ECP 2006 DILI 510049

ENRICH

Report on the Documentation and Use of the ENRICH Garage Engine

Deliverable number	*D-3.4*
Dissemination level	*Public*
Delivery date	*30 October 2009*
Status	*Draft*
Author(s)	*James Cummings, Tomasz Parkoła, Mariusz Stanisławczyk and Marcin Werla*

e*Content*plus

This project is funded under the *e*Content*plus* programme[1], a multiannual Community programme to make digital content in Europe more accessible, usable and exploitable.

1 OJ L 79, 24.3.2005, p. 1.

Document Version Control

Version	Date	Change Made (and if appropriate reason for change)	Initials of Commentator(s) or Author(s)
0.0	25 Oct 09	Draft Deliverable	JC,OUCS and PSNC

Document Review

Reviewer	Institution	Date and result of the review

Approved By (signature)	**Date**

Accepted by at European Commission (signature)	**Date**

1 Executive Summary

The ENRICH Garage Engine (EGE) is a set of web tools for the conversion and validation of documents. It has both a web frontend and a RESTful web service API. This deliverable and indeed this report is a replacement for the original task and report as the first part of deliverable 3.4 of the ENRICH project. The original report related to the incorporation of METS and TEI together in the manuscriptorium framework, but the workpackage discovered that this was not necessary (the ENRICH TEI Specification catered for all the necessary metadata). More detail concerning this replacement is available in the Sections "ENRICH Garage Engine as a replacement deliverable" and "The background of the ENRICH Garage Engine" below. This report provides documentation for the EGE software which is released freely under an open source licence (GPL version 3). An introduction to the EGE Framework describes the basic concepts of how formats can be migrated through multiple transformative steps. This is then followed by an introduction to the EGE Web Client, a user-friendly web interface allowing format transformation based on predefined profiles. The report then delves a bit deeper in to the internal mechanisms of the EGE and how it works, followed by an overview of the EGE API. The creation of EGE Extensions to add additional functionality is documented along with the necessary XSL Converter Configuration. The EGE is available as a RESTful Web Service interface and the interaction with this for both conversion and validation of documents is detailed. The report concludes with information on downloading as well as installing and configuring the EGE web package under Apache Tomcat. This is followed by some final conclusions and recommendations. The recommendations of this report are:

This report would like to make the following five recommendations:

1. The stylesheets used for transformations in the EGE are sufficient for their usage in the ENRICH project. However, for greater use by other projects additional development should be encouraged of these transformations including further development for additional use cases and greater documentation on the extensibility and customisation of the underlying stylesheets. It is recommended that any additional development work in format migration and conversion be used to improve the existing EGE solution.
2. Increased development should also be considered for new stylesheets for additional formats (such as MARC) since the addition of a single new stylesheet then plugs into the EGE to allow a number of news transformation paths. It is recommended that where new format migration and conversion stylesheets are being developed for future projects, that these do so in a manner compatible with the EGE framework so that they may hopefully be incorporated into it.
3. The EGE is a coordinated RESTful web service framework that uses a documented API for interaction. Projects producing web-based software should use approaches that allow access to such APIs for building services on them, here demonstrated by the ENRICH EGE Web Client. Linking together of these individual web services to a greater whole is a roadmap for richer and more robust web applications. It is recommended that projects developing web-based software should produce a RESTful web service API and build and front-end systems on top of this.
4. The EGE is an open extensible system which one is able to customise to individual project needs. It is also open source, released under the GPL version 3 licence.

Where possible it is recommended that software be open source and built in a manner that encourages extended and improving that software.

5. TEI P5 XML, here used as the ENRICH TEI P5 Specification, is not only a good format for the representation of textual materials and metadata, but is also sufficient as an intermediate format in a conversion pathway between two disparate formats. TEI P5 XML is recommended for any similar endeavours undertaken at a later date.

