EUROPEAN COMMISSION
DIRECTORATE-GENERAL
INFORMATICS
Information systems Directorate

European Commission

Open e-TrustEx 2.6.0
Software Architecture Document

Date:	14/08/2020
Version:	2.6
Authors:	Sandro D'OrazioSandro D'Orazio, Sandro D'Orazio , Cristian Chiriac
Revised by:	Maarten Daniels, Anamaria Batrinu
Approved by:	Chrysanthi Giortsou, Tanya Chetcuti
Public:	
Reference Number:	

TABLE OF CONTENTS

Document History

Version	Date	Comment	Modified Pages
2.6	14/08/2020	Adaptation corresponding to release 2.14. (admin console use case diagram has been updated)	
2.5	09/03/2020	The document has been adapted to hold the information for both internal and open source version of the application. The diagrams generated with EA have been replaced with diagrams generated with Confluence Gliffy.	
2.4	09/10/2019	Corrected the ICD link	[REF1]
2.3	07/02/2019	Updated the supported maximum file size from 100MB to 500MB	6, 26
2.2	21/09/2017	Review and update corresponding to 2.0 release	Where needed
2.1	03/11/2014	Corrections following review	
2.03	14/10/2014	EIP Diagram updates Removed JAX-WS references from text Other minor corrections	
2.02	24/09/2014	Diagram updates Other minor corrections/ improvements	
2.01	04/12/2013	Version 2.01 (final)	All
2.00	15/11/2013	Submission of version 2.00 for review	All

1. Introduction

1.1. Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system.

1.2. Scope

The architecture described in this document concerns the system e-TrustEx developed and used by the European Commission as part of the ISA programme.

1.3. References

#	Document	Contents outline
[REF1]	e-TrustEx Interface Control Document	Interface control document for front-offices and external systems. This document is available inside the European Commission network as well as in the release documentation for the open version of the application.
[REF2]	Java EE	Java Enterprise Edition is Oracle's Java platform for enterprise application development. It contains APIs for ORM, SOAP and RESTful web services, distributed components, JMS and much more.
[REF3]	Enterprise Integration Patterns	Patterns and best practices for Enterprise Integration by Gregor Hohpe
[REF4]	Spring Framework	An open-source framework for Java enterprise application development, featuring Dependency Injection, Aspect-Oriented Programming, support for web application development, JDBC, JPA, JMS, SOAP and RESTful web services and much more.
[REF5]	Spring Integration	Extension of the Spring programming model to support the well-known Enterprise Integration Patterns
[REF6]	Spring Web Services	Spring Web Services is a product of the Spring community focused on creating document-driven Web services
[REF7]	Schematron	Rule-based validation language for making assertions about the presence or absence of patterns in XML trees
[REF8]	WildFly Application Server	Java EE certified open source application server
[REF9]	EJB	Enterprise Java Beans
[REF10]	SOAP	Simple Object Access Protocol

[REF11]	WS-Security	Standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality
[REF12]	WS-Policy	The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe the policies of a Web Service
[REF13]	WS-SecurityPolicy	WS-Policy defines a framework for allowing web services to express their constraints and requirements. These are security related policies
[REF14]	X.509	Public-Key Infrastructure (X.509)
[REF15]	MTOM	*Message Transmission Optimization Mechanism* is used for the efficient transmission of binary data to and from SOAP web services by compressing the data and sending it in a separate MIME part.
[REF16]	HTTP Chunking	A mechanism by which data is broken up into a number of chunks when sent over an HTTP connection.

1.4. Definitions

#	Document	Contents outline
[REF17]	Document	A Document in the scope of the e-TrustEx platform is a piece of information structured using XML
[REF18]	Transaction	A Transaction in the scope of the e-TrustEx platform represents an operation on a document. The submission of a document bundle through the system is an e-TrustEx transaction
[REF19]	Profile	A document exchange profile aggregates a set of transactions.
[REF20]	Party	A Party is an entity exchanging documents through the platform.
[REF21]	Bundle	Set of multiple document wrappers exchanged by the Parties playing a role in a data exchange scenario.
[REF22]	Document wrapper	A Document Wrapper is an entity composed of a binary file and its metadata, which can be exchanged through the platform
[REF23]	Interchange Agreement (or ICA)	An Interchange Agreement represents a Contract between two or more Parties on the use of e-TrustEx for the electronic exchange of information in the context of one Profile.
[REF24]	CIA (or C.I.A.)	Confidentiality, Integrity and Availability

[REF25]	Business Domain	A Business Domain represents the specific context of a business.

1.5. Document Content Overview

After summarizing the architectural representation, goals and constraints, this document describes the system using several architectural views (Use Case, logical, process, deployment, implementation and data) and then concludes with size, performance and quality considerations.

2. ARCHITECTURAL REPRESENTATION

The next two sections of the document describe architectural goals and constraints.

A Use Case diagram describes architecturally relevant Use Cases. A short explanation of their impact on the architecture accompanies the diagram. The following views will also be provided:

- A logical view provides a basis for understanding the structure and organization of the design of the system through its components and their interactions.
- An implementation view describes the software layers and the main software components by using a component diagram.
- A deployment view provides a description of the hardware components and the relation between them. This view gives a technical description of protocols and hardware nodes used.
- A data view provides information about data persistency. A class diagram models main system data.

UML diagrams are systematically used to represent the different views of the system.

3. ARCHITECTURAL GOALS AND CONSTRAINTS

The following non-functional requirements have been identified:

Non-functional requirement	**Description**
Support of large files	The application shall allow the transfer of large files (up to 500 MB each) of any type.
Support of groups of files	The application shall allow the transfer of groups of linked files.
Interoperability	The application shall allow the connection of heterogeneous systems.
Confidentiality	The system allows ensuring that transferred documents are not viewed by anybody else than the sender and the final recipient.
Integrity	The system allows guaranteeing integrity of the transferred files.
File storage	A dedicated file system shall be used to store binaries, but it needs to be set up by the technical support team.

4. SECURITY

4.1.1. Introduction

The e-TrustEx platform supports requirements for authentication, integrity and validity of documents during transmission and storage. Besides those, it also includes measures for authorization, confidentiality, auditing and non-repudiation.

The security requirements supported by e-TrustEx are described as follows:

- *Confidentiality* is needed to prevent third parties from eavesdropping on information that is being transmitted.
- *Authentication* ensures that the parties involved in communication are really who they say they are.
- *Authorization* ensures that users only have access to the resources they are allowed to.
- *Integrity* guarantees that a message is not modified during its transmission.
- *Validity* certifies a stored message does not lose its legal attributes.
- *Auditing* controls enable relevant parties like authorized bodies to inspect transactions afterwards.
- *Non-repudiation* measures prevent users from denying actions they have undertaken.
- *Storage security* concerns measures taken to secure data stored in the database and file storage
- *Availability*

To provide an answer to these requirements the platform provides the means to link Confidentiality, Integrity and Availability (C.I.A.) levels to a document interchange agreement between parties. The Retrieve Interchange agreement service allows the sender to retrieve the C.I.A. information before sending a document.

The following chapters of the document describe how the platform implements the security requirements.

4.1.2. Confidentiality

The connection to the platform web services can be done using HTTPS. HTTPS needs to be configured in the Wildfly AS ([REF8]), otherwise simple HTTP is used. HTTPS is a secure version of the HTTP protocol and is being used to protect data transactions. It uses one way Secure Socket Layer (SSL) and digital certificates ([REF14]) to encrypt a data transfer session over an otherwise insecure HTTP connection. Implementers of the platform should only allow connection to the open, e-Trustex services though https connection to guarantee confidentiality of message exchange especially in cases where message level confidentiality is not used.

The use of HTTPS guarantees transport level confidentiality however there might be a need to ensure it also at message level. The platform provides support for end-to-end encryption and acts as public key repository (used to encrypt the message). It also provides the sender the information about the Confidentiality level of the message exchange through the Retrieve Interchange Agreement service.

