



European eInvoicing Standard in Italy

|  |  |
| --- | --- |
| **Project number** | INEA/CEF/ICT/A2017/1560867 2017-IT-IA-0150 |
| **Project acronym** | EeISI |
| **Project title** | European eInvoicing Standard in Italy |
| **Starting date** | 1 May 2018 |
| **Ending date** | 30 June 2019 |
| **Programme** | Connecting Europe Facility (CEF)  CEF-TC-2017-3: eInvoicing |

**Deliverable D3.2**

**eInvoice system upgrade design of technical specification**

|  |  |
| --- | --- |
| **Related WP** | WP3 - Design |
| **Deliverable number** | 3.2 |
| **Due date** | 23/11/2018 |
| **Revision date** | 25/11/2018 |
| **Actual date** | 30/11/2018 |

**Deliverable Info**

|  |  |
| --- | --- |
| **Responsible organisation** | Infocert |
| **Editor** | Infocert |
| **Contributors** | Infocert |
| **Reviewers** | Infocert |
|  |  |
| **Abstract** | **This document describes the technical specification and high-level Architecture for the eInvoicing mapper framework** and **data entry component applied on eInvoice mapping software stream of EeISI – European eInvoicing Standard in Italy.**  **This is how the project fits into the existing landscape and existing processes and standards.** |
| **Keywords** | Architecture, Design |
|  |  |
| **Acknowledgement** | This work was partially supported by the European Commission (EC) through the Connecting Europe Facility (CEF) programme under project EeISI (grant agreement no. INEA/CEF/ICT/A2017/1560867 2017-IT-IA-0150) |
| **Disclaimer** | The sole responsibility of this publication lies with the author(s). The European Union is not responsible for any use that may be made of the information contained therein. |
| **Confidentiality** | The information in this document is confidential and restricted only to the members of the EeISI consortium  (including the Commission Services). |
|  |  |
| **Note** | *-* |
|  |  |

**Change Log**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Organisation** | **Description** |
| 0.1 | 22/11/2018 |  | InfoCert | First draft |
| 1.0 | 04/12/2018 |  | InfoCert | Data entry component |
| 1.1 | 06/12/2018 |  | InfoCert | Revision |
| 2.0 | 28/12/2018 |  | InfoCert | First release |
| 2.1 | 31/01/2019 |  | InfoCert | Update API mapper description |
| 2.1.1 | 31/03/2020 | Roberto Reale | AGID | Technical review and quality assessment |

# Table of contents

[Table of contents 4](#_Toc533759733)

[List of tables 4](#_Toc533759734)

[List of figures 5](#_Toc533759735)

[1. Introduction 6](#_Toc533759736)

[1.1. Purpose 6](#_Toc533759737)

[1.2. Scope 7](#_Toc533759738)

[1.3. Overview 8](#_Toc533759739)

[1.4. Definitions, Acronyms and Abbreviations 9](#_Toc533759740)

[1.5. References 10](#_Toc533759741)

[2. Data Entry component 11](#_Toc533759742)

[2.1. Constraints and Assumptions 11](#_Toc533759743)

[2.2. Functional Requirements 12](#_Toc533759744)

[2.3. High level requirements 12](#_Toc533759745)

[2.4. Non-Functional Requirements 14](#_Toc533759746)

[2.5. Architecture views 15](#_Toc533759747)

[2.5.1. Components catalogue 15](#_Toc533759748)

[2.5.2. Diagrams 16](#_Toc533759749)

[2.5.3. Technology 17](#_Toc533759750)

[2.6. Architectural Decision 18](#_Toc533759751)

[3. Mapper 19](#_Toc533759752)

[3.1. Constraints and Assumptions 19](#_Toc533759753)

[3.2. Definitions 19](#_Toc533759754)

[3.3. Mapping Architecture 20](#_Toc533759755)

[3.4. Transformation 22](#_Toc533759756)

[3.4.1. XML Validation 22](#_Toc533759757)

[3.4.2. Local Validation using CEN 23](#_Toc533759758)

[3.4.3. Peppol BIS 3 Validation 23](#_Toc533759759)

[3.4.4. XML CEN Validation 23](#_Toc533759760)

[3.4.5. The EeISI Framework 23](#_Toc533759761)

# List of tables

[Table 2 Data entry module: high level requirements 13](#_Toc533759762)

