is the subscriber, the R&A component is the event server
- The event name is 'available', the event payload of the SUBSCRIBE method carries the description for the sensor or aggregated sensor
- If the lifetime is zero, the subscription constitutes a normal discovery of available sensors, while a lifetime of non-zero constitutes a subscription to current and future availability of sensors.

Nokia Research Center / Helsinki  
Dirk Trossen, Dana PavelFINAL  
24.10.06

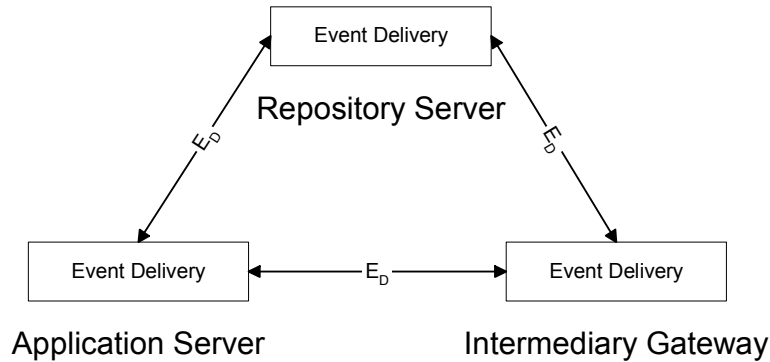
- The FROM field contains the application identifier, while the TO field contains the application server identifier.
- The NOTIFY methods returns the matching sensors, based on the provided semantic of the subscription semantic. The initial NOTIFY contains the currently available sensors, while future NOTIFY methods will contain then available sensors. Every plain sensor description additionally includes an identifier of the intermediary gateway at which the sensor is registered.

### 3.1.2 The RAPI<sub>AA</sub> interface

The RAPI<sub>AA</sub> interface allows for instructing the acquisition component to acquire (aggregated or plain) sensor data from a particular intermediary gateway. Note that the pre-requisite functionality, as described in Section 2.3, is executed before the acquisition subscription is made. This interface is based on the event delivery paradigm outlined in Section 2.1:

- The application is the subscriber, the acquisition component is the event server
- The event name is '*acquire*', the event payload of the SUBSCRIBE method carries the query for the (aggregated or plain) sensor data
- If the lifetime is zero, the subscription constitutes a simple query for particular (aggregated or plain) sensor values, while a lifetime of non-zero constitutes a subscription to current and future values.
- If the application wants to acquire sensor data from a particular intermediary gateway, the identifier of this gateway needs to be given in the event payload. This identifier depends on the particular bearer of information supported by the event delivery component.
- The FROM field contains the application identifier, while the TO field contains the application server identifier.
- If the acquisition request is invalid, e.g., due to lack of access rights, inavailability of aggregation functionality or other, the CONFIRM method will return an appropriate error reason code.
- The NOTIFY methods returns the current values based on the original query. The body of the NOTIFY method also includes an identifier of the intermediary gateway at which the values were acquired.

**3.2 The E<sub>D</sub> Interface**



**Figure 5** The E<sub>D</sub> Interface

The E<sub>D</sub> interface provides the event delivery functionality as described in Section 2.1 for the remote case, i.e., for communicating between application server, repository server, and intermediary gateway. The parameters described in Section 2.1 are therefore provided by this interface.

All communication between remote components happens via the E<sub>D</sub> interface, together with using the generic E<sub>D(local)</sub> template interface. In other words, any particular interface X is implemented by conveying the parameters to the event delivery component through the E<sub>D(local)</sub> template interface. The event delivery component in turn provides the delivery of the parameters to the appropriate component according to the definition of interface X, either locally through the E<sub>D(local)</sub> interface or remotely through the E<sub>D</sub> interface plus a local dispatching via the E<sub>D(local)</sub> interface.