TABLE OF CONTENTS

2 ENRICH Garage Engine as a replacement deliverable

The inclusion of the ENRICH Garage Engine (EGE) and its documentation is as a replacement deliverable in the context of the ENRICH project. When the original description of work was undertaken it was assumed that Manuscriptorium would move to a system that required METS containerization. Thus task 3.3 was set out as “Enhancement of internal environment of Manuscriptorium through implementation of METS containerization of the Manuscriptorium Scheme”. In investigating this task, OUCS and AIP discovered that because of the consequent developments in the TEI P5 ODD Specification created for the ENRICH project, and the adoption of this Specification for the internal Manuscriptorium Scheme, that the use of METS inside Manuscriptorium was not only now unnecessary, but undesirable. The aspects of METS that ENRICH would have leveraged are now entirely catered for in the ENRICH TEI P5 specification, and it is more beneficial for long-term preservation to keep to this specification. OUCS found that the investigation phase of this task was still useful in that it identified concretely that METS was not required in this instance. Simultaneously, having developed migration tools as part of task 3.1 and reporting on them in deliverable 3.3, it was decided that as part of WP3 OUCS will develop a new conversion tool framework with the remaining time left in task 3.3. This set of web services is called the ENRICH Garage Engine (EGE) would be designed to assist in the conversion, transformation, and validation of ENRICH data to other formats. This includes not only migration from legacy formats but transformations of ENRICH data to and from other applications such as MSWord. The EGE web services have been produced by PSNC. All software produced as part of this task has been be publicly released under an open source license (GPL version 3 in the case of the EGE). This has (more than) replaced the time originally budgeted for the remainder of task 3.3 and the portion of the deliverable (D 3.4 part 1) concerning METS/TEI interoperability is instead replaced by the documentation on the development and use of the EGE. The report on the handling of Unicode and non-Unicode data in Manuscriptorium and TEI P5 conversions will remain unchanged as part of that deliverable (D 3.4 part 2).

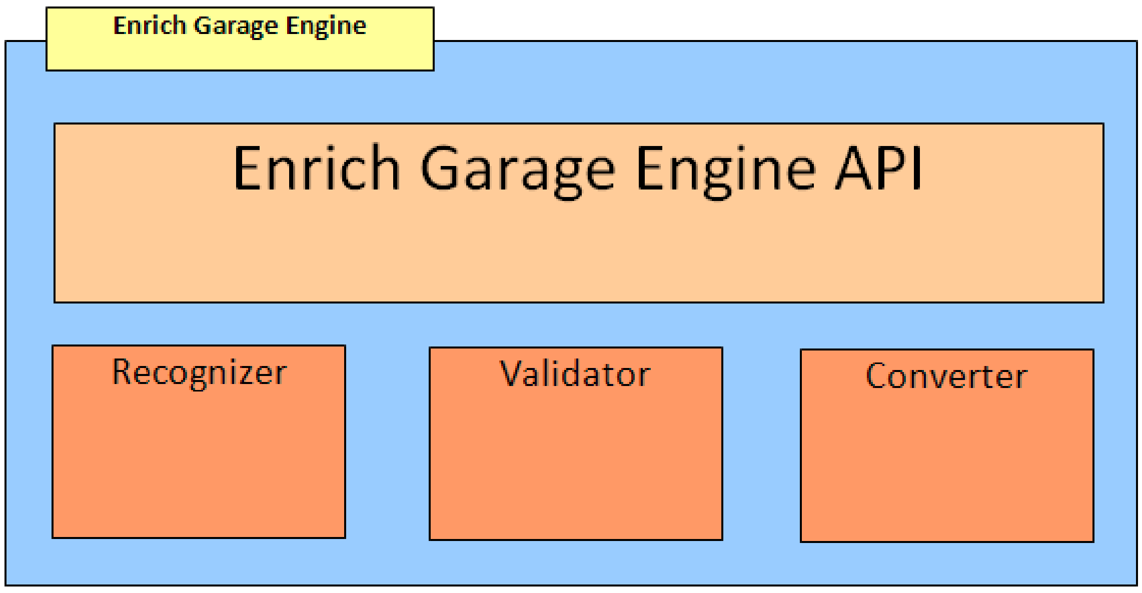
3 The background of the ENRICH Garage Engine

The ENRICH Garage Engine (EGE) is a migration tool technology which was not conceived for the original Description of Work for the ENRICH project but has now been developed by the PSNC ENRICH Partner as part of WP3. This tool is a Java-based format migration engine accessible in a number of different manners, such as through a web-form or directly via a RESTful API, which will facilitate the migration of legacy data through a number of different formats. Partly it builds on work that the TEI has done for the International Standards Organization (ISO), in seamlessly round-tripping ISO standards documents from Microsoft Word to TEI P5 XML and back again. The EGE allows users to submit a document for conversion to another format, and it will be analyzed, recognized as a particular type, converted, validated, and returned in a user-friendly manner. This builds upon a number of existing stylesheets and conversions to allow conversion through multiple formats using an intermediate formats. The EGE was a recommended development of the report of ENRICH D3.3 on format migration and is discussed in more detail there.