# List of figures

[Figure 1: High level application scheme 16](https://infocertspa-my.sharepoint.com/personal/yyi9655_infocert_it/Documents/einvoicing/EeISI/WP3%20Design/EeISI_D3.2_eInvoice%20system%20upgrade%20design%20of%20technical%20specificationRel.2.0.docx#_Toc533759781)

[Figure 2 A mapping process is in general made up by three steps. 19](#_Toc533759782)

[Figure 3: validation phase diagram with additional entry points 20](#_Toc533759783)

[Figure 4: high level schema of mapper conversions. 21](#_Toc533759784)

1. Introduction
   1. Purpose

The purpose of this document is to describe the technical specifications and high-level Architecture for the eInvoicing mapper framework and additional data entry component applied on eInvoice mapping software stream of EeISI – European eInvoicing Standard in Italy.

This is how the project fits into the existing landscape and existing processes and standards.

This document is jointly written by the Domain Architect, Solution Architect and Solution Manager, each working on his view on the same matter which is functionally and technically driven.

In the end, all the three parties have to agree on the content.

This document forms the base of subsequent phases of the project like Implementation, Platform Validation and Valorization.

The intended audiences for this document are:

* Solution Architect
* Domain Manager/Sub-Domain Manager
* Project/Program Manager
* Stakeholders of the project

* 1. Scope

Definition of a software platform that will provide a set of services related to the cross transformations of electronic invoices, according to these guidelines:

* Usage of open source technologies.
* Different application layers between core methods and external interfaces.
* Modularity in the internal design.
* Extendibility guaranteed by components packaged as plugins and deployed as separate modules.
* Clear logging.
* External configuration that guides the application behavior and composition.
* Integrability with interfaces in standard technologies that allow an easy discovery of the exposed methods.
* Small footprint on hardware resources.
* High scalability due to a multithreaded management of the requests.

The document describes two main components of eInvoice mapper framework:

* a "mapping" component (in this document named “Mapper”) that allows to convert invoices from various formats according to CEN TS 16931-2 defined syntaxes (namely UBL and CII) to FatturaPA format. A first version of the Mapper has already been realized in the previous eIGOR project (eInvoicing GO Regional) on the basis of draft EN 16931-1 and syntax binding technical specifications (CEN TS 16931-3). Within EeISI project the main objective is to consolidate the mapper according to published version of the norm and technical specifications and to include the Italian CIUS rules. In addition, the mapper is enriched with new entry points (intermediate CEN meta language) and a mapping to Peppol BIS 3 profile which has been published in 2018 spring.
* a “data entry” component that allows interfacing the mapper to manually upload or update a file an invoice that meets the expected format.

In particular the data entry component will have to:

* Create a CEN based invoice or a CEN based invoice compliant with Italian CIUS
* Import an xml file (full or partial invoice expressed in XMLPA or UBL 2.1 or CII or CEN BTs-values)
* Provide information for each BT about associated CEN business rules and Italian business rules
* Export the CEN based invoice to an xml file (XMLPA or UBL 2.1 or CII or CEN BTs-values)
* Validate the CEN based invoice on the basis of a specific schematron (national CIUS, community CIUS,…)

* 1. Overview

This document is organized as follows:

Chapter 2 concerns the Data Entry parts of the eInvoicing System

• Constraints and Assumption: this chapter presents the relevant assumptions and constrains affecting the architecture

• Requirements: this chapter describes the functional and non-functional requirements reported in the requirement documentation

• Domain Blueprint Representation: this chapter illustrates the project contribution to the target domain blueprint

• Architecture Views: this chapter contains the architectural views describing the proposed architecture

• Technology: this chapter explains the architectural decisions influencing the target architecture presented in the previous chapters.

