NPW3C2011-008

Design Examples
- For Analyzer Developer –
(Revision 1.0)

NEC Corporation

1. Introduction

1.1 Objective

ORIENT is an interface (I/F) for sending and receiving data between ARMOR and analysis engine, ARMOR and user application. The standardization of I/F defined in ORIENT is in progress. The interoperability of data, which has mostly been exchanged between analysis engine/ARMOR/user applications in the manufacturing of individual system till now, will increase by using this standard I/F. Hence, it will become easy to use analysis engine in many systems by making analysis engine correspond to ORIENT.

This document aims to provide the guidelines for user application to make analysis engine correspond to ORIENT. The combination of analysis engine and ARMOR is called as metadata integration platform. To make analysis engine correspond to ORIENT, it is necessary to think about a method by which metadata integration platform makes data public to user application and a method by which analysis engine and ARMOR communicate inside metadata integration platform. Here, these correspondence methods are called as metadata integration platform correspondence methods. There are various methods to make analysis engine correspond to metadata integration platform. It is desired to know advantages and disadvantages of each method in advance. Further, concrete methods to realize specific metadata integration platform correspondence method are shown with the help of specific examples of processing flow and configuration for each metadata integration platform correspondence method.

1.2 Envisioned readers

The intended readers of this document are the developers of analysis engine, who make analysis engine, correspond to metadata integration platform. As a prerequisite, it is assumed that the overview of ORIENT, ARMOR and metadata integration platform has already been understood through related documents.

1.3 Scope of study

The objective of this document is to explain the guidelines of correspondence method of analysis engine with metadata integration platform through selection guidelines and concrete examples of correspondence method of analysis engine and metadata integration platform. Therefore, detailed explanation is not provided in this document as it is assumed that the overview of ORIENT, ARMOR and metadata integration platform has already been understood. The guidelines of applications, which are required in view of configuration of system, are provided in a separate document.

As regards the selection guidelines and concrete examples, only guidelines for deciding metadata integration platform correspondence method and necessary information required for deciding system configuration are provided in this document instead of the detailed description.

1.4 Document composition

This document comprises Section 2 and Section 3. In Section 2, metadata integration platform correspondence method is classified along with its advantages and disadvantages. In the metadata integration platform

correspondence method, two types of classification are provided, namely classification of ORIENT usage method, and implementation method classification for ORIENT usage between ARMOR and analysis engine.
In Section 3, the details of process flow etc. of correspondence method of each metadata integration platform are explained and can be referred to at the time of applying metadata integration platform correspondence method.

1.5 Definition of terminologies

The definitions of terminologies are listed below. The terminologies shown below are used without explanation in the document.

Table 1 Definition of terminologies

Terminology	Definition
RDF	It is the abbreviation of Resource Description Framework. It is a framework, which is used at the time of describing information about resources over web.
RDF/XML	Usage of XML as a serialization rule of RDF.
Application	A system, which realizes real world monitoring service by using metadata integration platform comprising ARMOR and analysis engine
Engine	A system, which outputs analysis result such as location of person etc. by analyzing raw data of sensor, media and text system.
ARMOR	An implementation of analysis data manager, which connects application and engine as per ORIENT specification protocol.
ORIENT	It is an interface (I/F) for sending and receiving data between ARMOR and analysis engine/user application. ORIENT stands for “Operation / Restriction / Entities” based framework.
Service	The service described in this document consists of application, analysis system and ARMOR.
Real world monitoring service	It is the general term of the service, which is the target of this development using ARMOR.
RDB	It is the abbreviation of relational database indicating a database based on relational model. It is assumed that it is possible to access data by SQL.
Metadata integration platform	A system in which ARMOR and analysis engine, are combined

1.6 Related documents

[1] ORIENT Software Development Kit User's Manual - For Application Developer - (NPW3C2011-003)

[2] ORIENT Software Development Kit User's Manual - For Analyzer Developer - (NPW3C2011-004)

[3] Analysis Data Transfer Framework "ORIENT" Specification (NPW3C2011-002)

2. Methods for applying analysis engines to the metadata integration platform

In order for application programs to use the standard I/F "ORIENT", we provide some types of methods for applying analysis engines to the standard I/F "ORIENT". With this re-usage of application becomes easy. Here, making analysis engine correspond to metadata integration platform is called metadata integration platform correspondence. There are multiple methods to carry out correspondence of this metadata integration platform and it is advised to understand the advantages and disadvantages in designing the architecture.

In this section, an overview of several methods of metadata integration platform correspondence is described and the guidelines, mentioning which correspondence method is good to use, are shown. First, it shows that analysis engine exchanges analysis data with ARMOR by ORIENT operation either write or query operation as shown below. Next, it shows the selection options of implementation method at the time of communicating with ARMOR. Further, in the end of this section, selection guidelines of metadata integration platform correspondence method mentioned in other sections are summarized.

2.1 Types of ORIENT usage methods

The standard I/F "ORIENT" is used for the external/internal interfaces of ARMOR. ARMOR receives analysis data generated by analysis engines and sends the result data to the application programs through "ORIENT". Methods to exchange analysis result data between analysis engines and ARMOR can be classified into the following two types: Write type, which use Write operation of ORIENT, and Query response type, which corresponds to Query operation of ORIENT. This section describes an overview of respective ORIENT usage methods along with their advantages and disadvantages. Fig. 1 provides an overview of two types of methods for applying analysis engines to the standard I/F "ORIENT.

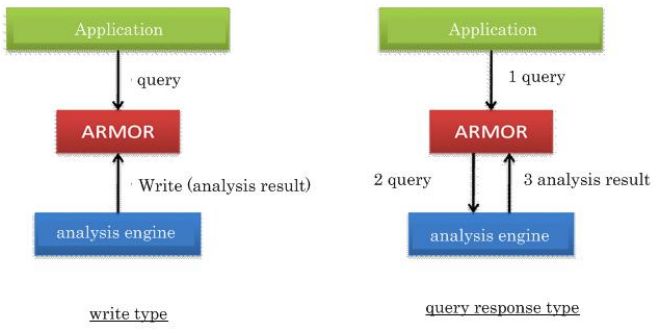


Figure 1 Types of ORIENT usage methods

(1) Write type

As regards the method to make analysis data public, it is a method which writes analysis data in ARMOR through write operation of ORIENT.

(1-1) Behaviour

Analysis engine registers analysis data to ARMOR through write operation when analysis data is obtained by performing analysis processing. The method to make analysis data of application public can be considered in two ways. The first method is the method in which application obtains data, which is accumulated in ARMOR, through query. The second method is the method in which ARMOR notifies the application through notify operation. In the former method, when the application issues data acquisition request to ARMOR through query, ARMOR selects stored analysis data by query conditions and returns it. The write operation and query operation are carried out for asynchronously. In the latter method, when analysis engine hands over analysis result to ARMOR by write operation, ARMOR notifies it to the application through notify operation.