4.1.3. Authentication

By default, the system requires basic authentication over the HTTPS connection. The fact that HTTPS is used to set up the initial secure connection makes it difficult for someone to steal the passwords by listening on the communication as it is encrypted.

If stronger authentication is required, implementers can decide to impose the usage of two-way SSL connections to access the platform web-services. In that case, the client will have to provide a certificate trusted by the server to establish the SSL connection and thus authenticate to the server using this certificate. This configuration is external to the platform as it depends on the implementer's IT infrastructure, and thus out of the scope of this document.

4.1.4. Authorisation

The platform comes with a complete authorisation implementation based on the concept of interchange agreements.

A **Document exchange profile** is a set of *transactions* that define operations (e.g. submission, retrieval, etc.) on specific documents. An **Interchange Agreement** models the contract between parties in the context of a specific document exchange profile.

Every time a party tries to access the system, e-TrustEx checks if there is an interchange agreement authorising the caller to do so.

4.1.5. Integrity

The platform guarantees the integrity of message exchange though the use of digital signatures ([REF11],[REF12],[REF13]). The messages returned by the platform are signed and parties can be required to sign their calls to the platform web-services. The use of XML digital signatures is an optional feature that is configurable in the system. The use of XML signature is not mandatory as it adds complexity to the solution and, depending on the business case, HTTPS encryption might be good enough to ensure the required level of integrity.

This integrity is only valid for the transport and the platform's signatures are purely technical and have no "business" value. The platform does not provide out of the box support for end to end signature. However, if such business signatures are required, the platform will provide the information to sender parties through the Retrieve Interchange Agreement service.

4.1.6. Validity

The platform supports different types of message validation, the standard XSD validation and the Schematron ([REF7]) validation, a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. This guarantees the business validity of the messages stored in the system. Some more complex checks can be implemented in Java (e.g. if parent reference exists).

4.1.7. Auditing

The platform logs all the service calls and stores the transactions in the system's database. Auditors can get temporary access to the platform database and log files in case of control.

4.1.8. Non Repudiation

When a party submits a message, e-TrustEx generates a signed acknowledgement containing information about the submitted message. This signed acknowledgement can be used as a proof of submission ensuring the non-repudiation of (submission of) messages sent through the platform. The party submitting the message is responsible for storing the acknowledgement generated by e-TrustEx.

4.1.9. Storage Security

The databases that contain business data, logging information and audit trails is hosted in a secure environment where only authorized users have access. Audit trails are kept to ensure that the data is not tampered with. Additionally, it is assumed that regular backups of the databases are generated and securely archived.

The large attachments files are encrypted before being stored on the file system. In addition to the file encryption, proper segregation of duties policies shall be applied to ensure that people having access to the encryption key cannot access the file system and vice versa.

For the Open Source version of the platform, storage security is the responsibility of the implementer.

4.1.10. Availability

The e-Trustex platform uses JMS queues for time-consuming operations. Thread pools can be configured in order to limit the total number of concurrent users to ensure good service availability.

SECURITY DISCLAIMER

On top of the security e-TrustEx (including the Administration Console) provides, the user shall add an extra security layer at application server level. DIGIT shall not be held responsible for any security breach that might occur due to User not respecting this recommendation.

5. USE-CASE VIEW

This section provides a representation of the architecturally significant use cases.

5.1. Selection Rationale

The architecturally significant use cases have been selected based on the following criteria:

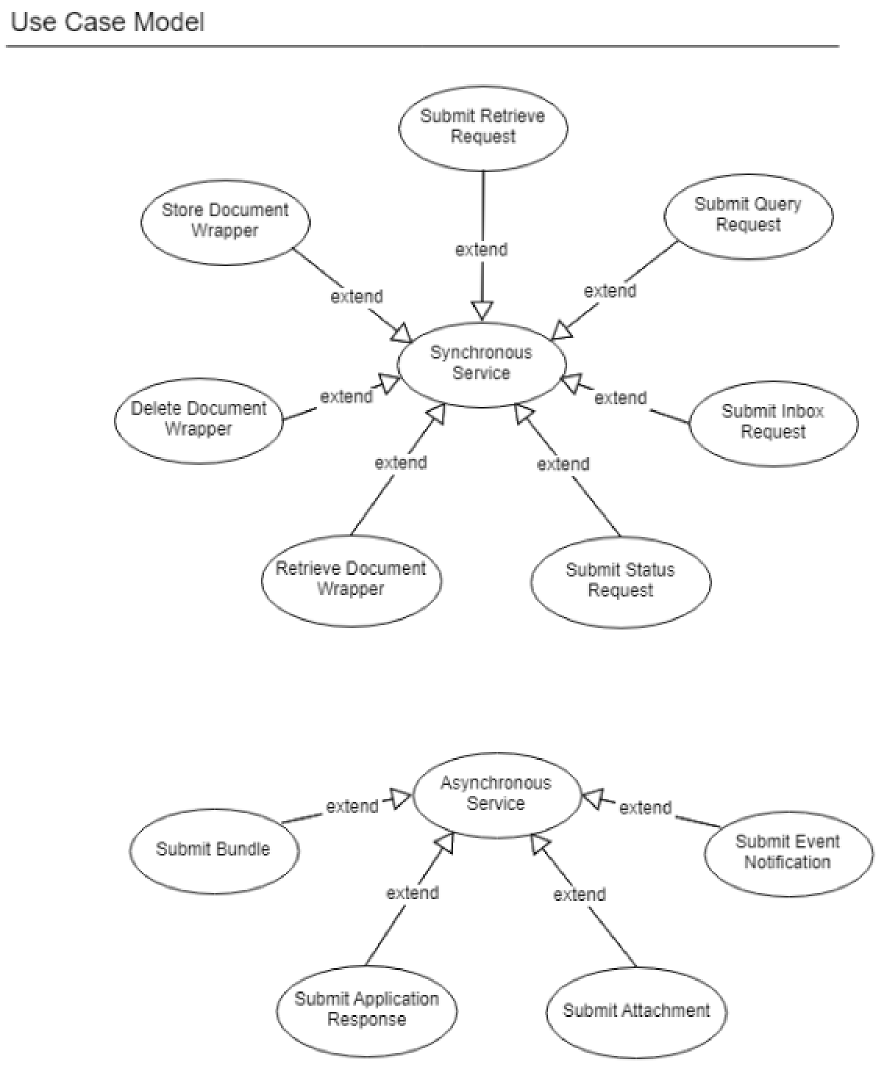
– Use cases affecting multiple components of the system, thereby providing a cross-cutting view of the system architecture;

– Use cases representing critical parts of the architecture, thereby addressing the technical risks of the project at an earlier stage.