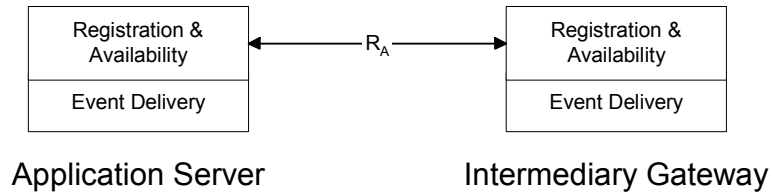
Since the E<sub>D</sub> interface is the basic remote interface, its establishment has to take into account the particular problem of NAT/firewall traversal in mobile environments. In other words, the interface needs to be established from the intermediary gateway towards the application server and towards the repository server. The communication protocol, defined in task 2.2b, will ensure this.

The event delivery component provides appropriate mappings of the interface onto different bearers of information. For SMS, HTTP, and pure IP bearers (requirement 8.b.I and 8.b.II.1 in [3]), this mapping needs to be defined within task 2.2b. For SIP events (according to requirement 8.b.II.2), this mapping happens onto appropriate SIP event packages.

Nokia Research Center / Helsinki  
Dirk Trossen, Dana Pavel

FINAL  
24.10.06

**3.3 The R<sub>A</sub> Interface**

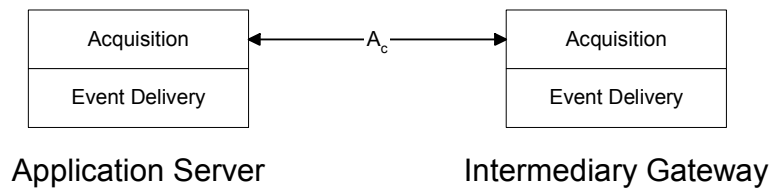


**Figure 6** The R<sub>A</sub> Interface

The R<sub>A</sub> interface allows for publishing the availability of sensors with the N-RSA system. This interface is based on the event delivery paradigm outlined in Section 2.1:

- The R&A component at the intermediary gateway is the publisher, the R&A component at the application server is the event server
- The event name is 'available', the event payload of the PUBLISH method carries the description for the sensor(s)
- It is possible to publish information about several sensors within a single operation by appropriately building descriptions for each sensor, carried as a single body in the publication
- The FROM field contains the identifier of the intermediary gateway, the TO field contains the application server identifier

**3.4 The A<sub>C</sub> Interface**

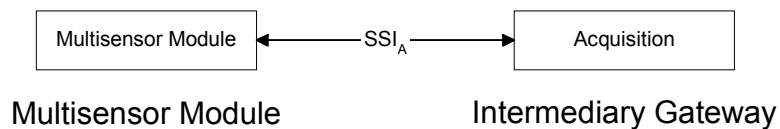


**Figure 7** The A<sub>C</sub> Interface

The A<sub>C</sub> interface allows for acquiring (aggregated or plain) sensor data from a particular intermediary gateway. Note that the pre-requisite functionality, as described in Section 2.3, is executed before the acquisition subscription is made. This interface is based on the event delivery paradigm outlined in Section 2.1:

- The acquisition component at the application server is the subscriber, the acquisition component at the intermediary gateway is the event server
- The event name is 'acquire', the event payload of the SUBSCRIBE method carries the query for the (aggregated or plain) sensor data
- If the lifetime is zero, the subscription constitutes a simple query for particular (aggregated or plain) sensor values, while a lifetime of non-zero constitutes a subscription to current and future values.
- The FROM field contains the application server identifier, the TO field contains the identifier of the intermediary gateway
- If the acquisition request cannot be fulfilled, e.g., due to lack of resources, the CONFIRM method will return an appropriate error reason code.
- The NOTIFY methods return the current values based on the original subscription.