This document explains the description of application obtaining the data through query, and does not discuss the details of method to notify from ARMOR.

(1-2) Advantages and disadvantages

The advantages of write type include reduction in implementation and processing cost of analysis engine. It is because ARMOR responds to the query and the implementation for responding to the query becomes unnecessary at analysis engine side. Therefore, write type is recommended if the implementation cost is to be reduced.

However, the possibility of increase in processing load of analysis engine and ARMOR is cited as the disadvantage. In write type, analysis data, which has a possibility of being used by the application, is processed in advance and required to store in ARMOR. Therefore, analysis processing is carried out even for the analysis result, which is not actually used by the application and data is stored in ARMOR. As a result, the load on analysis processing increases and the data stored in ARMOR is possible to increase.

(2) Query response type

As regards the method to make analysis data of query response type public, it corresponds to query operation of ORIENT and provides analysis data to the application through ARMOR.

(2-1) Behaviour

The right part of Fig.1 shows the query response type. In the query response type, if application issues a query to ARMOR (shown as 1 in Figure), then ARMOR forwards it to analysis engine (shown as 2 in Figure). In the analysis engine, analysis result, stored previously in it, and analysis result in response to query are created and returned (shown as 3 in Figure). These processing are carried out in sequence.

(2-2) Advantages and disadvantages

The advantages of query response type include the possibility of reduction in the load of analysis engine and the reduction in the volume of data, maintained in ARMOR. It is because analysis processing is carried out only when there is a request from application and analysis result is not stored in ARMOR in query response type. However, high processing cost at the time of query in case analysis is carried out as a

response to a query and high implementation cost are cited as disadvantages. The reason behind the high implementation cost is the necessity of implementing search processing at analysis engine side. Further, increase in the processing cost at the time of query is caused owing to the configuration wherein analysis processing is carried out to respond to a query. In case multiple queries are received simultaneously, processing load will increase and cause problem.

2.2 Implementation method of ORIENT usage

In this section, the configuration required for analysis engine to communicate with ARMOR (Implementation method of ORIENT usage) is explained.

The implementation method of ORIENT usage can be broadly classified into (1) communication library usage type (2) external GW (Gateway) type and (3) XML direct operation type". Further, an overview of implementation method of each ORIENT usage is explained with advantages and disadvantages.

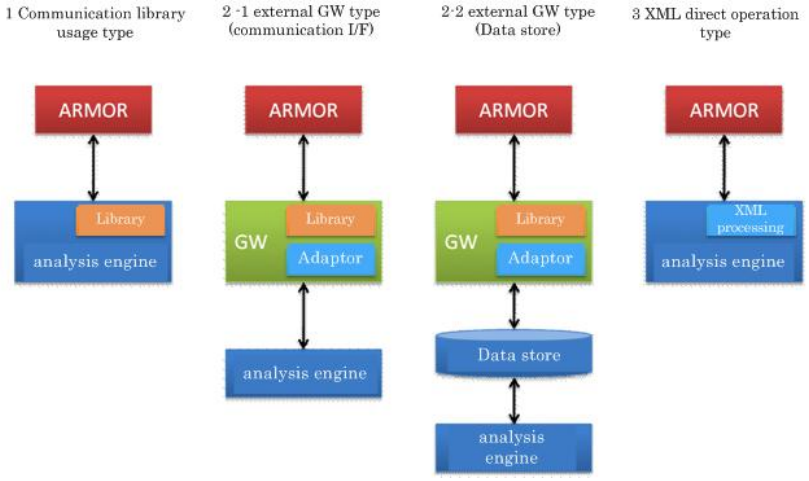


Figure 2 ORIENT usage implementation method

Figure 2 ORIENT usage implementation method shows the configuration for implementation method of each ORIENT usage.

(1) Communication library usage type

As regards the communication library usage type, it is a method in which ORIENT communication library is embedded in analysis engine and direct communication is carried out with ARMOR. Analysis engine carries out processing through ORIENT communication library at the time of responding to write operation and query for ARMOR. The implementation is easy as ORIENT communication library hides the processing details.

This communication library usage method is the most recommended implementation method of ORIENT usage.

Advantages and disadvantages

The advantages of implementation method of ORIENT usage through communication library usage type include easy implementation, easy to maintain interoperability and small communication overhead for direct communication. It is because ORIENT communication library provides API suitable for ORIENT operation and high connectivity is assured using same ORIENT communication library by both ARMOR and analysis engine. However, the difficulty of embedding it in analysis engine, created in language other than Java, as the ORIENT communication library is provided as Java library is cited as disadvantage.

(2) External GW type

External GW type is an ORIENT usage implementation method in which analysis engine accesses ARMOR via gateway (GW). Gateway mediates the communication between ARMOR and analysis engine. In communication with ARMOR, ORIENT communication library is and the communication data with ARMOR is converted into a format, which can be used by analysis engine and data store via an adapter. This method is used in cases analysis engine is coded in a language other than Java and embedding of ORIENT communication library is difficult or in case analysis engine is already configured to save the data in data store. In external GW type, two types of configuration can be considered. First is communication I/F usage, and the second is data store usage. The respective configuration will be explained after describing the common advantages and disadvantages of external GW type.

Advantages and disadvantages

The advantages of ORIENT usage implementation method by external GW type include the ease to use analysis engine, written in language other than Java and the non requirement of modification in analysis engine. It is because the gateway program can be created as an independent program. However, occurrence of communication overhead and complexity of configuration are cited as disadvantages. This is due to an increase in the program or process, which exists in between.

(2-1) External GW type (communication I/F usage)

External GW type (communication I/F usage) is the configuration in which gateway directly communicates with analysis engine. Therefore, the role of the adapter is to convert it from ORIENT data type to a type which can be communicated with analysis engine (reverse conversion).

Advantages and disadvantages

The advantages of external GW type (communication I/F usage) include reduction of communication compared to external GW type (data store usage) as the component such as data store is reduced. Further, it is possible to perform write and query response operation as long as analysis engine is working.

However, the disadvantage is that it cannot be used if analysis engine does not support each operation.

(2-2) External GW type (data store usage)

External GW type (data store usage) is the configuration in which analysis engine writes analysis result in a data store, similar to relational database, and the gateway, which responds to ARMOR query, extracts analysis result from that data store. External GW type is used when analysis engine is configured to write analysis result in data store.

Advantages and disadvantages

The advantage of external GW type (data store usage) is that the architecture is easy through the configuration of analysis engine. Further, it is easy to store data as handing over of data via data store is linked to data storage.

However, the disadvantage is that it finds difficult to support write operation. The cost pertaining to the introduction and management of data store will increase due to the increase in components called as data store.

(3) XML direct operation type

XML direct operation type is an ORIENT usage implementation method, which creates ORIENT message without using ORIENT communication library and exchanges it with ARMOR. As regards the method of creating ORIENT message, a method wherein the template (XML template) of ORIENT message is arranged in advance and the required data is embedded.