4 Introduction to the EGE framework

The EGE is a major part of the ENRICH Garage system; its implementation is responsible for the management of conversions, transformations and validations of data performed by the ENRICH Garage. Conversions, transformations and validations are performed by a set of plug-ins compatible with the EGE plug-in specification. The software was initially created by the PSNC and OUCS as a part of the ENRICH project funded by the European Commission. EGE is distributed under an open-source license.

The EGE is a framework. It provides definitions of plug-ins (building blocks) which can be used in combination to perform a series of operations. There is also the API that can be used by other applications utilizing EGE. The image below presents the overview of the EGE architecture.



The use of the EGE is possible via the EGE API, which allows a program to:

- discover the functionality available for a particular data format that EGE provides with a current set of plug-ins;
- perform those available operations (e.g. conversion).

There are three basic types of EGE plug-ins: Recognizer, Converter and Validator.

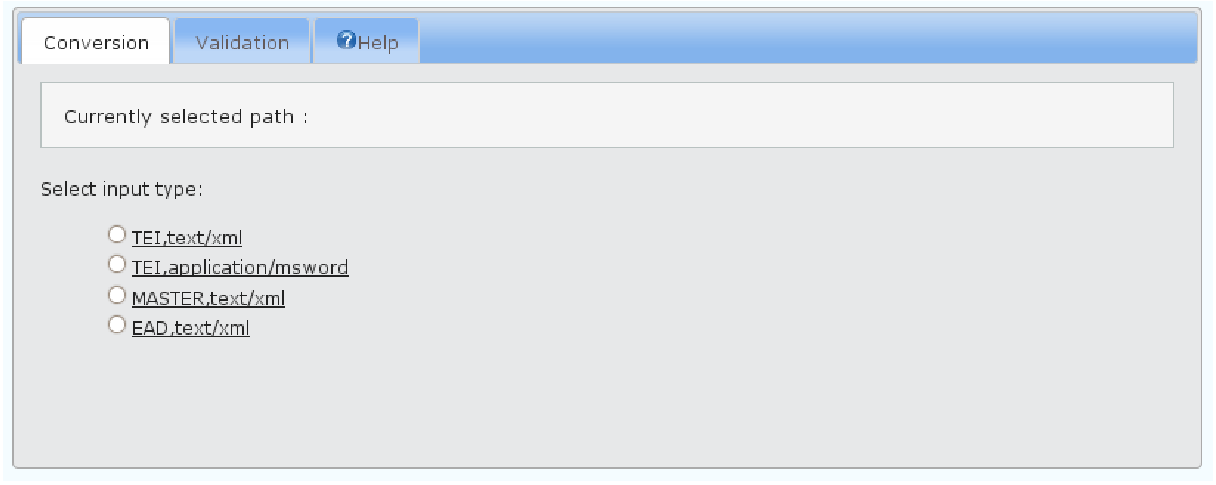
- **Recognizer** - this plug-in is responsible for the recognition of the Internet Media Type (MIME type) of the given input data. For example, it will receive the input data and state that the input data has text/xml MIME type. The recognized data may then be further validated to check the format of the data.
- **Validator** - this plug-in is responsible for validation of the input data. For example it may be used to validate the ENRICH TEI P5 data stored in a MIME type (e.g. text/xml) either received from end user or created by one of the converters. The following notation is assumed: ENRICH TEI P5 (text/xml) - it means that validator is able to validate ENRICH TEI P5 format encoded in text/xml.
- **Converter** - this plug-in is responsible for converting the input data. It may be, for example, conversion from XML to Word, conversion from Word to PDF, conversion of the XML from one form to another (e.g. MASTER -> ENRICH TEI P5) or even cleaning the input data (e.g. removing redundant information).

The idea is that there may be a number of implementations for each particular plug-in type (e.g. two recognizers, five converters and three validators) connected to EGE instance. Plug-ins are connected to the EGE automatically at the start time and can be combined to perform complex iterative operations. The extensions mechanism is based on the Java Plugin Framework (http://jpf.sourceforge.net/) library. Thanks to this approach it is possible for other interested parties to extend the EGE if desired.

5 Introduction to the EGE Web Client

The ENRICH EGE Web Client gives an easy to use front end for exploiting the abilities of the ENRICH EGE. The ENRICH EGE is responsible for the management of conversions, transformations and validations of data performed by the ENRICH Garage. Conversions, transformations and validations are performed by a set of plug-ins compatible with the EGE plug-in specification. The software was initially created by the PSNC and OUCS as a part of the ENRICH project funded by the European Commission. EGE is distributed under an open-source license. It is straightforward and intuitive for users and has a clear progression through the steps needed to transform a document from one format to another. The EGE Web Client is a user-friendly frontend for the EGE Web Service but if one was converting many files, or doing so automatically as part of another process, it might be more appropriate to use the web service instead.