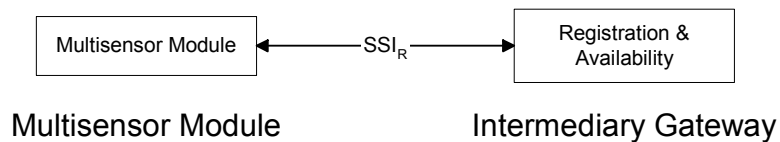
**3.5 The SSI<sub>A</sub> Interface**



**Figure 8** The SSI<sub>A</sub> Interface

The SSI<sub>A</sub> interface allows for acquiring sensor data from a particular multisensor module. The API for this interface is the Sensor API in Figure 1, the protocol chosen is the SSI protocol [2].

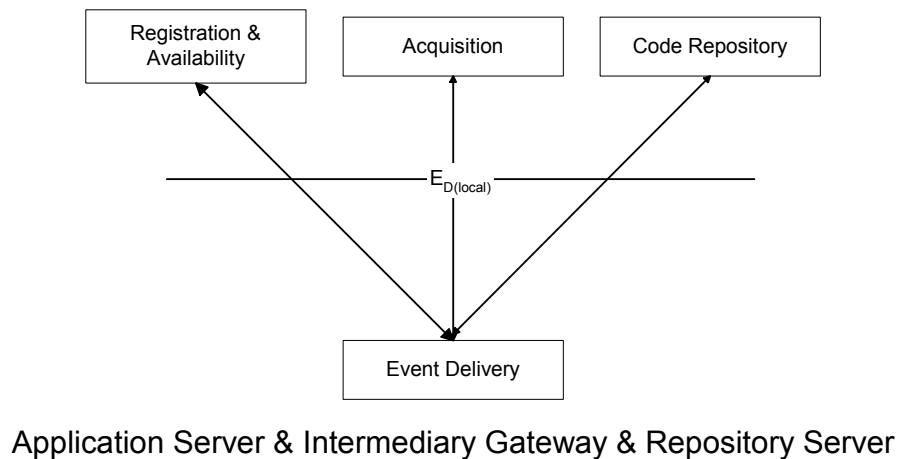
**3.6 The SSI<sub>R</sub> Interface**



**Figure 9** The SSI<sub>R</sub> Interface

The  $SSI_R$  interface allows for subscribing and querying the capabilities of a multisensor module. The API for this interface is the Sensor API in Figure 1, the protocol chosen is the SSI protocol [2].

**3.7 The  $E_{D(local)}$  Interface**

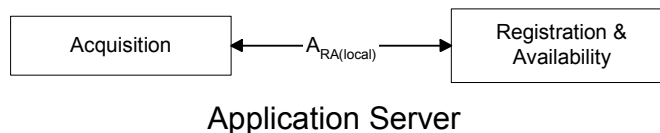


**Figure 10** The  $E_{D(local)}$  Interface

The  $E_{D(local)}$  interface serves as a “template interface”, providing locally the event delivery functionality as described in Section 2.1. The parameters described in Section 2.1 are therefore provided by this interface.

For any communication between components that is based on the event delivery paradigm as described in Section 2.1, a particular interface X is implemented through conveying the parameters to the event delivery component via the  $E_{D(local)}$  template interface. The event delivery component in turn provides the delivery of the parameters to the appropriate component according to the definition of interface X, either locally through the  $E_{D(local)}$  interface or remotely through the  $E_D$  interface plus a local dispatching via the  $E_{D(local)}$  interface.

**3.8 The  $A_{RA(local)}$  Interface**

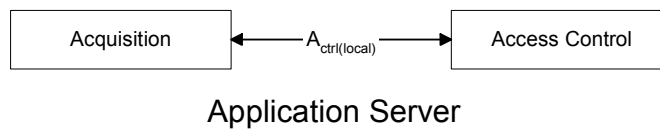


**Figure 11** The  $A_{RA(local)}$  Interface

The  $A_{RA(local)}$  interface allows for querying the availability of certain sensor and aggregation functionality.

For that, the sensor and aggregation functionality description is provided to the R&A component, which returns Boolean values as results. Several sensors and aggregation functionalities can be checked, resulting in a Boolean result vector.