As regards XML direct operation type, implementation of ORIENT usage is comparatively easy if data format is simple in write type.

Advantages and disadvantages

The advantage of XML direct operation type is that ORIENT usage can be easily implemented in case data format is simple. Further, it is sufficient to replace simple strings for creating ORIENT message from XML template as the library which performs communication is provided in various languages.

However, the disadvantages such as not suitable for creating complex XML data, difficult to support analysis engine ,which needs query response, and low interoperability can be cited.

2.3 Summary of selection guidelines for metadata integration platform correspondence method

First, write type, which can reduce the implementation cost in case load against ARMOR is sufficiently less, is recommended for ORIENT usage method. However, the data volume gets oversize in case data is stored by write operation. It is better to use query response type as longs as analysis result is not used frequently.

Communication library usage type is recommended for implementation method of ORIENT usage if communication library is easily embedded and analysis engine is created in Java. It is advisable to use external GW type if analysis engine is not created in Java and analysis engine cannot be easily reconfigured. Further, as regards XML direct operation type, it is not recommended as its compatibility and interoperability are low and can be used if data format is simple.

3. Metadata integration platform correspondence method details

In this section, the concrete processing flow of each metadata integration platform correspondence method are shown with the objective to serve as a reference at the time of designing the architecture to make analysis engine correspond to metadata integration platform. The processing flow and system configuration shown in this section are just examples and are not limited to only these.

3.1 Communication library usage type

Communication library usage type creates message and communicates with ARMOR by using ORIENT communication library. Refer to “Related documents” [1] and [2] for the concrete usage method of ORIENT communication library.

3.1.1 Write type

In Write type, first the message (ORIENT Method), which is sent to ARMOR, is created in ORIENTMethodFactory defined in ORIENT communication library and then it is sent to ARMOR by ORIENTMethodSender. For details, refer to Section 3.2 of Related documents [1].
It is necessary that HTTP access to the port of ARMOR, which works, should be enabled in order to behave as client of HTTP at the time of transmission.

3.1.2 Query response type

In the Query response type, reception class inherits AbstractSyncORIENTMethodReceiver of ORIENT of communication library and response processing is carried out. In response processing after obtaining the contents of query, response is created by adding the data corresponding to that query. For details, refer to “Related document” [2] and Section 2.1.
Query response type is implemented as a servlet, which can be accessed in HTTP. Therefore, it is necessary that HTTP is accessible from ARMOR side.

3.2 External GW type

In external GW type, the server which converts the data into the ORIENT format or program (gateway) is operated apart from analysis engine and analysis result is exchanged between ARMOR and analysis engine via this gateway. As regards external GW type, first common items are explained and then processing flow of respective method is explained as external GW type has been classified into data store usage and communication I/F usage.

3.2.1 Common items

Figure 3 Overview flow of external GW type is a figure, which shows the overview of process flow for write operation and query response in external GW type.

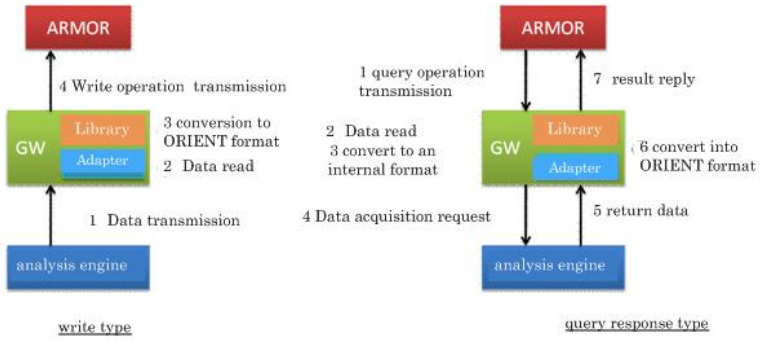


Figure 3 Overview flow of external GW type

In write type, analysis engine transmits the data to the gateway (1 on the left side of Figure), gateway program reads this data via adapter, which can interpret the transmitted data (2 on the left side of Figure), and the data is converted into ORIENT format (3 on the left side of Figure). The data converted in ORIENT format is transmitted (4 on the left side of Figure) to ARMOR by write operation.

On the other hand, in query response type ARMOR transmits the query operation to gateway (1 on the right side of Figure) gateway program reads the received query (2 on the right side in Figure) and converts it into the format, which can be interpreted by analysis engine (3 on the right side of Figure). Then, a request to obtain data is issued to analysis engine (4 on the right side of Figure). At analysis engine side, if analysis result is obtained after the completion of analysis processing, analysis result data is returned to gateway (5 on the right side of Figure). Gateway, which received the analysis result, converts this analysis result into ORIENT format (6 on the right side of Figure) and returns it as a query result to ARMOR (7 on the right side of Figure).

Here, the processing 3, 4 of write type, and the processing 1, 2, 6, 7 of query response type use ORIENT communication library same as communication library usage type. Therefore, for details of these processing refer to related documents mentioned in Section 4.1.

3.2.2 Data store usage

External GW type (data store usage) is configured in such a way so that external gateway hands over the stored data to ARMOR using data store such as RDB without directly communicating with analysis engine. To acquire such configuration in external GW type (data store usage) only query response type is supported (configuration in which analysis engine is replaced by data store as shown on the right hand side of Figure 3 Overview flow of external GW type).

Here, in order to explain it using concrete examples a use case, where sensor data called as real-world monitoring service is processed, is assumed. In the real-world monitoring service, sensor data is appropriately created and

analysis result is created via analysis engine. This analysis result is exchanged with ARMOR in external GW type (data store usage) format.

(1) Component explanation

Figure 4 Real-world monitoring service analysis processing component provides the overview image of components related to analysis processing in real-world monitoring service.

The real-world monitoring system consists of ARMOR, external gateway, analysis result DB and analysis engine. The external gateway plays the role of converting a query from ARMOR into SQL query. The analysis result DB is RDB that stores analysis result. Analysis engine, which creates analysis result, registers analysis result in this analysis result DB.

The detailed behavior of these components is explained in the subsequent processing sequence.

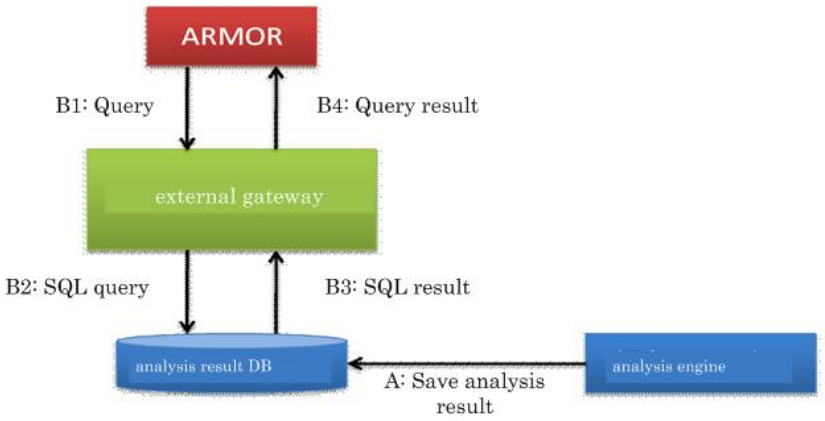


Figure 4 Real-world monitoring service analysis processing component

(2) Sequence explanation

The sequence in data store usage of external GW usage type is divided into query response sequence, which responds to a query from ARMOR, and analysis result write sequence, which writes analysis result of analysis engine in data store. Here, respective sequence is explained.

