Building a protocol validator for Business to Business Communications

Rudi van Drunen, Competa IT B.V. (r.van.drunen@competa.com)
Rix Groenboom, Parasoft Netherlands (rix.groenboom@parasoft.nl)

Abstract

In this paper we describe the design and implementation of a system essential to enable the deregulation of the energy market in the Netherlands. The system is used to test and validate secure communications using XML messages through the AS2 standard between the business partners in the market. The tool is comprised of an Enterprise Service Bus component, a service virtualization component, a database with business logic and an user interface added. The version 1.0 of the system was built in less than one month.

1. Introduction

The energy market in the Netherlands has been deregulated. To facilitate this a communications structure is designed in which the different energy suppliers are able to communicate to each other. This communications structure and protocol is critical in the supply of energy (electricity, gas) from supplier to consumer. To validate these protocols a simulation and validation environment is required, and used for certification of the different market parties. This experience report discussed the implementation of such an certification environment for the gas transportation; more of these tools will be required for other market processes and will be constructed in a similar fashion.

The exchange of messages in the protocol is built up in different layers. The transport medium is the Internet, and every supplier has connected their production systems to the Internet, to reach all others. In the next layer the transport protocol is http, secured by ssl, so we're using standard https here. Then the data protocol is AS2, which provides a way to send (XML) data in an encrypted manner using S/Mime to an authenticated receiver. The actual payload is an XML message that should adhere to the Dutch energy interconnection standard.

The goal is to provide a test environment (in the project called Testtool) that can be used to certify over 100 market parties to see whether they adhere to a new XML definition that is being rollout during a migration process taking about one year. There are over fifty application and protocol level test-scenarios that need to be checked for each party before they are certified and can participate in the new communications infrastructure.
2. Architecture

To validate the complete communication stack the engine should be able to validate and test the communication on every level. Building this engine from scratch is a major undertaking, and will not be flexible. Therefore we choose a number of building blocks to technically implement the validation engine.

The https and AS2 communications are being handled by an Open Source Enterprise Service Bus (ESB) solution (UltraESB\(^1\)), which then hands the XML payload to a product called Virtualize\(^2\). Virtualize is used as a service virtualization engine which tests the validity of the XML message, handles responses and stores the data in a database. The actual validation (as part of the certification process) is done using the data in the database and checking on the right contents and sequence of the XML payloads. The database is a MySQL database that stores all messages and metadata of the messages and the sender / receiver. All of this is driven by a number of Java classes that can be controlled by a Web GUI.

Figure 1 shows the global architecture of the Testtool. The different messages (notification (NOT) and Message Delivery Notice (MDN) as Acknowledge) are asynchronously for external communications and synchronously for internal communications. The external party is simulated in the test environment as depicted using the Meldelson\(^3\) tooling.

![Figure 1: Overview of the Testtool](image-url)
2.1 AS2

The protocol on AS2 level allows for multi level encryption of the data on the wire, as well as validation of the communication partners using certificates. Next to that communications can take place in a synchronous and asynchronous manner.

The AS2 message enters the system through a https connection. A standard server certificate, issued by a trusted certificate authority is used to validate the sender and decrypt the ssl transmission. The setup corresponds with the way an ssl-enabled webserver is set up. Important is that the Testtool, as well as the partner, can work both with one and two-way SSL.

As the ESB solution incorporates the webserver, the server certificate, as well as the (path to the) root CA needs to be present in the Java keystore it uses (a so-called identity.jks). This setup is shown in Figure 2.

![Figure 2: ssl setup for AS2 messages](image)

The OpenSource ESB we employ in this project is UltraESB. This engine has the ability to handle the (AS2) communication with the partner in a synchronous and asynchronous manner, while it communicates in a synchronous fashion to the backend (in this case the Virtualize engine). Configuration and adding functionality to UltraESB turned out to be simple by changing the configuration files that are a mix of Java and XML.

As the protocol supports bi-directional communication we need to not only accept messages but also be able to send them. We needed to add extra functionality to the ESB to facilitate sending messages to partners, when the test environment is initiating the communication. We did this by adding functionality to select the correct certificate that belongs to the business partner with whom we need to set-up the AS2 communication. A custom http-header field: dest-endpoint is used to communicate this information from Virtualize to UltraESB.