• Architectural Decisions: this chapter explains the architectural decisions influencing the target architecture presented in the previous chapters

Chapter 3 concerns the Mapper parts of the eInvoicing Systems

• Mappings overview

• Transformation

• Details about UBL, CEN and FatturaPA transformation

* 1. Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| **B2B** | Business to Business |
| **B2C** | Business to Consumer/Citizen |
| **B2G** | Business to Government |
| **BII** | Business Interoperability Interfaces |
| **C2G** | Citizen to Government |
| **CCTS** | Core Component Technical Specification |
| **CEF** | Connecting Europe Facility |
| **CEM** | Certified Electronic Mail – Legal Mail (PEC Posta Elettronica Certificata in Italy) |
| **CEN** | European Committee for Standardisation |
| **CII** | Cross Industry electronic Invoice |
| **CIUS** | Core Invoice Usage Specification |
| **DSI** | Digital Service Infrastructures |
| **EDIFACT** | Electronic Data Interchange For Administration, Commerce and Transport |
| **EMSFEI** | European Multi-Stakeholder Forum on eInvoicing |
| **e-SENS** | Electronic Simple European Networked Services |
| **FatturaPA** | Public administration electronic invoice framework (FatturaPubblica Amministrazione) |
| **G2G** | Government to Government |
| **INEA** | Innovation and Networks Executive Agency |
| **OASIS** | Organization for the Advancement of Structured Information Standards |
| **PEPPOL** | Pan-European Public Procurement Online |
| **PEPPOL-BIS** | Pan-European Public Procurement Online Business Interoperability Specifications |
| **SDI** | Electronic exchange system in Italy (Sistema Di Interscambio) |
| **UBL** | Universal Business Language |
| **UN/CEFACT** | United Nations Centre for Trade Facilitation and Electronic Business |
| **UNTDID** | UN Trade Data Interchange Directory |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Location |
| **URN** | Uniform Resource Name |
| **XML** | Extensible Mark-up Language |

* 1. References

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application.

* EN 16931-1:2017 Electronic invoicing - Part 1: Semantic data model of the core elements of an electronic invoice
* CEN/TS 16931-2:2017 Electronic invoicing - Part 2: List of syntaxes that comply with EN 16931-1
* CEN/TS 16931-3-1:2017 Electronic invoicing - Part 3 - 1: Syntax bindings of the core elements of an electronic invoice - Syntax binding methodology
* CEN/TS 16931-3-2:2017 Electronic invoicing - Part 3 - 2: Syntax bindings of the core elements of an electronic invoice - Binding to ISO/IEC 19845 (UBL 2.1)
* CEN/TS 16931-3-3:2017 Electronic invoicing - Part 3 - 3: Syntax bindings of the core elements of an electronic invoice - Binding to UN/CEFACT XML
* CEN/TS 16931-3-4:2017 Electronic invoicing - Part 3 - 4: Syntax bindings of the core elements of an electronic invoice - Binding to ISO/IEC 9735 (UN/EDIFACT)
* ISO 3166 1, Codes for the representation of names of countries and their subdivisions — Part 1: Country codes
* ISO 4217, Codes for the representation of currencies
* ISO 639 2, Codes for the representation of names of languages
* ISO 8601, Data elements and interchange formats — Information interchange — Representation of dates and times
* ISO 15000-5, Electronic Business Extensible Markup Language (ebXML) — Part 5: Core Components Specification (CCS)
* ISO 6523, Information technology — Structure for the identification of organizations and organization parts
* ISO/IEC 19845, Information technology -- Universal business language version 2.1 (UBL v2.1)

Moreover the following Italian documentation is referenced in this deliverable:

* Schema del file xml FatturaPA versione 1.2 - xsd
* Specifiche tecniche del formato della FatturaPA versione 1.2.1- pdf
* Rappresentazione tabellare del tracciato FatturaPA versione 1.2.1- pdf
* Rappresentazione tabellare del tracciato FatturaPA versione 1.2.1- excel
* Foglio di stile per la visualizzazione della FatturaPA versione 1.2.1 - xslt
* generica Foglio di stile per la visualizzazione della Fattura Ordinaria versione 1.2.1 - xslt
* Elenco modifiche al tracciato FatturaPA - pdf
* Suggerimenti per la compilazione della FatturaPA versione 1.5

1. Data Entry component
   1. Constraints and Assumptions

Here are detailed some considerations regarding the major technical and functional constraints that have some impact on the software design.