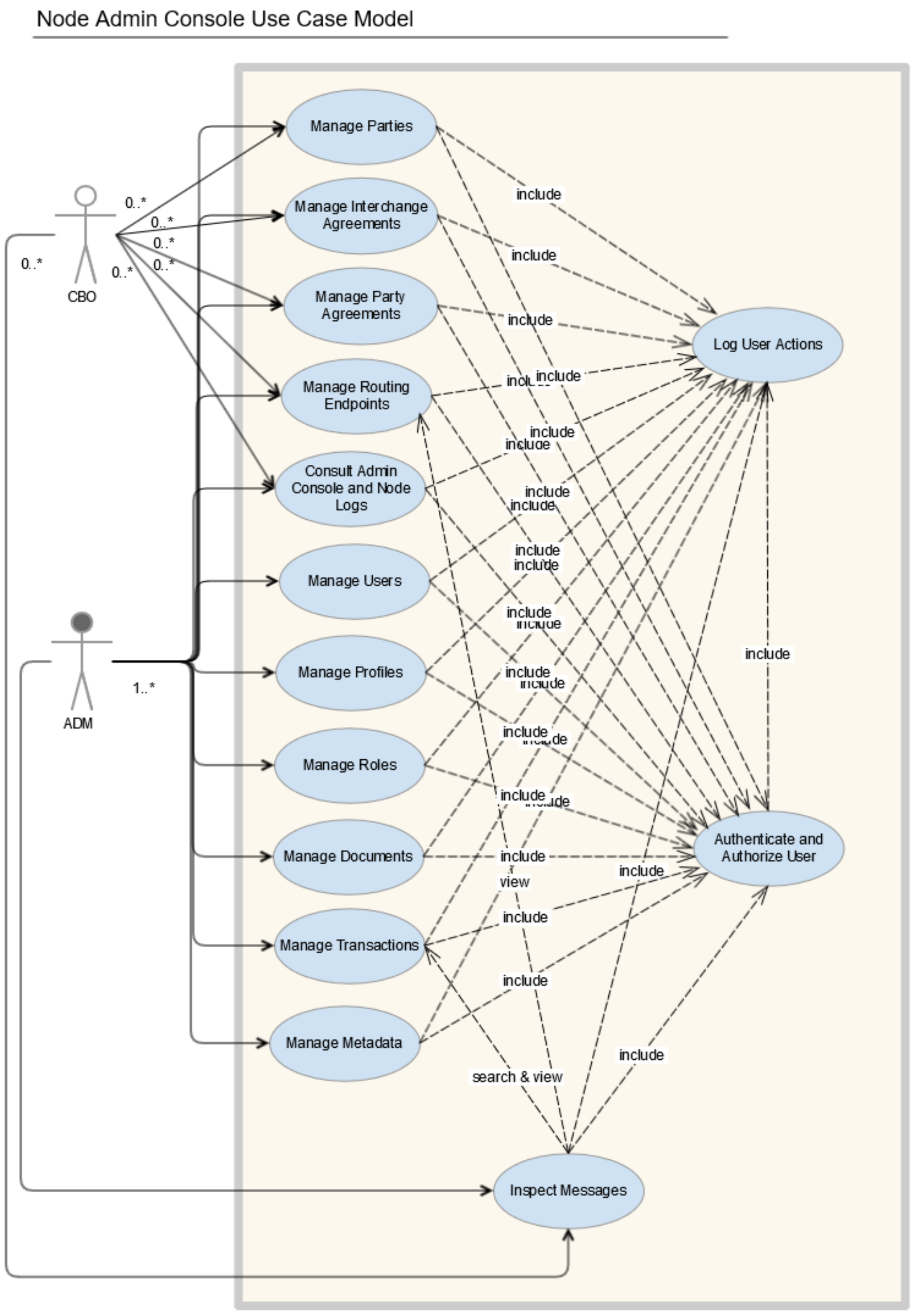
The following use cases have been selected:

- Use cases addressing the synchronous services offered by the platform
- Use cases addressing the submission of (potentially large) binary files (UC Store Document Wrapper)
- Use cases addressing the asynchronous services offered by the platform

The following Use Cases diagram displays the e-TrustEx use cases:



The diagram below displays the main use cases implemented in the e-Trustex Admin module.



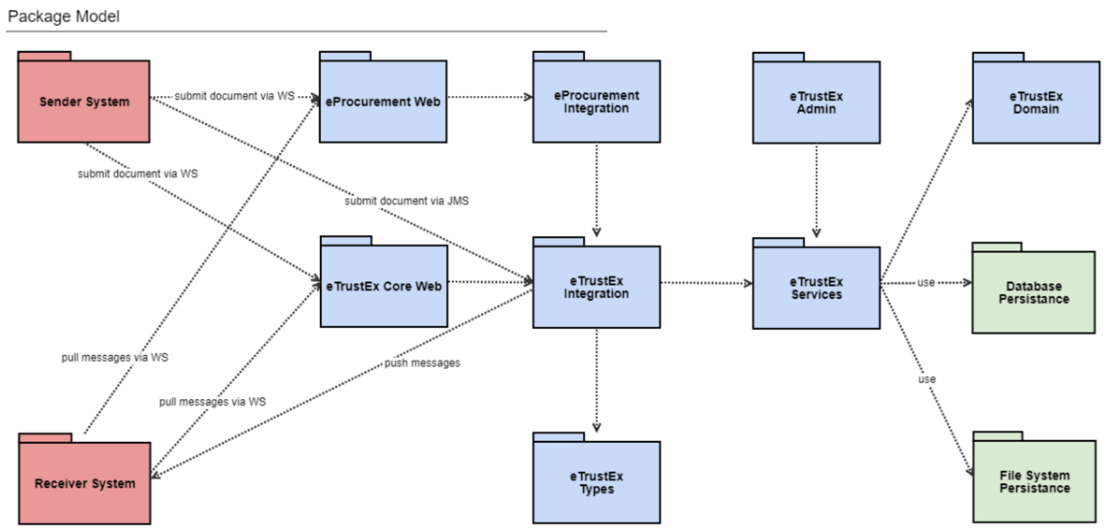
6. Logical View

6.1. Overview

This chapter describes the main application modules, how these modules interact and implement system use cases.

6.2. Architecturally Significant Design Packages

The following diagram provides a high-level view of the main packages composing the system. The Database persistence and file system persistence are logical packages representing the physical data storages used by the platform. The other packages represent different application layers and give an overview of the organisation of the platform's code as each of them translates into a separate maven project.



6.2.1. Sender and Receiver systems

Sender systems can submit documents using the platform web-services or by pushing them into JMS queues.

Receiver systems can either retrieve their messages polling the platform read services or receive them via the e-TrustEx store and forward mechanisms. E-TrustEx can forward messages via JMS or by using a "call-back" web service implemented by the receiver system.

6.2.2. e-TrustEx core web layer

This module is a Java J2EE web application offering web services interfaces to the clients of the platform. It offers a generic SOAP endpoint using the Spring WS framework and specific web services features dealing with the large attachments (MTOM - [REF15] - and Http Chunking - [REF16]).

6.2.3. e-TrustEx Integration

This is the core of the message-processing platform. This module is a lightweight ESB based on the Spring Integration framework ([REF5]). It provides common message processing building blocks for checking authorisation, storing and validating the incoming documents and finally routing them to the receiver's system.

6.2.4. e-TrustEx types

This package contains all the java classes generated from the different WSDLs and XSDs used by the system. The integration layer described in previous section is using this JAXB annotated objects to parse incoming requests and to build responses.

6.2.5. e-TrustEx Domain

This is the platform's domain layer; it contains EJB3 ([REF9]) entities implementing the domain objects. This layer is further described in section 8.1 Data Model of this document.

6.2.6. e-TrustEx Services

This is the platform's services layer that provides access to the database, the file system, the integration layer and the Administration Console component. Services are POJOs that are configured in the dependent components using the Spring Framework Dependency Injection mechanism ([REF4]).

6.2.7. e-TrustEx Storage (file system and database)

The platform uses a database to store its configuration and the exchanged messages. The file storage is used to store the binary files submitted to the platform.

6.2.8. e-TrustEx Administration Console (a.k.a. eTrustEx Admin)

The administration web console of the platform facilitates the configuration of the system. This is a standard java J2EE web application using Spring Model View Controller pattern implementation and JQuery for the presentation layer. It uses EU Login for authentication and Spring Security for authorization. The Open Source version uses basic authentication and it is recommended that implementers use their own authentication meachanism.

6.2.9. eProcurement Web

This Java EE ([REF2]) web application module exposes SOAP web service endpoints for eProcurement. The web services implementation is Spring WS ([REF6]).

6.2.10. eProcurement Integration

This module processes messages coming via the web services in the eProcurement Web component. The processing chain is implemented with Spring Integration ([REF5]).

6.3. Use-Case Realizations

The following use cases have been chosen to describe how software packages behave:

- A sender system stores a document wrapper (binary file)
- A Party calls a Synchronous service
- A Party calls an Asynchronous service

There are two kinds of services implemented in the platform:

- the asynchronous services that are used by a party to send a document (e.g. submit bundle service),
- the synchronous services that are used to handle large binaries or to query and retrieve documents like store document wrapper or retrieve request

As any use case of the e-TrustEx Integration module derives from the generic synchronous and asynchronous use cases, they will be detailed in this chapter of the document. Storing of a large binary by the system is also relevant for this chapter as, even if it is also a synchronous service, it slightly deviates from the generic one.