5.1 Using the EGE Web Client



Currently selected path :

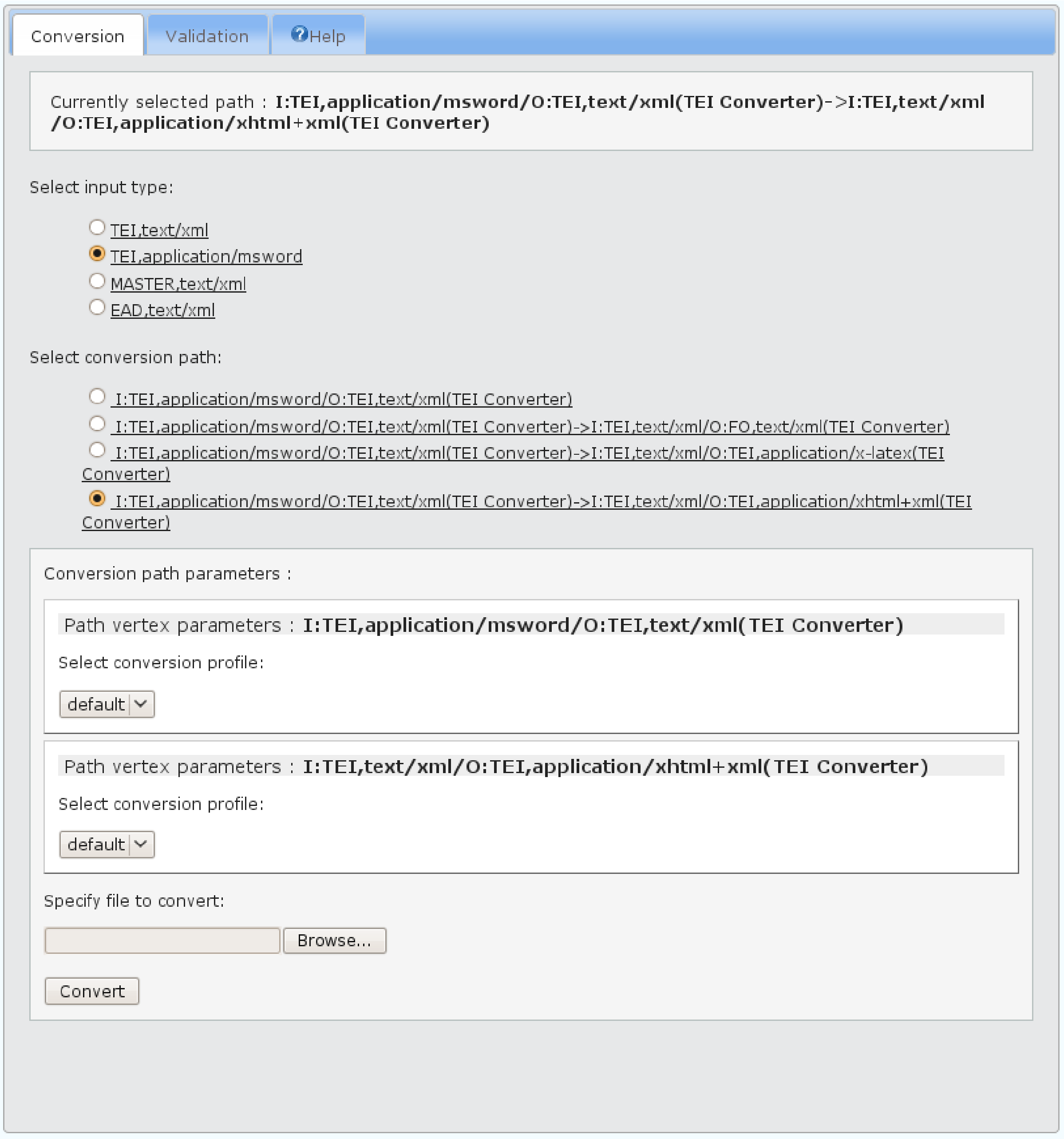
Select input type:

- TEI,text/xml
- TEI,application/msword
- MASTER,text/xml
- EAD,text/xml

The EGE Web Client should already have been set up and running as a web application, all interaction with the EGE Web Client is through a web browser, no extra software should be needed, however javascript must be enabled. To use the EGE Web Client you go to the appropriate URL. In the case of the ENRICH project this is: http://dlibra.psnc.pl/ege-webclient/ for the time being. If this has not been set up, please go to the Download page . This webpage contains three tabs 'Conversion', 'Validation' and 'Help'. The last of these links through to the help documentation, while the first two are where one can undertake to use the main functions of the EGE Web Client.