|  |  |  |
| --- | --- | --- |
| Constraint | Description | |
| Architectural Principles | | Infrastructure independent:   * Possibility to deploy the system in on-premise & cloud mode, to respond to heterogeneous technical and compliance requirements   Modernization:   * Design the development following the main guidelines on technological modernization and oriented to microservices and their communication |
| Timeline | | 31/01/2019 |
| Existing Landscape | | The project involves the development of the web application, based on the use of a java library called Mapper, in a simple and usable way for programming, the following functions:   * Bidirectional conversion and validation between the following xml formats: CEN, CIUS-IT <-> XMLPA, UBL 2.1, CII |
| Geographical constraints | | Invoice validation and transformations processes are based CEN and CIUS-IT regulations.  Additional invoice validation according to specific standards (national CIUS, sectoral CIUS, ...) |
| Risk Willingness | |  |
| Complexity | | High:  • The software has to be designed infrastructure independent.  • Current legislation and regulation compliance in both Europe and Italy requires complex implementation rules.  • The software has to be entirely designed |

* 1. Functional Requirements

The data entry module consists of an additional component to the eInvoicing mapper framework. This module can be used separately or integrated into the framework. The main goal is to provide a tool to create and validate a CEN compliant invoice. The tool is built with a CEN centered approach taking into consideration the core invoice semantic model described in EN 16931-1:2017.

The main functionalities offered to the final users are:

1. Create a CEN based invoice or a CEN based invoice compliant with Italian CIUS
2. Import an xml file (full or partial invoice expressed in XMLPA or UBL 2.1 or CII or CEN BTs-values)
3. Provide information for each BT about associated CEN business rules and Italian business rules
4. Export the CEN based invoice to an xml file (XMLPA or UBL 2.1 or CII or CEN BTs-values)
5. Validate the CEN based invoice on the basis of a specific schematron (national CIUS, community CIUS,…)
   1. High level requirements

| HLR-ID | Req name | Req description |
| --- | --- | --- |
| HLR-001 | CEN-centric | The module shall enable the creation of a CEN based invoice (built on the core invoice semantic model described by EN-16931-1:2017) |
| HLR-002 | Italian customization | The module shall enable the creation of a CEN base invoice fully compliant with Italian CIUS |
| HLR-003 | Import | The module shall have an import functionality of partial or full invoice expressed in xml (XMLPA, UBL 2.1, CII, CEN BTs-values) |
| HLR-004 | CEN information | The module shall have an information tool which provides all information related to a BT (name, description, related business rules, data values admitted, Italian CIUS restrictions,…) |
| HLR-005 | CEN validation | The module should integrate automatic validation mechanisms on the basis of CEN business rules and Italian CIUS restrictions or Italian mapping rules |
| HLR-006 | Export | The module shall have an export functionality of partial or full invoice expressed in xml (XMLPA, UBL 2.1, CII, CEN BTs-values) |
| HLR-007 | Additional validation | The module should provide a mechanism to additionally validate the full CEN compliant invoices against validation artefacts (eg. National CIUS, community CIUS,…) |
| HLR-008 | Categories | The module shall have the information grouped by identified categories (seller, header, buyer, data, vat and totals) |
| HLR-009 | Mandatory | The module shall facilitate the user identifying the mandatory information needed for a CEN compliant einvoice or an Italian CIUS compliant invoice |
| HLR-010 | Optional | The module should provide a mechanism to select the optional information required in an invoice |
| HLR-011 | Progress status | The module should provide a mechanism to clearly show the progress status of invoice preparation |
| HLR-012 | Localization | The module should provide a localization in both languages English and Italian |
| HLR-013 | Users | The module does not need a specific users management (login, profile, …) |

Table 2 Data entry module: high level requirements

* 1. Non-Functional Requirements

The application functionalities must be implemented accordingly to current Infocert design guidelines and must possibly follow:

* Creation of dynamic rules engine easily configurable with new BT (xsd).
* Digital Modernization, that is expressed in:
  + Modernization (tools / compiler / libraries)
  + Predisposition of the project to the new CI / CD methodologies
  + Fluidity of the web layout and extensive web-browser compatibility
* Decoupling engine API & web API.
* Multilingual (Web / Doc)

Consequently:

* Simplicity in adding / modifying BT
* Integration of the system into functionally more substantial services
* Ease in future developments of new features
* Flexible deploy on-premises e public cloud
  1. Architecture views

This is a high level overview of all the involved components and the relationship between them.

We have divided this section in:

* Components catalogue. In this section we list all the components that are part of the solution.
* Diagrams. We have drawn the application landscape diagram
  + 1. Components catalogue

**Logical components**

Microservices API:

The API microservices component forms the wrapper of the Mapper. This component allows the Mapper to dialogue with external entities, in the case described, allows dialogue with the web.