As mentioned in the previous chapter, e-TrustEx uses the Spring Integration framework ([REF5]). This framework is an implementation of Gregor Hohpe Enterprise Integration Patterns (EIP - [REF3]). The following chapters will use EIP notation diagrams to describe the message processing flows.

Here are the basic concepts for messaging systems as defined on the EIP website that will come in handy to understand how the platform works:

"***Channels*** *— Messaging applications transmit data through a Message Channel, a virtual pipe that connects a sender to a receiver. A newly installed messaging system doesn't contain any channels; you must determine how your applications need to communicate and then create the channels to facilitate it.*

Messages *— A Message is an atomic packet of data that can be transmitted on a channel. Thus to transmit data, an application must break the data into one or more packets, wrap each packet as a message, and then send the message on a channel. Likewise, a receiver application receives a message and must extract the data from the message to process it. The message system will try repeatedly to deliver the message (e.g., transmit it from the sender to the receiver) until it succeeds.*

Multi-step delivery *— In the simplest case, the message system delivers a message directly from the sender's computer to the receiver's computer. However, actions often need to be performed on the message after it is sent by its original sender but before it is received by its final receiver. For example, the message may have to be validated or transformed because the receiver expects a different message format than the sender. A Pipes and Filters architecture describes how multiple processing steps can be chained together using channels.*

Routing *— In a large enterprise with numerous applications and channels to connect them, a message may have to go through several channels to reach its final destination. The route a message must follow may be so complex that the original sender does not know what channel will get the message to the final receiver. Instead, the original sender sends the message to a Message Router, an application component and filter in the pipes-and-filters architecture, which will determine how to navigate the channel topology and direct the message to the final receiver, or at least to the next router.*

Transformation *— Various applications may not agree on the format for the same conceptual data; the sender formats the message one way, yet the receiver expects it to be formatted another way. To reconcile this, the message must go through an intermediate filter, a Message Translator, that converts the message from one format to another.*

Endpoints *— An application does not have some built-in capability to interface with a messaging system. Rather, it must contain a layer of code that knows both how the application works and how the messaging system works, bridging the two so that they work together. This bridge code is a set of coordinated Message Endpoints that enable the application to send and receive messages.*"

The format of the Messages exchanged through the e-TrustEx platform is XML.

Additional information on message routing may be provided upon request.

6.3.1. Common message processing components

Some "message processing" steps are common to all types of messages processed by the platform. In order to avoid code duplication these common behaviours have been implemented using reusable components. These components need to access the platform service layer and are implemented as EIP Service Activators.

The ***Authorisation service activator*** is responsible for assessing if a party is allowed to call a specific service. In order to do that it uses the credentials provided to the system through HTTP(S) basic authentication and retrieves the Party linked to these credentials (see Section 8.1), the *CallerParty* (also known as *IssuerParty*). The system then tries to extract the *SenderParty* from the incoming message. If this required piece of information is missing, or the specified Sender Party is not known by the system, the platform will return an unauthorised access error to the caller.

The e-TrustEx platform offers support for third parties, meaning that one party can delegate to another one the right to perform transactions on the system. Therefore, the service activator checks if the *CallerParty* and the *SenderParty* are the same and, if they are different and no delegation exists between them, the platform will return an unauthorised access error to the caller.

If a *ReceiverParty* is specified in the incoming message, the system will query its database to determine if there is an interchange agreement involving the two parties pointing to a profile containing the transaction related to the occurring service call. If no *Interchange Agreement* is found the platform will return an unauthorised access error to the caller. Specifying a *ReceiverParty* is mandatory for all asynchronous services.

The ***XSD Validation Service Activator*** is a configurable component that applies XSD validation on the incoming XML message. The XSD is linked to a document or a transaction in the system metadata. The platform allows the user to either provide an URL pointing to the XSD or to store it directly in the system database. The XSD validation failure will either trigger a SOAP fault if executed synchronously or lead to the generation of an Application Response error message if executed asynchronously.

An *Application response* is an XML document generated by the system to notify the caller asynchronously of errors that occurred during the message processing or, more generally speaking, to notify changes in the message state (e.g. a message moving to ERROR state). This type of document is also used by parties willing to modify the state of a message via the *Submit Application Response* service. Readers can find more information on the topic in the data view chapter of this document (Section 8.1).

The ***Schematron Service Activator:***

"Schematron is a rule-based validation language for making assertions about the presence or absence of patterns in XML trees. It is a structural schema language expressed in XML using a small number of elements and XPath. In a typical implementation, the Schematron schema XML is processed into normal XSLT code for deployment anywhere that XSLT can be used"

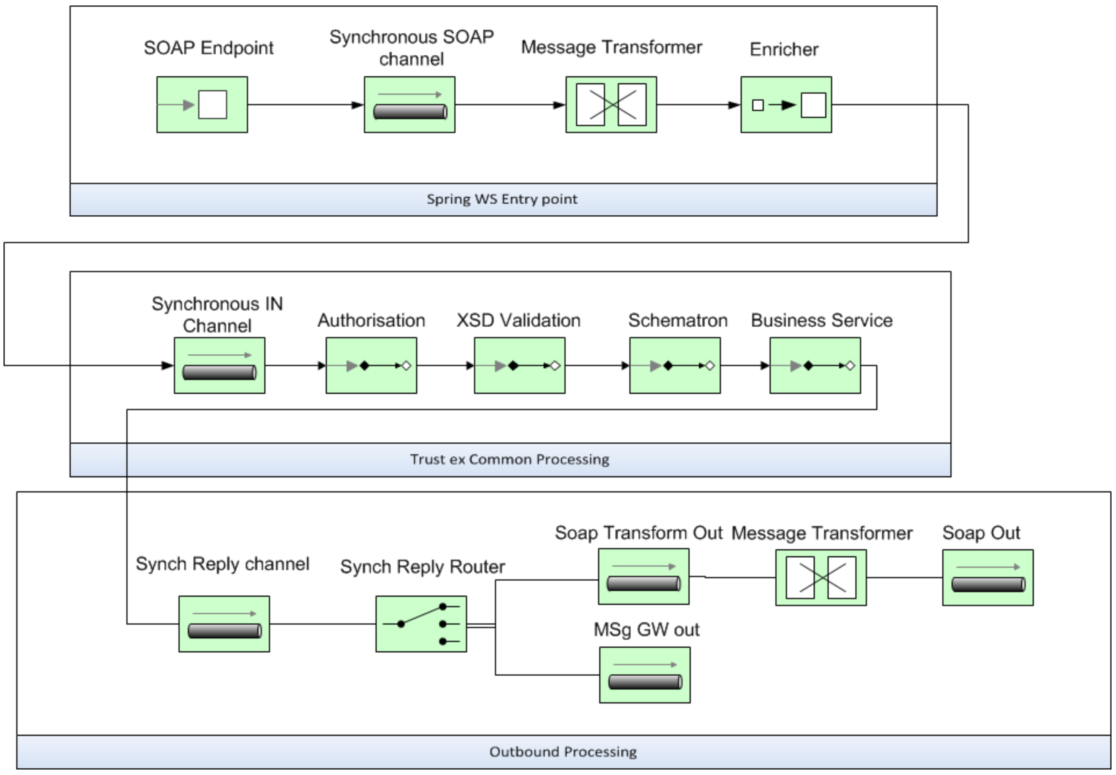
This service activator applies the schematron XSLT transformation to the incoming XML document and handles possible errors in the same way the XSD Validation service activator does. The XSL templates are stored in the system's database and linked to the transactions or documents defined in the platform.

Schematron ([REF7]) is used to implement business validation rules on incoming XML messages that are not supported by standard XSD validation.

The ***Business Service Activator*** is responsible for calling the java object implementing the specific business logic for the service. This must be configured in the Spring application context and must implement specific platform java interfaces. The Business service activator retrieves the business service using either a naming convention or using metadata that can be stored in the platform's database.