(2-1) Query response sequence

Figure 5 Real-world monitoring service query response sequence provides sequence image, which shows flow of processing responding to query from ARMOR. An overview of behaviour of each sequence is shown below.

1. Sending ORIENT message

   ARMOR sends query operation to external gateway after receiving application request etc. External gateway is implemented as a servlet using the servlet library of ORIENT communication library.

Therefore, the processing of this part can be implemented in the same way as that of query response processing of Section 4.1.

2. Analysis of Query request

External gateway, which received the Query, analyzes the Query message and changes it into a format that can be processed. Even this processing is simple as a message can be converted into Java objects by using the functions of ORIENT communication library.

3. Acquisition of insert key

It is a processing to obtain insert key, which indicates analysis data to be searched, from Java object that shows query operation. It obtains the value from the received Java object.

4. Acquisition of analysis result

It is a processing, which performs the conversion from insert key to SQL query, and obtains analysis result. First, it creates a SQL query having conditions to obtain analysis result, which carries the insert key obtained in processing of 3. Then, it issues SQL query generated for analysis result DB and obtains results.

5. Conversion of analysis result into ORIENT message

It is a processing, which converts the analysis result having the registration key, specified and obtained in processing of 4, into ORIENT message. For details, refer to related documents same as Section 4.1 as the conversion to an ORIENT message is a Java object operation, which is implemented in ORIENT communication library.

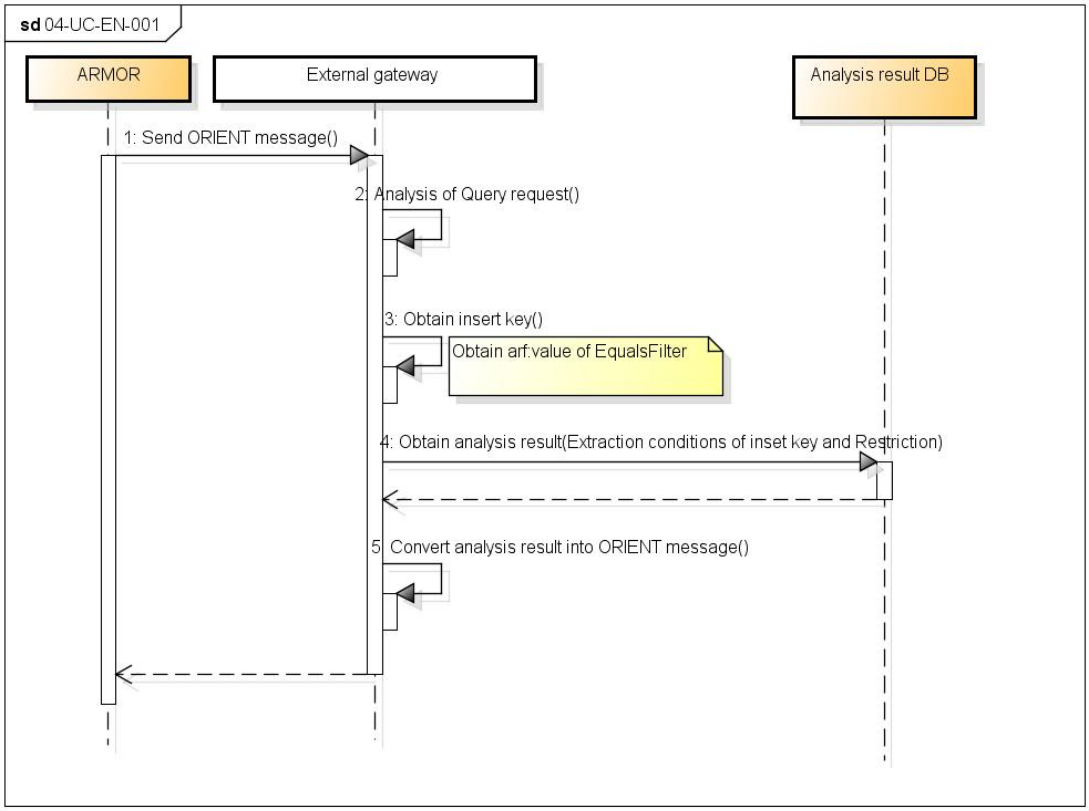


Figure 5 Real-world monitoring service query response sequence

(2-2) Analysis result write sequence

Figure 6 Real-world monitoring service analysis result write sequence provides a sequence image, which shows the processing flow of writes analysis result in data store. In this sequence, analysis engine analyzes the sensor data received from sensor device (not shown in Figure 4) and the result is written in analysis DB. The overview of sequence behavior is mentioned below.

1. Sending sensor data

   In this processing the obtained sensor data is sent to analysis engine by sensor device. Sensor device does need to wait for processing result from analysis engine. Transmission method is optional.

2. Performing analysis processing

   Analysis engine, which receives sensor data, performs analysis processing. This analysis processing is a unique analysis engine processing, which is not described here.

3. Data conversion

   Analysis engine, which creates analysis result, converts analysis result into SQL format that can be written in analysis result DB. At this time, analysis engine adds the carried out time of analysis processing, sensor device ID and metadata, used at the time of obtaining analysis result.

4. Saving data

   Analysis engine, which converts the data, sends analysis result, converted in SQL format, to analysis result DB and saves it. Analysis result DB is RDB. Once analysis result is saved, processing is returned.