This component is a software infrastructure developed according to the microservice standard:

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.

The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack. In a microservices architecture, services are [fine-grained](https://en.wikipedia.org/wiki/Service_granularity_principle) and the [protocols](https://en.wikipedia.org/wiki/Protocol_(computing)) are lightweight. The benefit of decomposing an application into different smaller services is that it improves [modularity](https://en.wikipedia.org/wiki/Modular_programming). This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.

This layer of microservices also makes it possible to make the use of the mapper independent from the infrastructure, allowing greater flexibility

Web Application:

Use the API Microservices component to allow access to the functions of the mapper.

This component allows a user to access the mapper's functions via the web.

**Detailed components**:

* Engine: API REST Bidirectional conversion CEN <-> XMLPA,UBL 2.1,CII
* Engine: API REST Bidirectional conversion CIUS-IT <-> XMLPA,UBL 2.1,CII
* Engine: API REST Validation (Schematron predefined)
* FrontEnd: Fluid Layout and Dynamic Process Flow
* FrontEnd: Making CEN Compliant Invoce
* FrontEnd: Making CIUS-IT Compliant Invoce
* Full integration API REST - FrontEnd
  + 1. Diagrams

Below the high level application scheme.

In this diagram it is highlighted that the Microservice API layer is part of the Mapper: the web application exposes and makes available the functions provided through the microservices.

Figure 1: High level application scheme

Web Application

Microservice API

Jar Mapper

Mapper

* + 1. Technology

Below we list the technologies / tools used broken down by component:

MICROSERVICE API

* Maven
* Java Spark

FRONTEND

* NodeJs
* ReactJS
* Bootstrap

TESTING TOOLS

* JUnit
* Katalon Studio
* Apache JMeter
  1. Architectural Decision

The following issues were analyzed and solved in order to provide a consistent software design

|  |  |
| --- | --- |
| Title | Decision |
| Implementation environment | As requested by the Agreement, the EeISI DataEntry will be implemented using only opens source technologies; the software libraries used by the application will follow this approach. |
| Compatibility with pre-existing Mapper software | The EeISI DataEntry application environment is running on JDK 8; the component will use version 1.8 as source target of the java build. |
| Data persistency | The software will not save any kind of transient information in a database. The requests will be managed in a synchronous way with no need of a data persistency.  The reports will be generated at runtime. |
| Configuration strategy | All the properties of the software are filesystem based; the local path can be added as a Property object. |
| Integrability | The API layer is made of public REST methods related to the high-level functionalities (create,validate, transform). |
| Log management | The logging framework SLF4J will be used by the application. |
| Microservices | The architecture was designed following the main principles of microservices, decoupling the API REST tier from the frontend web console. |

1. Mapper
   1. Constraints and Assumptions

Details and considerations regarding the mapping model follow here below.

|  |  |  |
| --- | --- | --- |
| Constraint / Assumption | Description | |
| Principles | | To use standard and well-established methodologies in order to:  - reduce the effort needed to evolve the system;  - make the mapping easy to understand and interpret. |
| Geographical constraints | | Invoice validation and transformations processes are based on Italian (FatturaPA), European (CEN, UBL and CII protocols, XML CEN) and custom (Peppol BIS 3) regulations. |
| Complexity | | High:  • Considering the number of “fields” in the different invoice formats and the somewhat different meaning that each field has in each format, the possible number of mappings explodes.  • It should be possible to extend the mapping that will be provided with new custom mapping. |
| Mapping invoices between different legal systems. | | CEN  CIUS: a way to make validation rules more restricted than the ones defined at European level without violating general principles.  UBL 2.1  CII  Peppol BIS 3 format |

* 1. Definitions

In eEISI, mapping is the macro process that, starting with an invoice, allows to obtain another invoice expressed in a different format, yet semantically equivalent.

Mapping can be considered split in three main logical function.

* **Input validation**: ensure that the incoming invoice obey a set of defined constraints. This layer discards the invoice if it does not adhere to the desired constraints.
* **Transformation**: the same semantic information contained in the incoming invoice is transferred to a new invoice, generally described with another format.
* **Output validation**: ensure that the outgoing invoice obey a set of defined constraints or discard the invoice otherwise.