6.3.2. *Synchronous services*

This is the EIP representation of the message flow for synchronous services:



The platform offers two entry points to provide synchronous services. They are both SOAP ([REF10]) based but use different web service implementations.

The Spring WS entry point uses Spring WS framework ([REF6]) to provide a generic SOAP endpoint for synchronous services. It routes incoming messages to the appropriate channel based on the message content (content based routing). Incoming messages are routed to the Synchronous SOAP channel where they are transformed into e-TrustEx platform messages, a specialisation of the Generic Spring integration Message by the Message Transformer component.

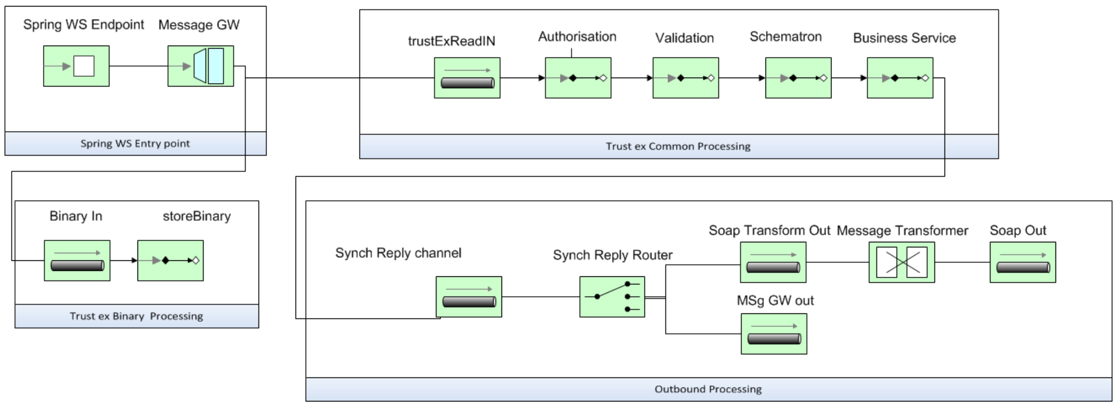
This entry point directly deals with the XML avoiding binding it to java objects. The Message Transformer runs a set of configurable XQueries on the incoming XML to extract relevant information like the message ID, the sender and receiver party IDs and performs some basic integrity checks. The extracted information is set in the header of the e-TrustEx platform message.

Some routing metadata is added to the header by the Header enricher component. This metadata is used by the platform to determine to which channel the response must be submitted.

The incoming message, after the initial transformation into a platform message, is routed to the e-TrustEx common processing message handling chain where it is validated and processed by the business service linked with the service being called.

6.3.3. *Store document wrapper*

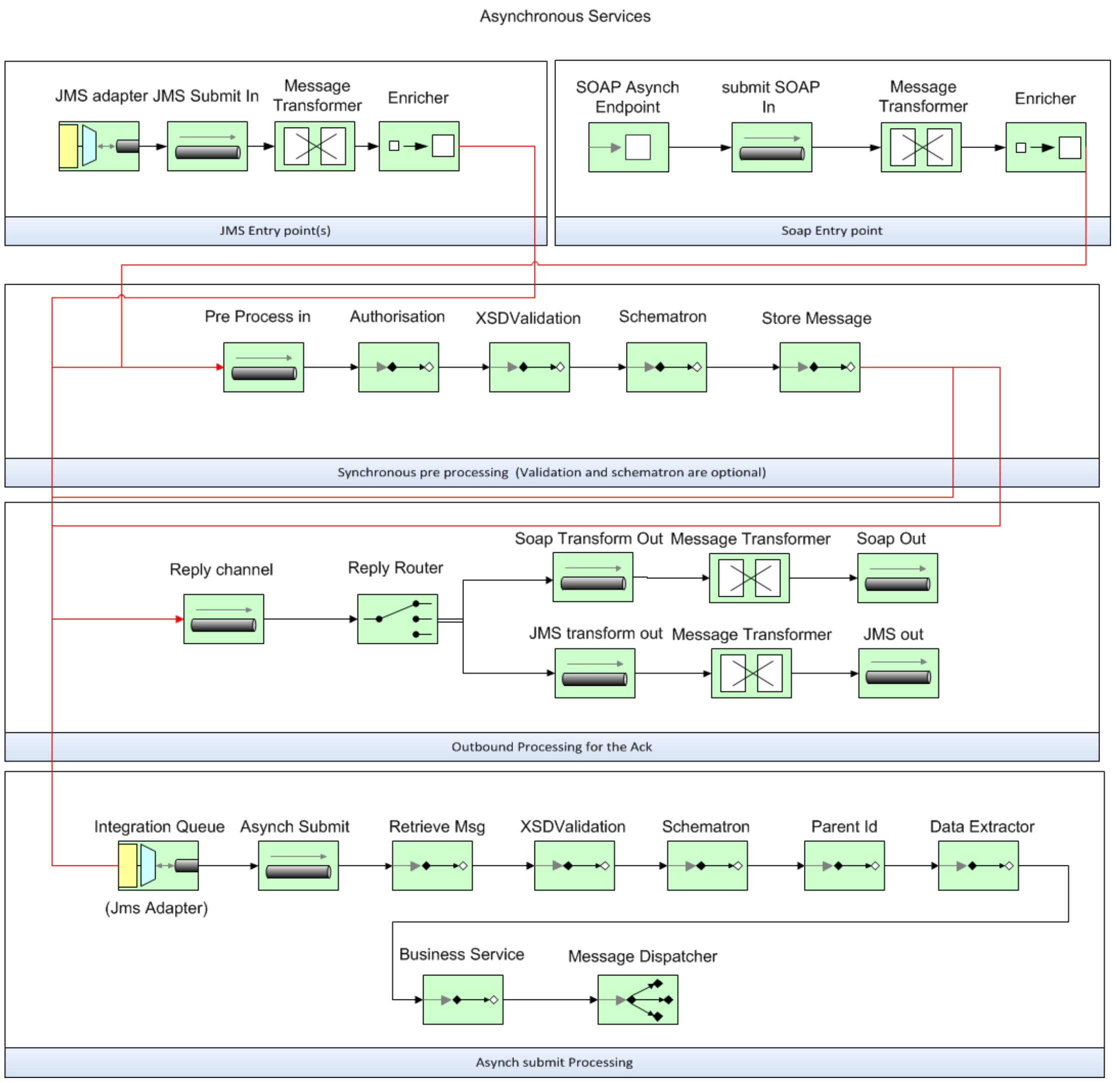
This service slightly deviates from the other synchronous services offered by the platform because the Document wrappers, containing potentially large binary files, are processed in two steps. In the first step, the system streams the binary attachment to the file system and removes it from the incoming XML message to avoid high memory consumption. In the second step, it sends the rest of the XML message to the common synchronous message handling chain. Here is the EIP representation of the Store Document Wrapper message flow:



The *Store Document Wrapper* web service interface supports MTOM ([REF15]) to optimize the size of the transferred encoded binary size and HTTP chunking ([REF16]) to allow the streaming of the binary file to the disk.

6.3.4. Asynchronous services

The following diagram represents the message flow for the asynchronous services provided by the system.



The platform provides asynchronous services via SOAP ([REF10]) or JMS interfaces. The SOAP entry point for asynchronous services uses the same components as the one for synchronous services.

The JMS entry point is composed of one JMS queue and other message processing components. The message adapter retrieves a message from the JMS queue and sends it to the JMS submit channel where it is transformed into a platform internal message and enriched. The security configuration of the JMS resources is application server dependent and will not be detailed in this document.

Asynchronous submission of messages to the platform is a two-step process. The first step is the pre-processing of the message. During this step the platforms performs the authorisation verification and some basic integrity checks on the incoming message. This pre-processing is done synchronously for SOAP calls and asynchronously for messages submitted via JMS. It is also possible to configure the platform to perform the XSD and Schematron validation of the message in the pre-processing chain if the message is coming from the SOAP entry point. At the end of this synchronous processing, the message is stored in the system's database and routed to the ‘outbound processing for the Ack’ message handling chain and the asynchronous processing channel via the system's internal integration queue.