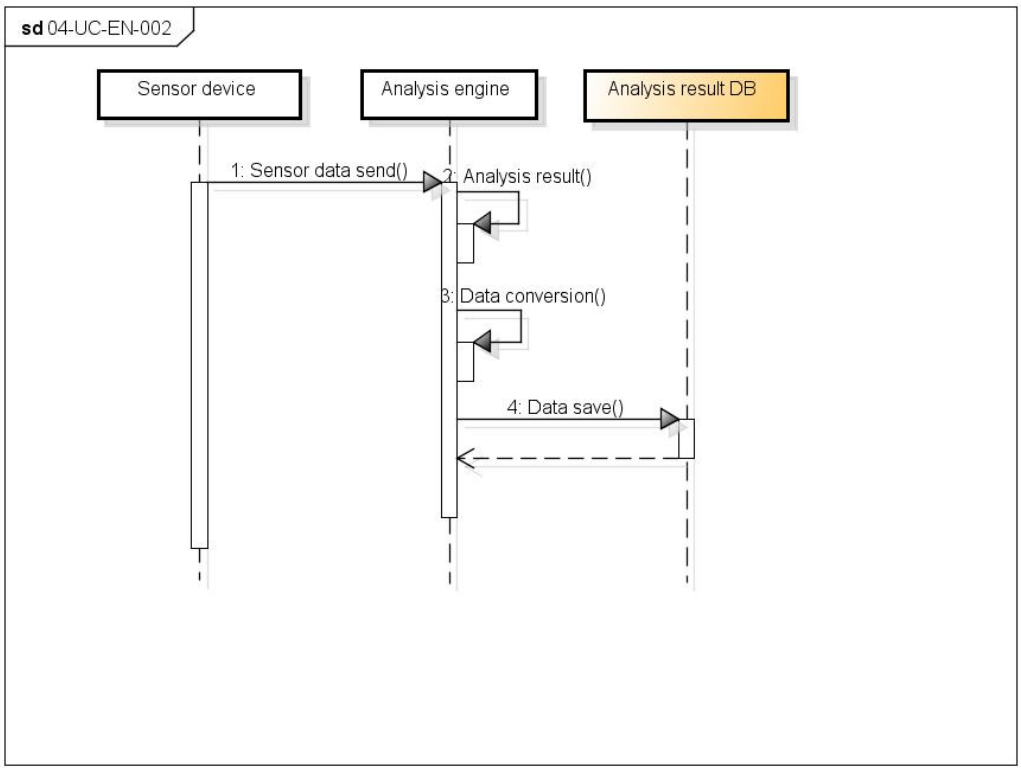


Figure 6 Real-world monitoring service analysis result write sequence

3.2.3 Communication I/F usage

External GW type (communication I/F usage) is not a data store such as RDB. It communicates with analysis engine via communication means. The configuration is same as mentioned in Figure 3 Overview flow of external GW type. As regards communication method, methods such as forwarding JSON or CSV by Java-RMI, HTTP, external command execution can be considered. Further, a method that is more suitable to the requirement is used. For example, if analysis engine can be executed by calling a command, a method such as creating a script to execute analysis engine command in adapter portion and transferring data by file can be considered.

The processing flow is omitted as it is almost similar to the flow described previously.

3.3 XML direct operation type

XML direct operation type is a method that creates ORIENT message using template and hands over analysis data to ARMOR. The configuration and processing flow are shown on the basis of concrete examples. The points, which require attention, are explained. For example, processing flow is shown by using sample code. Here an example using java is given but its implementation is also possible in other programming languages.

In the concrete example, XML template is created on the basis of ORIENT message, created by using ORIENT communication library, and necessary items in the template are replaced in template engine velocity, which makes it easy create document from template. It is assumed that analysis data is recorded in CSV format and embedded in template as data by reading it from CSV. Apache http client library, in which http client functionality is implemented, is used for communication.

Figure 7 XML direct operation type component shows an overview of operation and components in a concrete example. If you refer to Figure 7 XML direct operation type component, the processing for sending analysis data goes as: (1) data is read from CSV file, (2) request is created by using template, (3) request is sent to ARMOR by http and (4) response returned as sending result is analyzed using XML parser.

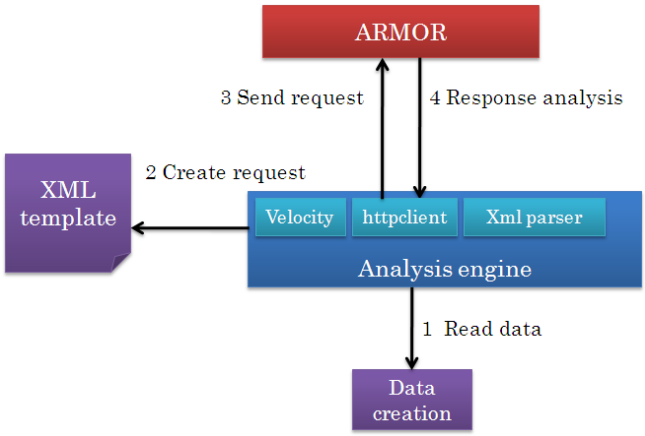


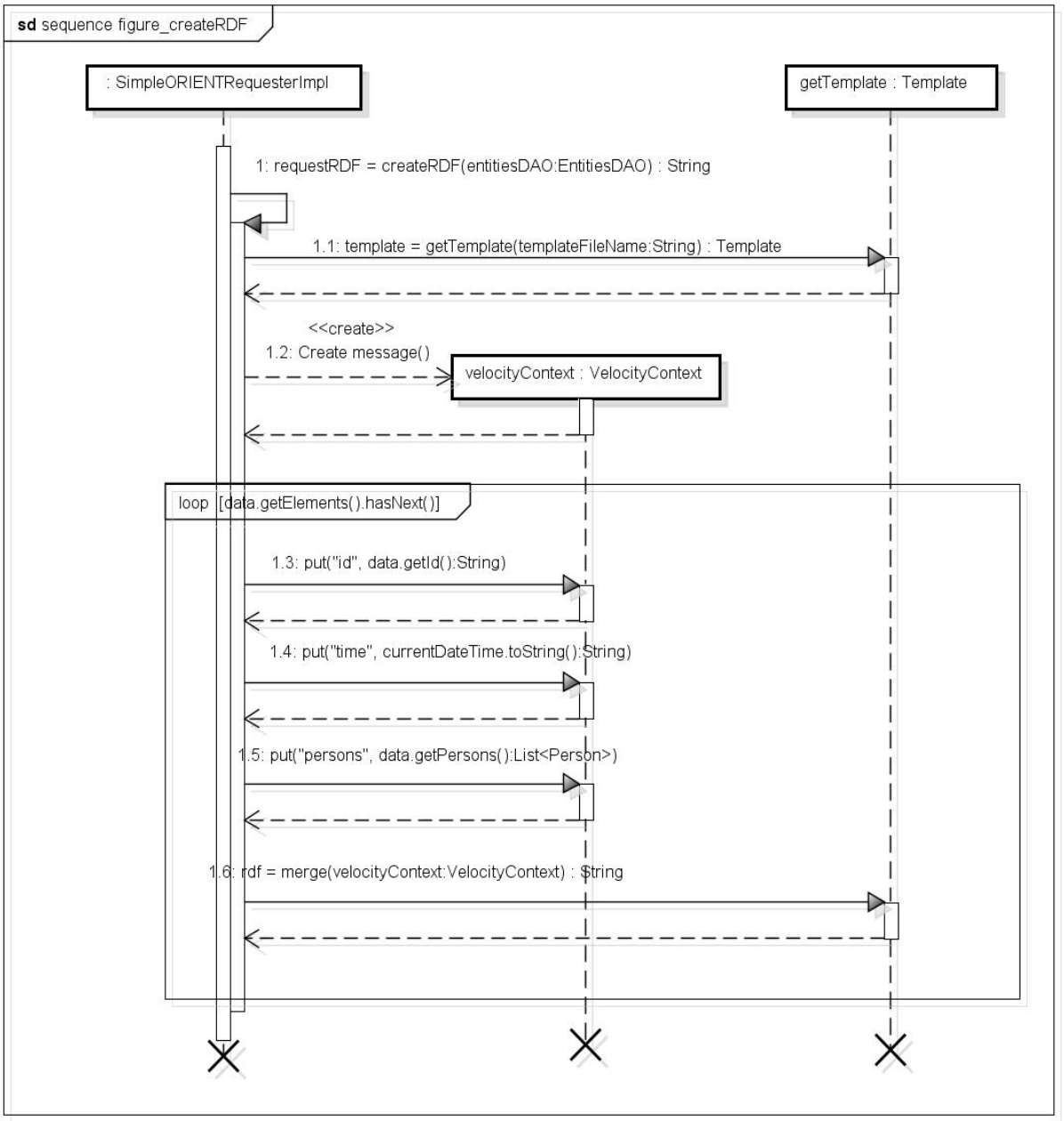
Figure 7 XML direct operation type component