Figure 2 A mapping process is in general made up by three steps.

The validation phase of the inputs can be logically divided into different steps:

* Syntactic validation, using XSD, described in paragraph 3.4.1
* Semantic validation, using Schematron, described in paragraph 3.4.3
* Country Validation (convertible to an invoice responding to a CIUS country localization) described in the paragraph 3.4.5

In the EeISI project, entry points were provided for each of the validations described above. Through these entry points it is possible to obtain validated documents with respect to each of the validations foreseen.

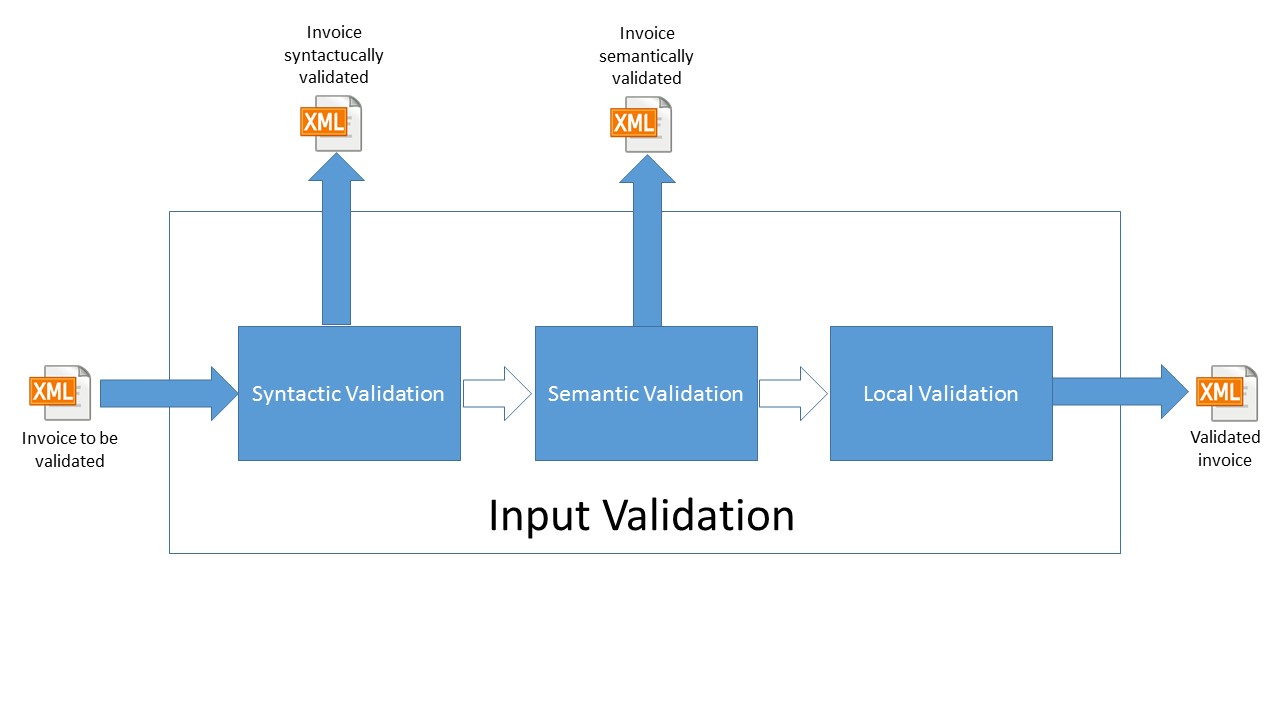


Figure 3: validation phase diagram with additional entry points

Through these entry points it will be possible, for example, to obtain a validated file only with respect to the semantics or only with respect to the syntax of the XML

* 1. Mapping Architecture

The European norm defines all the validation and meaning of information in an invoice related to the “CEN” model. This is a purely semantic model and a well accepted or standard format to represent such model does not exist.

All invoices must processed comply with the CEN format which identifies the semantic standard for European invoices.

Managing the direct passage from one format to another for all the formats to be managed was unachievable. For this reason we chose to use the CEN format as a model of intermediation from which to convert invoices from one format to another (a CEN centric approach).

A further advantage is the implementation of an XML CEN format that fully complies with the semantic CEN invoice model using the XML meta-language. The Mapper component allows to obtain a CEN XML file from any supported syntax, and can convert a cen XML file to any supported syntax.