The second step of the processing is the Asynchronous processing of the message by the platform. In order to avoid sending big XML messages in the internal JMS queue, the pre-processing chain just sends the internal id of the message that needs to be processed. The first step of the Asynchronous processing is thus the retrieval of the message. The message is then validated against the configured XSD and schematron files (optional).

The system supports parent child relationships between messages. It is possible to configure in the platform's metadata an XQuery to retrieve the parent id of a specific document. The message transformer extracts this ID from the message and places it in the header of the e-TrustEx message, ensuring its availability through the message processing. The ***Parent id Service Activator*** is responsible for linking the two messages in the system's database. The system allows configuring its behaviour in case the parent document cannot be found. The system can be configured to wait for the parent a given time period, to simply ignore the absence of the parent or to generate an error in case of missing parent.

The Data Extractor is responsible for extracting data from the message by using XPath expressions and saving the results to the database.

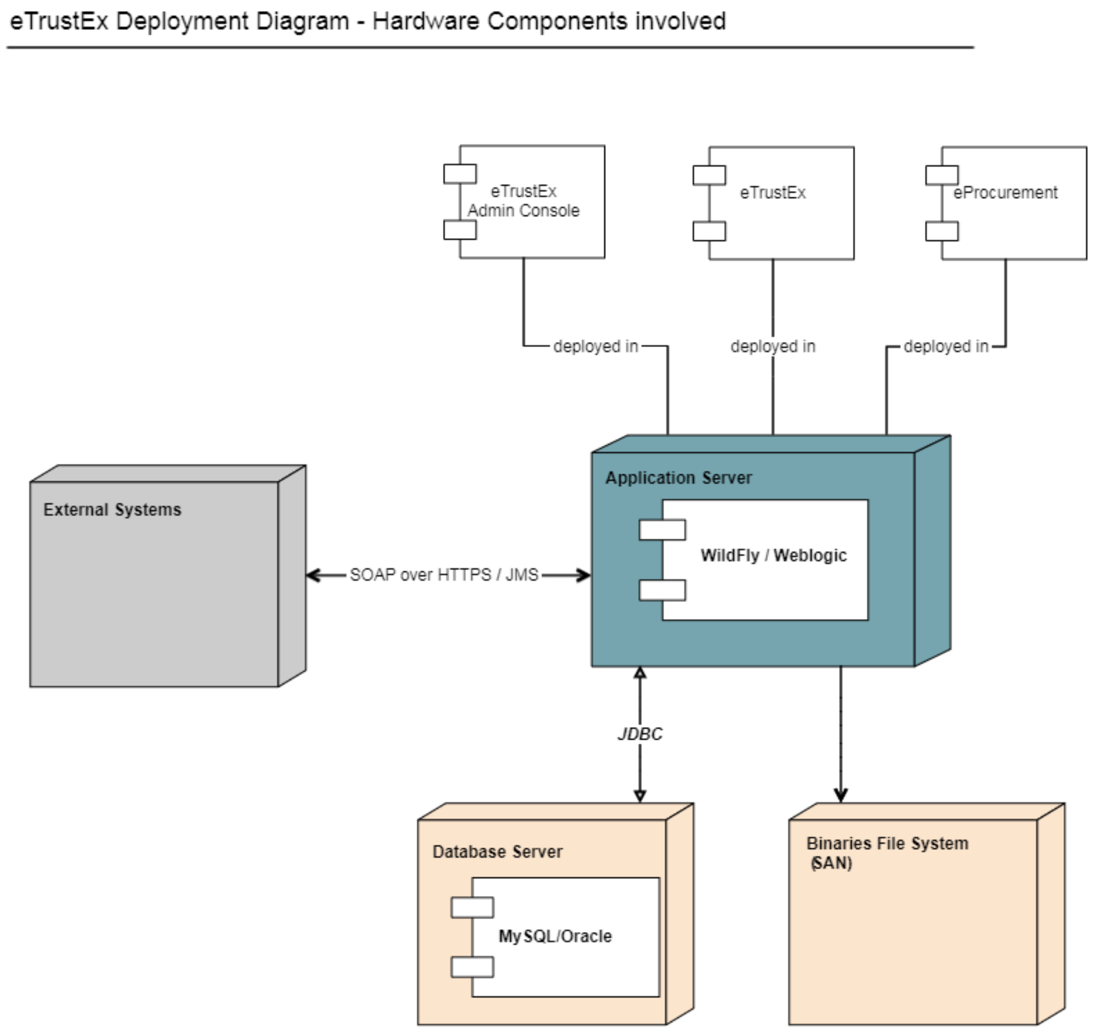
The message is then processed by the business service as described previously in this document and dispatched to the ReceiverParty messaging endpoint by the Message dispatcher. Currently the platform supports message dispatching to JMS queue or to a web service endpoint. In the case of the web-service call message dispatching, the ReceiverParty system must implement the exact same interface as the platform (submit bundle for instance). Message dispatching is also configurable in the system's database.

Finally, the ‘outbound processing for the Ack’ message handling chain builds and, optionally, signs an XML acknowledgement message that it returns to the caller.

7. DEPLOYMENT VIEW

The following is a description of the hardware nodes running the Execution Environment for the system.

The following diagram, a UML deployment diagram, provides a view of hardware components involved in this project.



It is important to note that not all physical nodes are represented on this diagram. Indeed, Database Servers and Application Servers could be duplicated for scalability and availability reasons. Furthermore, security mechanisms like firewalls are not shown.

These are the identified hardware nodes.

- External System node is a system that can connect to e-TrustEx services that are HTTP SOAP web services. External System can also submit message via JMS. As we are considering large binary files, HTTP chunking ([REF16]) and MTOM ([REF15]) are supported;
- For the Open Source version:
  - Wildfly application server 10.1.0.Final is responsible of the document management and so the document routing, validation and persistence;

    - MySql 5.5.x is responsible for the messages persistence and system configuration.
- For the internal version:
    - Oracle Weblogic application server 12c is responsible of the document management and so the document routing, validation and persistence;
    - Oracle 11g is responsible for the messages persistence and system configuration.
- The binary file system potentially shared is responsible of the storage of the large binary files.

IMPLEMENTATION VIEW

7.1. Overview

The following diagram describes the software layers of the system and their components coupled with an explanation of each element.

External systems access the platform through the web layer if they are using the web service interfaces or through the integration layer if they are using JMS. The web layer is also responsible of the authentication check. Web service endpoints are implemented using the Spring WS framework ([REF6]). This layer also holds the eTrustEx Administration Console component, a web application using Spring Security, Spring MVC and JQuery.

The integration layer uses the Spring Integration framework ([REF5]) and is responsible for the authorisation, the message validation and, more generally speaking, for the whole message processing. The components included in this layer are detailed in the chapter 6.2.3 of this document.