Second, when we are receiving the, in this case asynchronous, acknowledgement message we should end the handshake directly and not send a formal reply from the backend to the business partner again. In that
situation, the ESB would not forward the ‘response’ message, but direct it to "ground". We used this approach to have a clear separation of concern: the ESB deals only with the communication, only the Virtualize components are ‘aware’ of the type of message. If it is an incoming notification message that needs acknowledgement, Virtualize responds. If the incoming message is an acknowledgement we do not have to respond Virtualize adds the “ground” designator to the response message.

As there were no throughput requirements, as the system was to handle one communication at a time, with a payload maximum of about 1 Mbyte. We did not expect issues with buffering, and relied on internal buffers of the ESB. In practice this has proven to be sufficient. The ESB allowed us to set the maximum payload size, which will be set to 10 Mbytes in the production version. Together with the appropriate settings of the java VM (max heap size) we were able to handle this maximum payload size without problems.

2.2 PKI

To authenticate and encrypt / decrypt the XML messages on AS2 level a Public Key Infrastructure is being used as shown in Figure 3. Every partner in the energy market in the Netherlands has a certificate that is signed by a root that is public for the Dutch energy market, and is maintained by the government. This PKI is used in the ESB software to maintain authentic and safe communications. The path to and the root CA, as it is not a standard trusted CA as well as the certificates of every business partner must be known to the ESB application. The local certificate must be made known to every partner as well. The ESB certificates are maintained in a different Java keystores and controlled by a GUI.

![Figure 3: Message level encryption using PKI](image)

If an AS2 message enters the ESB, it will be authenticated by checking the certificate against the CA, and then decrypted with the local (private key) certificate. If the ESB needs to send a message, it encrypts it with the public
key of the partner and sends it on the wire. The Virtualize component used 3 http-headers to tell the ESB which certificates and endpoint to use.

For testing this process, we have used a separate AS2 client (Mendelson) that we can load with the certificates of the business partner so we can simulate and test the AS2 communication before bringing the application ‘live’. This helps in the troubleshooting of the correct loading of the certificates.

2.3 Virtualize

The simulation of the service that handles the communication between the partners is implemented using Parasoft Virtualize. The Virtualize toolkit was readily available and had all the functionality we needed for the backend. Next to that the customer had ample experience with other Parasoft tooling. In this setup Virtualize accepts XML messages on a Tomcat queue, checks the messages against the XML schema definitions and validates them. Based on the message content, Virtualize generates the response messages for a specific partner and sends it on to the ESB as http message. To let the ESB know which partners information to use in forwarding the message, Virtualize adds three custom http-headers:

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest-endpoint</td>
<td>AS2 endpoint where the ESB should deliver the message</td>
</tr>
<tr>
<td>cert-alias</td>
<td>The alias of the certificate in the keystore in the ESB that should be used for the encryption of the MIME payload</td>
</tr>
<tr>
<td>as2-id</td>
<td>The AS2 identifier of the business partner</td>
</tr>
</tbody>
</table>

Virtualize supplies the parameters to these headers based on the business partner identifier and the information stored in the backend database.

To be able to mimic asynchronous behavior, we need to be able to have a message that is generated in Virtualize not to be forwarded to the partner as described before we add the “ground” designator as value to dest-endpoint.

Using the scripting facilities of Virtualize we created python methods that are fired off when specific messages arrive. These scripts log the (validated) message and metadata in a database to be inspected on the contents, and respond though the ESB in a correct AS2 message to the client.

2.4 Database

In order to check the contents of the XML and its validity, the message is logged in a database. Next to the message itself, meta information on the business partner and timestamps are stored. This information than can be used to actually check the sequence and detailed contents of the XML message and report to the partner.
Besides storage of the incoming messages, the database also contains the different messages the tool should be able to send for certain test-scenario. This requires a trade-off between storing complete messages or only the relevant payload information and let Virtualize compose the correct XML message with right information. The database offers enough flexibility to support both approaches.

2.5 Database application

The application to validate the message transfer is a Java application that supports the process of validation of a partner, in which a number of test cases need to be performed. Each test case is a scenario in which a number (>= 1) of messages containing specific information need to be sent to or received from the test tool. The application can evaluate the database contents and report on the correct sequence and contents of the messages, and then flag a test (case) as succeeded or not.