**3.9 The  $A_{ctrl(local)}$  Interface**



**Figure 12** The  $A_{ctrl(local)}$  Interface

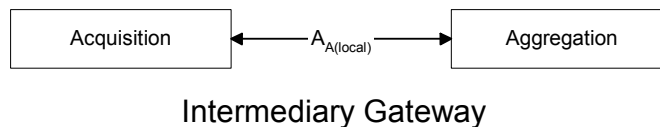
The  $A_{ctrl(local)}$  interface allows for querying whether or not a certain identity has proper access rights for particular sensor and aggregation functionality.

For that, the identity is given as a parameter together with a pre-defined identifier for the resource, i.e., the sensor or aggregation functionality.

The query is returned with a Boolean result.

Although depicted as a local interface, it is possible that  $A_{ctrl(local)}$  is implemented as a remote interface. This is for cases in which the access control component resides on an outside server (see also Section 2.6). This server might host other access control policies as well. For instance, the additional server might be compliant to XCAP[4], i.e., the interface  $A_{ctrl(local)}$  would then be compliant to XCAP, the policies would be defined as *XCAP usages*.

**3.10 The  $A_A(local)$  Interface**

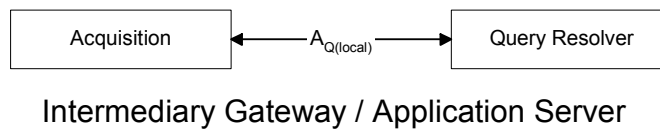


**Figure 13** The  $A_A(local)$  Interface

The  $A_A(local)$  interface allows for initiating the aggregation of particular sensor data.

For that, the sensor objects and the particular aggregation object, obtained at the acquisition component during the query resolving, are used as input parameters. The aggregation is fed into the aggregation object and returned.

**3.11 The  $A_{Q(local)}$  Interface**



**Figure 14** The  $A_{Q(local)}$  Interface

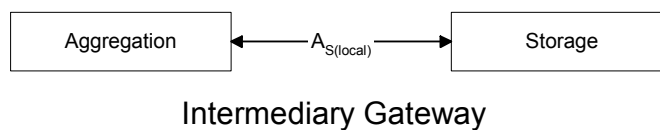
The  $A_{Q(local)}$  interface allows for initiating the resolving of the acquisition query at the intermediary gateway as well as at the application server (see also Section 2.3 and 2.4).

For that, the acquisition query is provided as an input parameter.

At the application server, the query resolver returns identifiers for the required sensors and aggregation functionalities to satisfy the query, or the resolver returns an error code. These identifiers will be used to verify access rights and availability as described in Section 2.3.

At the intermediary gateway, the query resolver returns pointers to objects for the required sensors and aggregation functionalities to satisfy the query, or the resolver returns an error code.

**3.12 The  $A_{S(local)}$  Interface**



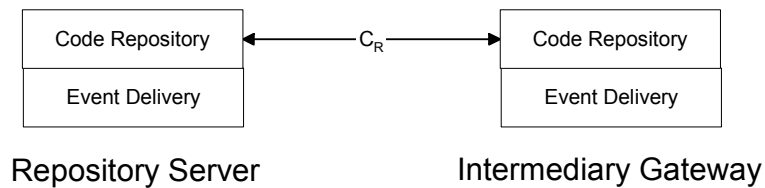
**Figure 15** The  $A_{S(local)}$  Interface

The  $A_{Q(local)}$  interface allows for storing and retrieving sensor data for future use in order to perform the desired aggregation (see also Section 2.5).

For that, the sensor data is written to the storage component using the current value and a sensor identifier. The storage component returns an identifier for future retrieval.

Upon future retrieval, this identifier is used to retrieve the data. An additional Boolean value determines whether or not to delete the stored value from the storage. The sensor value and sensor identifier are returned to the aggregation component.