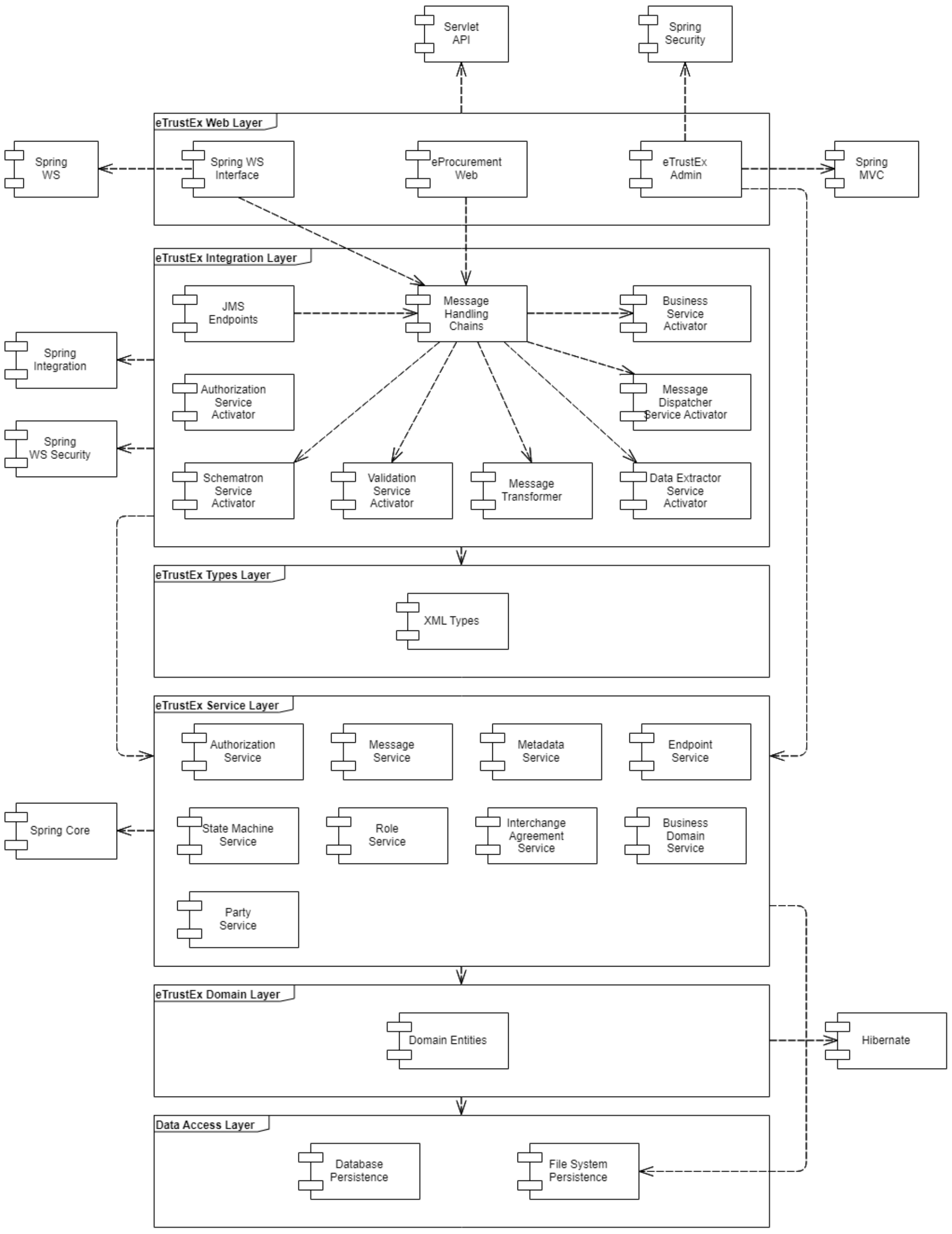
The types layer contains all the java objects generated from the XSDs and WSDLs used by the platform. These are JAX-B generated objects.

The services layer offers access to the domain objects of the platform as well as to the platform's data layer. These services are Plain Old Java Objects relying on the Spring framework ([REF4]) for dependency injection and for transaction management.

The Domain layer holds all the platform entities. The persistence of these entities is implemented using the Java Persistence API version 2.0.

Finally, the data persistence relies on a database and a file system to persist the data. The file system is used to store the large binaries and the database to persist the incoming messages and the system configuration.

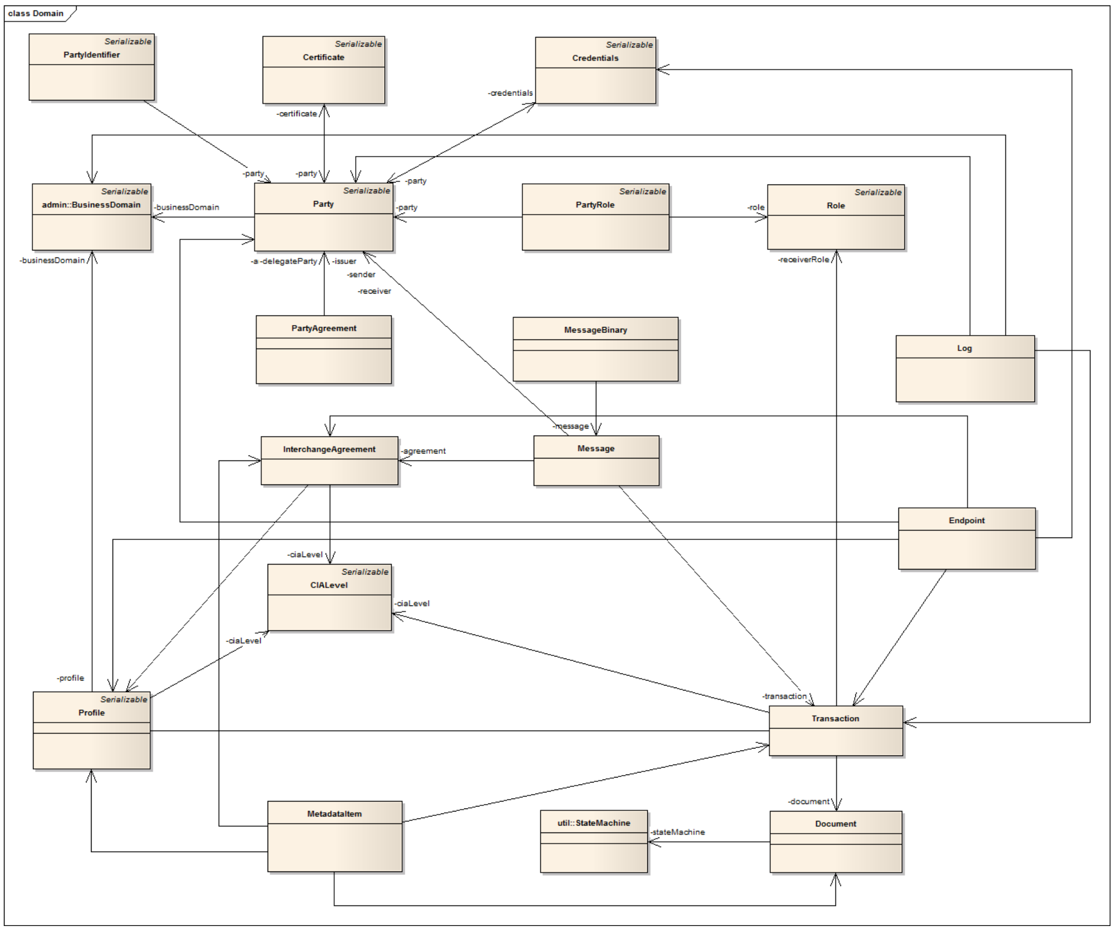
All these layers run on a J2EE application server. The platform has been tested on WildFly 10.1.0.Final and Weblogic 12c application servers.


Servlet API
Spring Security
eTrustEx Web Layer
Spring WS
Spring WS Interface
eProcurement Web
eTrustEx Admin
Spring MVC
eTrustEx Integration Layer
JMS Endpoints
Message Handling Chains
Business Service Activator
Spring Integration
Authorization Service Activator
Message Dispatcher Service Activator
Spring WS Security
Schematron Service Activator
Validation Service Activator
Message Transformer
Data Extractor Service Activator
eTrustEx Types Layer
XML Types
eTrustEx Service Layer
Authorization Service
Message Service
Metadata Service
Endpoint Service
Spring Core
State Machine Service
Role Service
Interchange Agreement Service
Business Domain Service
Party Service
eTrustEx Domain Layer
Domain Entities
Hibernate
Data Access Layer
Database Persistence
File System Persistence

8. Data View

8.1. Data Model

The following diagram shows a high-level abstraction of the data entities that must be implemented by the system:



The ***Document*** represents a certain type of data exchanged by parties. A Document is identified by its type code and may have multiple versions. It has a state (e.g. received), and may also have an associated state machine, which determines the valid state transitions based on the actions that participating parties trigger on them.