The conversion from one format to another is then carried out by passing from the semantic CEN model, the formats from which it is possible to start and to arrive are:

* UBL
* Peppol BIS 3
* CII
* Fattura PA
* XML CEN

The conversion from UBL and Peppol BIS 3 will be univocal while the conversion to Peppol BIS 3 will be differentiated as Peppol BIS 3 is a CIUS.

Below is a high-level diagram of the mapping architecture.

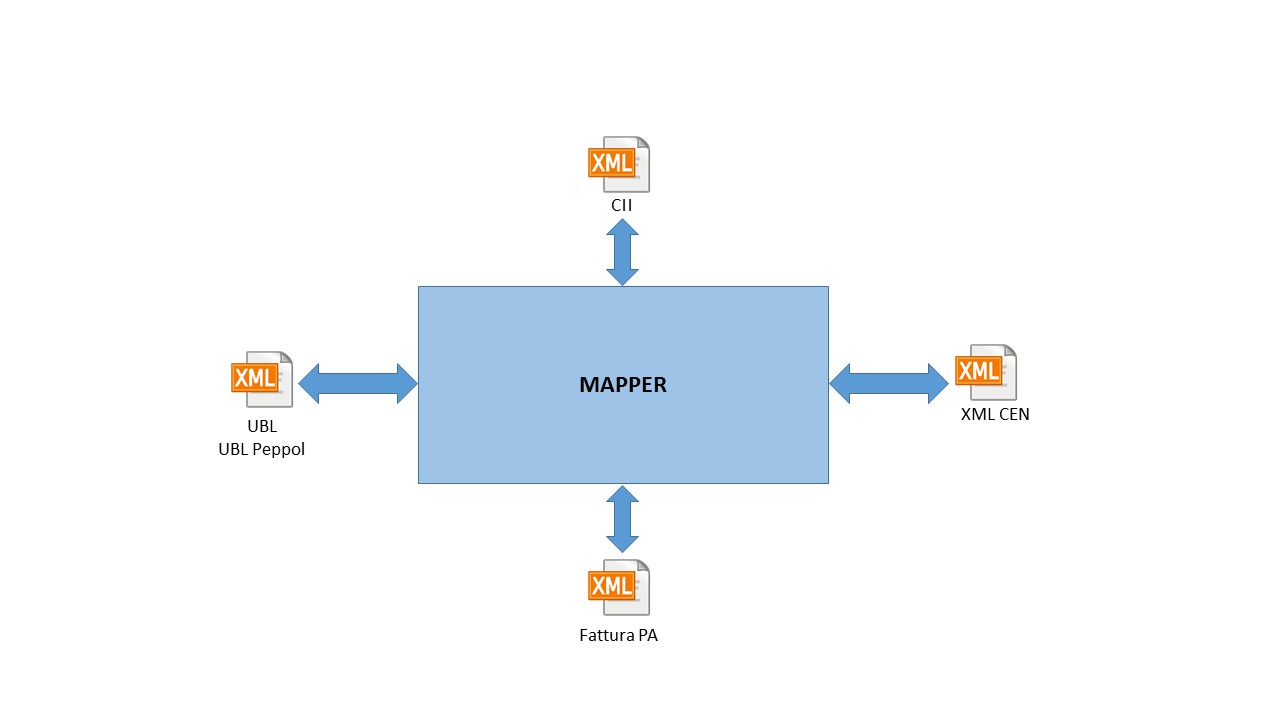


Figure 4: high level schema of mapper conversions.

* 1. Transformation
     1. XML Validation

Each transformation between two formats could apply a validation. In case the format to be validated is XML, two main validation models could be used: XML Schema and Schematron.

**XML Schema**

From Wikipedia:

XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) document. It can be used by programmers to verify each piece of item content in a document. They can check if it adheres to the description of the element it is placed in.[1]

Like all XML schema languages, XSD can be used to express a set of rules to which an XML document must conform in order to be considered "valid" according to that schema. However, unlike most other schema languages, XSD was also designed with the intent that determination of a document's validity would produce a collection of information adhering to specific data types. Such a post-validation infoset can be useful in the development of XML document processing software.