**3.13 The C<sub>R</sub> Interface (optional)**

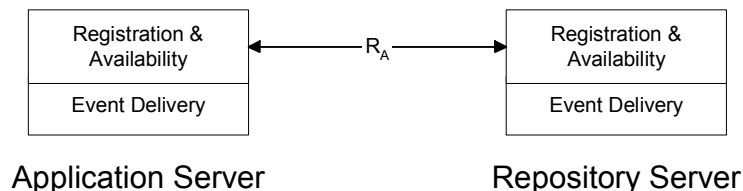


**Figure 16** The C<sub>R</sub> Interface

The C<sub>R</sub> interface allows for downloading required aggregation functionality to the intermediary gateway from the repository server. This interface is based on the event delivery paradigm outlined in Section 2.1:

- The code repository component at the intermediary gateway is the subscriber, the code repository component at the repository server is the event server
- The event name is 'download', the event payload of the SUBSCRIBE method carries the information about the aggregation functionality to be downloaded
- The lifetime is set to zero, making the subscription a simple query for download.
- The FROM field contains the identifier of the intermediary gateway, the TO field contains the repository server identifier
- The (only) NOTIFY methods returns the desired code in case of availability, otherwise the CONFIRM method will return an appropriate error reason

**3.14 The R<sub>A(AS-RS)</sub> Interface (optional)**

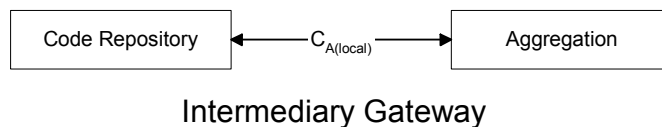


**Figure 17** The R<sub>A(AS-RS)</sub> Interface

The  $R_{A(AS-RS)}$  interface allows for publishing available aggregation functionality with the N-RSA system. This interface is based on the event delivery paradigm outlined in Section 2.1 and is similar to the  $R_A$  interface:

- The R&A component at the repository server is the publisher, the R&A component at the application server is the event server
- The event name is 'available', the event payload of the PUBLISH method carries the description for aggregation functionality
- It is possible to register several aggregation modules within a single operation by appropriately building descriptions for each functionality, carried as a single body in the publication

**3.15 The  $C_{A(local)}$  Interface (optional)**



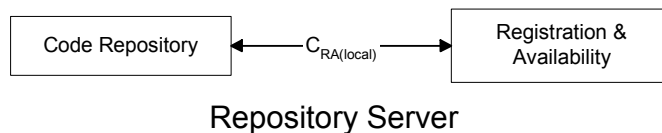
**Figure 18** The  $C_{A(local)}$  Interface

The  $C_{A(local)}$  interface allows for initiating an aggregation code download from the repository server.

For that, the required aggregation code is given as an input parameter. The aggregation code is identified through a given description of the to-be-performed aggregation.

The pointer to the aggregation code object for the requested aggregation is returned to the aggregation component.

**3.16 The  $C_{RA(local)}$  Interface (optional)**



**Figure 19** The  $C_{RA(local)}$  Interface

Nokia Research Center / Helsinki  
Dirk Trossen, Dana PavelFINAL  
24.10.06

The  $C_{RA(local)}$  interface enables the code repository at the repository server to publish available aggregation functionality locally.

This publishing usually happens upon start-up of the repository server or when new aggregation functionality has been made available to the code repository component locally.

The code repository component provides a description of the aggregation functionality to the local R&A component.

It is possible to register several aggregation modules within a single operation by appropriately building descriptions for each functionality, carried in a single operation.

#### 4. REFERENCES

- [1] A. Roach, "SIP-Specific Event Notification", RFC 3265, July 2002
- [2] J. Hyyryläinen, I. Jantunen "SSI Protocol Specification V1.0", April 2005, available at [http://ssi-protocol.net/SSI%20protocol%20specification\\_10a-2.pdf](http://ssi-protocol.net/SSI%20protocol%20specification_10a-2.pdf)
- [3] D. Trossen, D. Pavel, "N-RSA High-Level System Architecture", N-RSA 2004 project report, January 2004
- [4] J. Rosenberg, "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)", Internet Draft, Internet Engineering Task Force, (work in progress), May 2003