Next to this functionality the database application implements a portal for maintaining clients, and querying the database for certain test scenarios executed by partners, or provide a partner with the ability to have the test system initiate a test in the direction of the business partner.

3. Experiences / Usage

The described project has a high business value, as partners are obliged to certify themselves before being allowed to do business on the Dutch energy market. Certification ensures the peer-to-peer communication, and thus the national protocol will work. As market introduction depends on certification of the partners the deadlines are pretty tight. The development team managed to build a version 1.0 of the system within a month. This can be regarded as a good accomplishment, also given the fact that the messaging specification on a functional (which test-scenario’s to support) and technical level (final versions of the XML schema definitions) was still evolving during the development.

Problems encountered on the way were partly due to the different ways the business partners have set up their production systems and the PKI. During the process of testing we encountered problems in validation of the certificates as different partners had certificates signed by an older version of the root certificate, and thus were not able to be authenticated. The error messages from UltraESB did not point us in the right direction right away. This problem was solved in the checklist and standard operating procedure.

Other issues that we have seen were due to the fact that the system was required to run on a Windows platform that was present as a VPS in the cloud. The acceptance system and the development systems were set up for a single developer use so making parallel development more complicated.
Technical issues were seen in the management of the Windows firewall through the Java application, as we wanted to close the system off for all other IP numbers than those of the partners that registered for the validation. Managing the Windows firewall rules was difficult, so we implemented access lists on the Apache webserver as part of the ESB. Other difficulties were encountered in the file format of certificates and the multiple different keystores. Also the inner workings of some of the products used were not as expected, and needed adaption in the user code.

The next (2.0) phase of the project, currently on its way features the building of several web interfaces to facilitate the management of partners and the certification / validation process, in such a way that partners can start up their own validation process and see the progress of their certification.

4. Lessons Learned

A number of lessons can be learned from this project.

First of all, separation of the three main functionalities of Testtool, has been key to the success of this development. Using an of-the-shelf ESB to handle the (AS2) communication, the Virtualize engine to handle the XML payloads including technical validation of the message and the database application to support the actual validation process. This clear separation in architecture has proven to be important throughout the development process, although fine-tuning the interface between these three components required frequent detail adjustments.

Second, the modular design of Testtool, turned out to be very helpful to handle the changed specifications. Virtualize was programmed using a draft set of XSDs that in a later stage where replaced by the final versions. The core functionality depended only key (stabled) elements in the header of the XML message. Similarly, incomplete sets of the certificates influenced only the testing of the AS2 connectively not the overall development of the Testtool.

Another important issue in this project is that a solid PKI should be one of the requisites for starting. Designing a PKI and setting it up during a project that technically uses the PKI components yields to issues in communication.

An observation during the project resulted in the best practice of having infrastructure development to be ahead of software development. This will result in having a solid base to plan tests. In this project the infrastructure was designed and built alongside of the software environment, which resulted in some planning issues, especially with the tight deadlines we were facing.

Finally, the complete application could have been built from scratch, but as this was not an option within the project, readily available and re-usable “off-the-shelf” components were found in UltraESB and Virtualize. Even the custom build Java GUI will be re-usable for other certification environments that are
required in the future. The combination gives a flexible setup to handle changes in transport protocol, message formats and test-scenario’s. The use of the UltraESB and Virtualize combination turned out to be the right choice for building this tool.

5. Conclusion

Without a validation system and the system driving it, it is not possible to have a large number of partners using a complex protocol securely and reliably amongst each other, a requirement in the Dutch deregulation of the energy market. The validation service has moved to full production successfully. Whenever a new partner wants to join the energy market the validation and certification of their backend communication systems has become a simple job. This system, quickly built out of a number of off the shelf components, provides a quick way to validate communications, and can be reused for other protocols in the near future.

6. Acknowledgements

The authors want to thank all people involved in the project. Next to the customer and their very helpful staff, Richard Jaspers of Competa IT and Georg Tornqvist of OELAN are thanked for their work in implementing the software needed. Thanks go to Dirk Giesen of Parasoft Netherlands for his guidance and expert advice.

1 Ultra ESB: http://www.adroitlogic.org/
2 Virtualize: http://www.parasoft.com/
3 Mendelsson: http://as2.mendelson-e-c.com/