XML Schema Definition is the ubiquitous solution to validate XML content. Almost all formats natively supported by EeISI come with their related schema. XML Schema validation is better at guarantee that the structure of an XML file is as expected, i.e. that elements appear in a given order, that an element has the expected attributes and so on.

**Schematron**

From Wikipedia:

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.

Schematron is capable of expressing constraints in ways that other XML schema languages like XML Schema and DTD cannot. For example, it can require that the content of an element be controlled by one of its siblings. Or it can request or require that the root element, regardless of what element that is, must have specific attributes. Schematron can also specify required relationships between multiple XML files.

Constraints and content rules may be associated with "plain-English" validation error messages, allowing translation of numeric Schematron error codes into meaningful user error messages.

The current ISO recommendation is Information technology, Document Schema Definition Languages (DSDL), Part 3: Rule-based validation, Schematron (ISO/IEC 19757-3:2016).

Schematron allows to quickly and precisely understand how a piece of info in an XML is checked. Its main focus is to guarantee that the values of element and attributes in an XML files obey some given rules.

* + 1. Local Validation using CEN

CEN is a purely semantic model, its validation rules are well-defined in the norm and implemented to validate Fattura PA format.

CEN comes with a set of well defined rules that allows to validate a CEN model.

* A set of “cardinality” rules that specify how many elements of one type can be contained in the container type.
* A set of “business” rules that defines constraints that the values represented in CEN should obey.
  + 1. Peppol BIS 3 Validation

CEN is a purely semantic model, its validation rules are well-defined in the norm and implemented to validate a Peppol BIS 3 document (invoice or credit note).

* + 1. XML CEN Validation

CEN is a purely semantic model, its validation rules are well-defined in the norm and implemented to validate an XLM CEN document (invoice or credit note)

* + 1. The EeISI Framework

**I/O Interfaces**

The EeISI framework provides a clean and simple API that allows applications to quickly integrate it in their workflow and start converting invoices.

The interfaces developed for the eIGOR project are also maintained for EeISI.

For the EeISI project, compared to the previous eIGOR, two new interfaces have been implemented that allow us to obtain validation with respect to validated versus semantic and syntactic controls.

The two new method implemented are validateSemantic and validateSyntax that now add to the already existing validate method. Let’s check them out.

**Conversion**

When a full conversion between two different formats is needed, one should call the conversion() method.

public ConversionResult<byte[]> convert(final String sourceFormat, final String targetFormat, final InputStream invoice)

This method converts the provided invoice assuming it is expresses in the given source format. The result contains the converted invoice, if the conversion process was able to complete the conversion,

and any error or warning that may be occurred during the process.

sourceFormat is the format of the invoice to be converted.

targetFormat is the format the invoice will be converted to.

invoice is the invoice to convert.

If it is required to have a more detailed feedback on the conversion, it is possible to register some callbacks that are invoked when, during the conversion, noteworthy events occur.

**Validation**

eEisi gives the option to just validate a document. A document is considered valid when it would be converted without issues if fed to the conversion method.

eEisi proved three different validation method.

public ConversionResult<Void> validate(final String sourceFormat, final InputStream invoice)

public ConversionResult<Void> validateSyntax(final String sourceFormat, final InputStream invoice)

public ConversionResult<Void> validateSemantic(final String sourceFormat, final InputStream invoice)

The first one fully validate a given document. A document that is valid according to this method can be transformed in the specified target format without issues. This method usually applies an XSD schema to guarantee the document is a valid document, a schematron to check whether it complies with the CEN model and finally, if needed, a CIUS schematron to check whether it matches the Italian constraints.

The other two methods work exactly like the validate method, already existing and implemented for the eIGOR project. The input parameters are therefore the format of the document to be validated (String) and the document itself (InputStream). The ConversionResult object is provided as output, providing information about the result of the conversion.

ValidateSynatx validates just the syntax of the provided document, usually applying the XSD, while validateSemantic applies also the schematron to check the compliance with the CEN model.

**Custom Validation**

Two methods are provided to execute custom validation checks based on schematron and XSDs.

customSchSchematronValidation(File schemaFile, InputStream xmlToValidate)

This method accepts a file descriptor referencing a schematron file and applies it to the given XML.

customXsdValidation(File schemaFile, InputStream xmlToValidate)

This method accepts a file descriptor referencing an XSD file and applies it to the given XML.