The sequence for respective behavior and attention points are explained below. After the explanation of each processing, method to create template is considered.

The explanation of data creation is omitted as it is self explanatory.

(1) Create request processing

(1-1) Sequence



1) Template file is read from file system.
2) Data ID is obtained from JSON format data and stored in context of template.
3) Present date and time are obtained and stored in context of template.
4) Data, inserted in Entities of Write request, is obtained from JSON format data and stored in context of template.
5) RDF is created and returned.

(1-2) Sample code

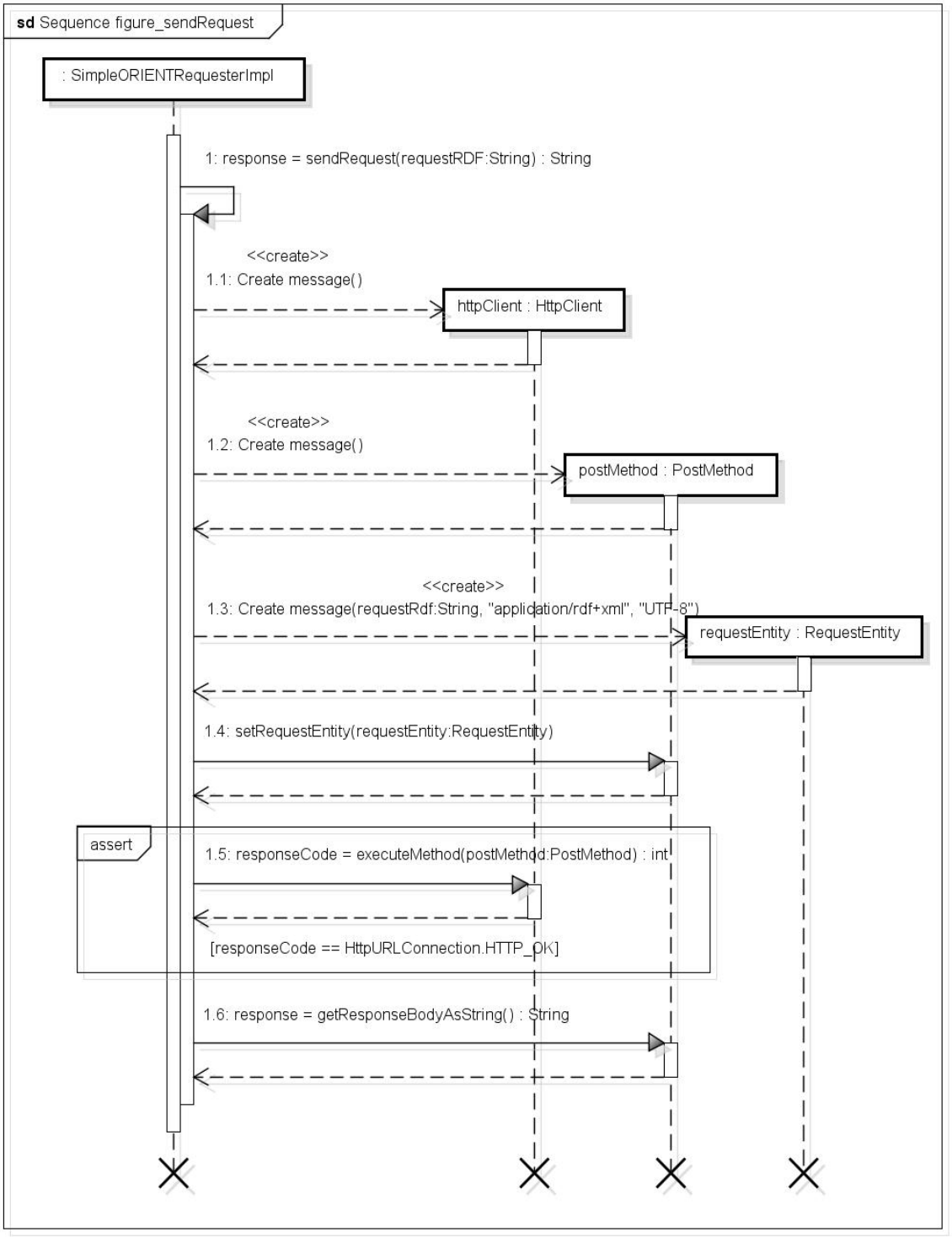
```
public String createRDF(EntitiesDAO entitiesDAO) {
    /** Prepare context of Velocity. */
    VelocityContext templateContext = new VelocityContext();
    // value to be configured in ${id}
    templateContext.put(CONTEXTKEY_ID, entitiesDAO.getId());
    // value to be configured in ${time}
    templateContext.put(CONTEXTKEY_TIME, new DateTime());
    // value to be configured in ${count}
    templateContext.put(CONTEXTKEY_COUNT, entitiesDAO.size());
    // value to be configured in ${persons}
    templateContext.put(entitiesDAO.getName(), entitiesDAO);

    /** Get template of Velocity. */
    Template template = getTemplate();

    /** Replace template by using Velocity. */

    return createRDF(template, templateContext);
}
/** Get template from setting file. */
private Template getTemplate() {
    // Omission
}
private String createRDF(Template template, VelocityContext templateContext) {
    /** Prepare Writer that stores the result of after replacement. */

    StringWriter stringWriter = new StringWriter();

    /** Replace string in Velocity. */
    template.merge(templateContext, stringWriter);

    /** Convert replaced result into string. */
    return stringWriter.toString();
}
```

(1-3) Implementation notes

- Implementation is possible by using optional template library, which dynamically creates string through context such as Velocity (Java), HTML::Template (Perl), Smarty (PHP) etc.
- By defining RDF/XML format as template ORIENT, communication library and XML creation library are not required.