A ***Transaction*** represents a service offered by the platform, an operation on a particular document. A Transaction is defined by sender and receiver roles, and a Document which is the subject of the transaction. A Transaction may have multiple versions. The submission of a document wrapper is an example of transaction.

A ***Profile*** groups a set of transactions that can be used in the context of a business domain.

A ***BusinessDomain*** represents the specific context of a business. The parties are configured to act in the context of an existing business domain, being able to execute specific operations, which are configured at the level of the profile associated to the business domain.

A ***Party*** represents a user of the system, identified by its name. A party may have associated credential information and always acts in the context of an existing business domain. It can also represent another system needing to exchange documents through the platform.

PartyIdentifier holds additional identity information about parties, such as an identifier and its scheme (e.g. Global Location Number, VAT Number).

The credentials specify credentials data, such as username and password. They are used by a party to connect to the platform. It also holds information determining whether a party must sign the messages it sends to the system or not.

Certificate: Stores certificate ([REF14]) data for encryption purposes.

The ***Role*** Entity defines roles available in the system. E.g. Customer, Supplier, BundleExchanger.

PartyRole: One Party can fulfil different roles depending on which types of messages are being exchanged. Message exchange transactions are based on a specified sender and receiver role. One Party can have more than one role, and these roles can be used independently.

PartyAgreement: Based on an agreement between parties, a party can have one or more delegates, who are allowed to act on behalf of the delegating party generically or on specific transactions. This relationship between parties is specified in the PartyAgreement.

The ***InterchangeAgreement*** specifies an interchange agreement between parties. An interchange agreement has participating parties, and each of them is playing a certain role. An interchange agreement may have a start date from which it is valid. The interchange agreement models the document exchange contract between parties. Parties agree on a set of business transactions, represented by a ***profile,*** that reflects the business they are doing together.

CIALevel specifies a Confidentiality, Integrity and Availability levels tuple. A CIALevel can be associated to an Interchange Agreement, Profile or Transaction.

A ***Message*** is an element of conversation between parties. It represents the information sent from a sender party to a receiver party. A message has a status code, and associated transaction.

A ***MessageBinary*** either contains the raw XML message that is stored by the system, either is a placeholder which refers to a file in the file store.

StateMachine defines state machines for Documents, according to the SCXML specification. A StateMachine defines valid states and transitions for a Document.

8.2. State Machines

8.2.1. Introduction

Each object of the system (including the Bundles) will have its own state machine. A state machine is always initiated by a Sender. It starts with a specific message (e.g. "The Sender sends a Bundle").

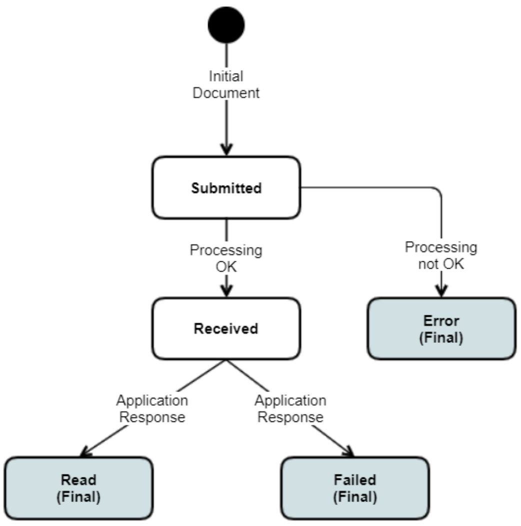
Only a message will be capable of passing an object from a state to the next one. A state machine will never represent the workflow of the object in the receiver back-office. While the object is in the receiver's application, the state of the object remains the same in e-TrustEx, until the back-office notifies (via an Application Response) that the workflow is complete.

The following sections illustrate the state machines which have to be supported by e-TrustEx.

Note: Once a state machine instance is started, each state transition is performed by the transmission of a message. Each message is subject to logging and monitoring.

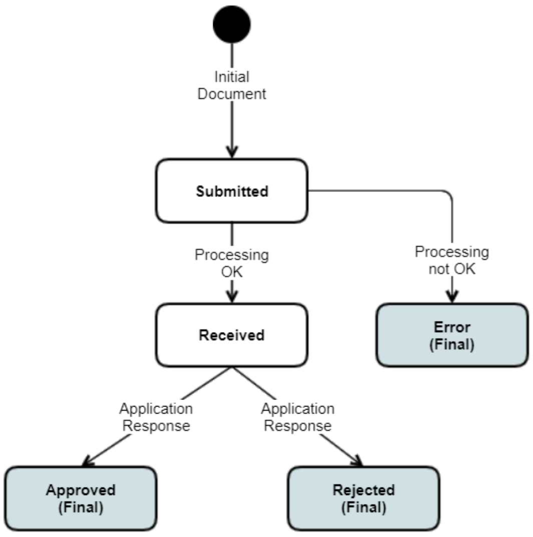
8.2.2. <u>Generic State Machines</u>

The document bundle, which is the generic service for exchanging message, has a specific state machine.



8.2.3. <u>The state machine for eProcurement services</u>

Most of the incoming eProcurement messages will have the following state machine:



9. Size and Performance

9.1. Size

Size restrictions, not on the application or its components themselves, but on the data that is being processed by e-TrustEx, have an impact on the architecture and configuration of the system.

To support the exchange of large binary files, MTOM ([REF15]) may be used to avoid Base 64 encoding that leads to up to 30 % of size overhead.

To be able to control the incoming streams, HTTP chunking ([REF16]) is used. The binary stream is read chunk by chunk, and, if the file size exceeds a given limit, the transmission is automatically ended.

That way of working also avoids loading the entire file into memory. The binary file chunks are stored in the e-TrustEx file system and kept during a given period (retention policy).

The implementers must determine the file system size, the database size and the maximum size of document wrapper according to their needs. By default, the maximum size for large binaries is 500 MB.

Extra restrictions can be implemented via the SLA (Service Level Agreement) policies. These restrictions concern the maximum size of a document wrapper, the maximum number of wrappers in a bundle and the maximum volume of data that can be sent by a party in a given period (day, week or month).

9.2. Performance

An important architectural decision that benefits the performance of e-TrustEx includes the decoupling of the solution into a synchronous and an asynchronous mode of communication. The synchronous mode of communication is used by services in which the business response is immediate, and which requires limited processing. The asynchronous mode however is aimed at services whose processing involves a series of long-running business actions which often require workflow steps performed by the back-office. In this mode, an incoming message will be stored in a queue where it waits to be read for further processing. If required, the number of asynchronous processing threads can be modified both in the number of total threads running and in the number of threads for a specific service (e.g. Submit Bundle). This type of configuration is specific to each Java EE server.

10. Quality

The architecture of e-TrustEx contributes to quality aspects of extensibility, reliability and portability in the following ways.

10.1. Extensibility

As indicated in “6.2 Architecturally Significant Design Packages”, External System node is a system that can connect to e-TrustEx services that are HTTP SOAP web services. External System can also submit message via JMS. As we are considering large binary files, HTTP chunking ([REF16]) and MTOM ([REF15]) are supported.

As shown in "0 Implementation View", e-TrustEx is designed in a layered fashion and consists of multiple interconnected modules. This modular design facilitates upgrades by replacing existing modules and extensions by adding additional modules.

10.2. Reliability

The reliability of e-TrustEx is enhanced through the decoupling of each architectural layer by JMS queues. A store and forward mechanism and automatic retry policy ensures that parts of the system can continue functioning without losing data when an issue occurs in a specific component.

10.3. Portability

The application can be deployed on WildFly Application Server ([REF8]) and can connect to MySQL database, as well as on Weblogic Application Server and connect to Oracle database.

As well as being extensible, e-TrustEx is carefully designed in such a way that it is independent of the specific external system that it is serving. The use of generic HTTP SOAP web services leaves the different layers unaffected when an additional external system needs to be supported by e-TrustEx.

The usage of JPA to access the database makes it easy for implementers to change the relational database used to store the platform data.