(2) Send request processing

(2-1) Sequence



1) Instance of HTTP client is created.
2) Destination to send ORIENT request is set.
3) Contents type and character code, which conform to ORIENT specification, are configured in HTTP header and RDF of ORIENT request 1 is configured in HTTP body.

4) Request is sent by HTTP POST.

(2-2) Sample code

```
public String sendRequest(String requestRDF) {
    /** Read URL of sending destination from setting file. */

    String armorUrl = ShieldSimpleORIENTRequesterConfig
            .getOrientArmorURIForFacialPictureCheck().toString();

    /**Prepare HttpClient. */
    HttpClient httpClient = new HttpClient();

    /**Prepare Post method. */
    PostMethod postMethod = new PostMethod(armorUrl);

    /** Set ContentType and character encoding. */
    RequestEntity requestEntity = new StringRequestEntity(requestRDF,
            HTTP_CONTENTTYPE, HTTP_CHARSET);
    postMethod.setRequestEntity(requestEntity);

    /** Send message by Post of HTTP. */
    int responseCode = httpClient.executeMethod(postMethod);
    if (responseCode != HttpURLConnection.HTTP_OK) {
        sLog.error(MESSAGE_HTTP_RESPONSE_CODE_IS_NOT_SUCCESS);
        throw new ShieldEngineRuntimeException();
    }

    /** Get contents of response. */
    return postMethod.getResponseBodyAsString();
}
```

(2-3) Implementation notes

- Implementation is carried out in accordance with "6 Protocol (Transmission protocol)" of related document [3].

  - The following protocols and serialization are used in sending and receiving of ORIENT data.
    - Protocol: HTTP/1.1 POST method
    - Serialize: RDF/XML format(UTF-8)

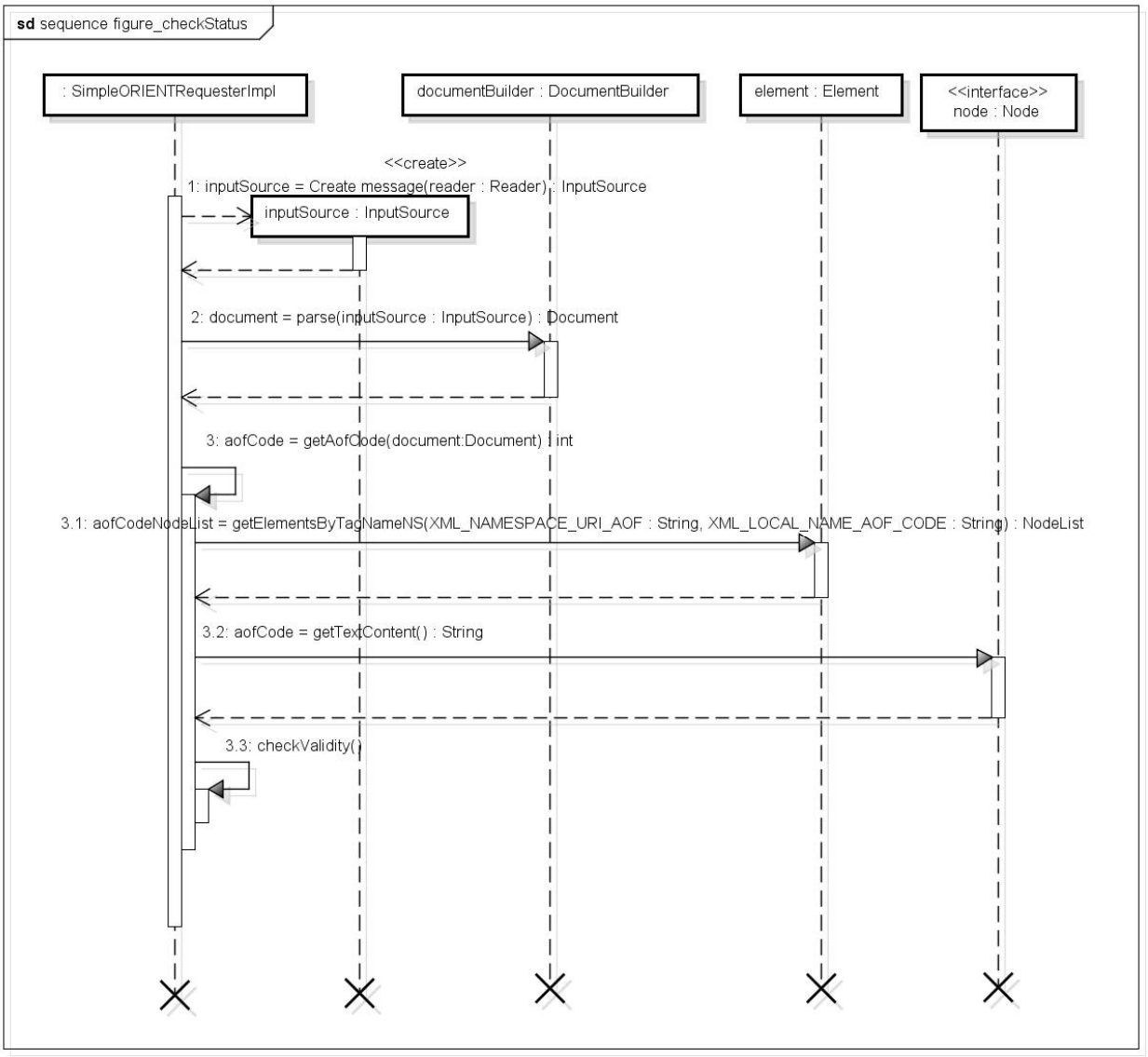
  - The following methods are used in case of requesting ORIENT operations.
    - RDF/XML data is sent as message body of HTTP POST.
    - Value of HTTP header is as follows.

HTTP header name	Parameter	Mandatory/Optional	Value
Content-Type	-	Mandatory	application/rdf+xml
Content-Type	charset	Mandatory	UTF-8
Content-Length	-	Optional	size of RDF/XML data is expressed in decimal octet

- It is possible to do the implementation using any programming language and library by conforming to the above mentioned specifications.

(3) Response analysis processing

(3-1) Sequence



1) XML document is created from response.
2) Reference Model Node URI obtains node of “http://www.nec.com/ivcp/orient/aof#code” from the created XML document.
3) Contents text of the obtained “aof:code” node is obtained.
4) Status is determined.

(3-2) Sample code

```
public void checkStatus(String response) {
    /** Preparation for reading string from response. */
    Reader reader = new StringReader(response);
    InputSource inputSource = new InputSource(reader);

    /** Preparation for XML parser. */
    DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
    documentBuilderFactory.setNamespaceAware(true);
    DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();

    /** Read response by XML parser. */
    Document document = (Document) documentBuilder.parse(inputSource);

    /** Obtain response code from aof:code. */
    int aofCode = getAofCode(document);

    /** Perform check of response code. */
    if (aofCode != CODE_AOF_STATUS_IS_NOT_SUCCESS)
        throw new ShieldEngineRuntimeException();
}

private int getAofCode(Document document) {
    /** Obtain root element from document. */
    Element element = document.getDocumentElement();

    /** Obtain aof:code. */
    NodeList aofCodeNodeList = element.getElementsByTagNameNS(
            XML_NAMESPACE_URI_AOF, XML_LOCAL_NAME_AOF_CODE);

    /** If aof:code does not exist, regard it as error due to incorrect response,. */
    if (aofCodeNodeList.item(0) == null)
        throw new ShieldEngineRuntimeException();

    /** Convert aof:code into numerical value. */

    return Integer.valueOf(aofCodeNodeList.item(0).getTextContent());
}
```

(3-3) Implementation notes

- Implementation is possible by using a general XML parser through obtaining the target node specifying Reference Model Node URI (ORIENT communication library is not required).
- In the acquisition method, which traces the tag name till target node, there is a possibility of failure in acquisition as path can differ depending on the variation in RDF expression.
- The implementation is possible by using a general XML parser. However, in this case there are no merits of being in a RDF format. If it is Java, then it is recommended to use ORIENT communication library.
- RDF parser such as “PHP RDF Parser” and “Simple JavaScript RDF Parser” etc. are also open in other programming languages.

(4) Template file

Template file, which is used at the time of sending ORIENT request to ARMOR, is described. The template shown here is created assuming it to use in velocity. The string enclosed with ${} are template strings and these are replaced by actual values by velocity processing. After showing the template, content by which each template string is replaced is shown.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF
      xmlns:arf="http://www.nec.com/ivcp/orient/arf#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:aof="http://www.nec.com/ivcp/orient/aof#"
      xmlns:ds_mst="http://www.nec.com/ivcp/orient/ds/media-stream#"
      xmlns:ds="http://www.nec.com/ivcp/orient/ds/general#"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:person="http://www.nec.com/ivcp/orient/ds/person#"
      xmlns="http://www.nec.com/ivcp/orient#"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:abf="http://www.nec.com/ivcp/orient/abf#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xml:base="http://www.nec.com/ivcp/orient/">
      <abf:Vocabulary rdf:about="http://nec.com/osdk/resource/id/auto/vocabulary/${id}">
            <abf:import rdf:resource="ds/media-stream#"/>
            <abf:import rdf:resource="ds/person#"/>
      </abf:Vocabulary>
      <abf:Operation rdf:about="../write/operation">
            <abf:hasRestriction>
                  <abf:Restriction rdf:about="http://nec.com/osdk/resource/id/auto/restriction/${id}"/>
            </abf:hasRestriction>
            <dc:date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">${time}</dc:date>
            <rdf:type rdf:resource="aof#Write"/>
      </abf:Operation>
      <abf:Entities rdf:about="http://nec.com/osdk/resource/id/auto/entities/${id}">
            <abf:operatedBy rdf:resource="../write/operation"/>
            <abf:count rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">${count}</abf:count>
            <abf:restrictedBy rdf:resource="http://nec.com/osdk/resource/id/auto/restriction/${id}"/>
            <abf:entities>
                  <rdf:Bag>
#foreach($person in $persons)
                        <rdf:li>
                              <ds:Person
rdf:about="http://nec.com/osdk/resource/id/auto/entity/${person.objectKey}">
                                    <ds:type rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
                                      ${person.type}
                                    </ds:type>
                                    <dc:identifier rdf:resource="${person.objectKey}"/>
                                    <dc:source>
                                          <ds_mst:MediaStream rdf:about="${person.registrationKey}"/>
                                    </dc:source>
                                    <rdf:type rdf:resource="abf#Entity"/>
                              </ds:Person>
                        </rdf:li>
#end
                  </rdf:Bag>
            </abf:entities>
      </abf:Entities>
```

${id} …　ORIENT request ID. It is obtained from data file.

${time} …　ORIENT request execution date time. It is obtained from system date time.

${count} …　cases of ds:Person. It is calculated from data file.

${person.objectKey} …　dc:identifier of ds:Person. It is obtained from data file.

${person.registrationKey} …　dc:source of ds:Person. It is obtained from data file.

${person.type} …　ds:type of ds:Person. It is obtained from data file.

(5) Template file creation method

The method to create template file used in XML operation type is described.

(5-1) Types of creation method

The following three methods are the main methods used for template file creation.

- Use ORIENT communication library
- Use sample
- Creation by referring the ORIENT specifications

Each method is briefly explained below.

(a) Method using ORIENT communication library

It is a method to create template using ORIENT communication library. By using ORIENT communication library, RDF model based on ORIENT specification is constructed, even without knowing the details of ORIENT specification details and RDF model, and message can be sent or received.

In the ORIENTMethod interface provided by ORIENT communication library, API is provided to serialize the RDF model created using API in RDF/XML format. It is possible to serialize the RDF model in file by using this API. Further, it can be used as template file using the serialized result.

(b) Method using sample

It is a method to create template file by modifying template file used in sample application.

In case DSM configuration differs, it is necessary to describe the child elements included in Entities elements in XML in advance. It is also possible that the content of Vocabulary elements require correction.

Moreover, it is also necessary to add the declaration of name space prefix. The method using ORIENT communication library is recommended if one does not want to perform these operations.

(c) Creation by referring to ORIENT specifications

It is a method to create template from the beginning by referring the message examples of ORIENT specifications. In addition to the understanding of ORIENT specifications, it is also necessary to understand RDF model. It is recommended to create template using either of the method mentioned above as it necessary to classify the problem whether it is a template problem or any other problem if a problem occurs and the degree of difficulty is high.

(5-2) Template creation method example: Method using ORIENT communication library

Among the template creation methods, method which uses using ORIENT communication library, is explained.

(a) Processing sequence

Template is created by the following processing sequence.

1) Write Operation request is created by using ORIENT communication library.
2) The created Write Operation request is serialized and generated in a file.
3) The portion, which is to be replaced in the output result, is corrected.

(b) Portion required replacement

The portion, which needs to be replaced in the file created by the below mentioned method, is summarized below.

- ID automatically created by ORIENT communication library

  Resource ID is automatically created by ORIENT communication library. The value of this portion needs to be changed for each message. It is better to replace the autoid.XXXXX portion as the resource ID generated automatically is in the following format.

http://nec.com/osdk/resource/id/auto/local name of resource/autoid.XXXX

  The value from XXXX onwards differs for each message.

- dc:date property of abf:Operation

  As regards the property of dc:date of abf:Operation, ORIENT communication library automatically sets the current date and time when Operation element is generated. It is necessary to replace this value by the time, when message is sent.

- abf:count property of abf:Entities

  The value of this property shows the number of Entity possessed by Entities. In the example shown above, it has two Entities but it is necessary to change the value for each message as it is believed that number of Entity will differ depending on the sent messages.

- Property value of each Entity

  The property value of each Entity is sent by configuring it in each message. The value that is to be included in the message is configured from program.