5.2 Converting Files with the EGE Web Client

The EGE Web Client allows you to convert documents from one format to another, possibly using an intermediate format before a second conversion. To know what forms of document it can convert to, the EGE Web Client needs to establish what format the input document is in. Once it knows this it can offer a conversion path, and allow a choice between multiple conversion profiles, before converting your file.



Currently selected path : **I:TEI,application/msword/O:TEI,text/xml(TEI Converter)->I:TEI,text/xml/O:TEI,application/xhtml+xml(TEI Converter)**

Select input type:

- ( ) TEI,text/xml
- (●) TEI,application/msword
- ( ) MASTER,text/xml
- ( ) EAD,text/xml

Select conversion path:

- ( ) I:TEI,application/msword/O:TEI,text/xml(TEI Converter)
- ( ) I:TEI,application/msword/O:TEI,text/xml(TEI Converter)->I:TEI,text/xml/O:FO,text/xml(TEI Converter)
- ( ) I:TEI,application/msword/O:TEI,text/xml(TEI Converter)->I:TEI,text/xml/O:TEI,application/x-latex(TEI Converter)
- (●) I:TEI,application/msword/O:TEI,text/xml(TEI Converter)->I:TEI,text/xml/O:TEI,application/xhtml+xml(TEI Converter)

Conversion path parameters :

Path vertex parameters : **I:TEI,application/msword/O:TEI,text/xml(TEI Converter)**

Select conversion profile: default

Path vertex parameters : **I:TEI,text/xml/O:TEI,application/xhtml+xml(TEI Converter)**

Select conversion profile: default

Specify file to convert:

5.2.1 Selecting Input Type

The EGE Web Client allows you to specify an input document's type from a list of choices predefined by the conversions that have been installed into that instance of the EGE Web Client. In the case of the ENRICH EGE Web Client conversion service these are:

- TEI,text/xml
- TEI,application/msword
- MASTER,text/xml
- EAD,text/xml

This takes the format of:

```
FORMAT,INTERNET MIME TYPE (IMT)
```

So in the case of the ENRICH EGE Web Client conversion service there are three formats TEI, MASTER, and EAD, and only two different IMT's text/xml and application/msword that it knows how to handle as input types. The EGE will test and validate your document after uploading it against its definition for the format you select.

Please note that the MS Word format referred to here is the DOCX format. Earlier word formats will cause an error.

5.2.2 Selecting Conversion Path

The conversion path is the route through which the EGE can transform your documents to other formats. This may simply be a single conversion or involve more than one steps. For example, having chosen 'TEI,text/xml' as the input type, one of the options available to us for conversion is:

```
I:TEI,text/xml/O:TEI,application/msword(TEI Converter)
```

The 'I:' section of this path indicates the IMT input type of the document to be converted. In this case it is "TEI,text/xml". This is followed by the output format indicated by 'O:'. In this case that format is: "TEI,application/msword(TEI Converter)". This shows that our input document which is in TEI XML will be converted to MS Word using the TEI converter (which transforms TEI files to the docx format). The reverse of this would be from MS Word to TEI:

```
I:TEI,application/msword/O:TEI,text/xml(TEI Converter)
```

One of the great strengths of the EGE is that it allows multiple transformations through a variety of formats. For example, the following path:

```
 I:TEI,application/msword/O:TEI,text/xml(TEI Converter) ->
I:TEI,text/xml/O:TEI,application/xhtml+xml(TEI Converter)
```

details the conversion path from an MS Word document to TEI XML, and then (after the arrow '->' indicating a second step in the path), the XML that results from the first transformation becomes the input to a conversion from TEI XML to XHTML.

5.2.3 Conversion Profiles

Beneath the selection of a conversion the EGE Web Client gives you the opportunity to modify the conversion path parameters through the selection of a predefined conversion profile. In the ENRICH EGE Web Client this gives you a choice between 'default', 'enrich' and 'iso' conversion profiles. One drop-down selection box for a profile will appear for each major step in the chosen conversion path. If you have not set up any conversion profiles for your own project, then using 'default' is probably the most sensible.

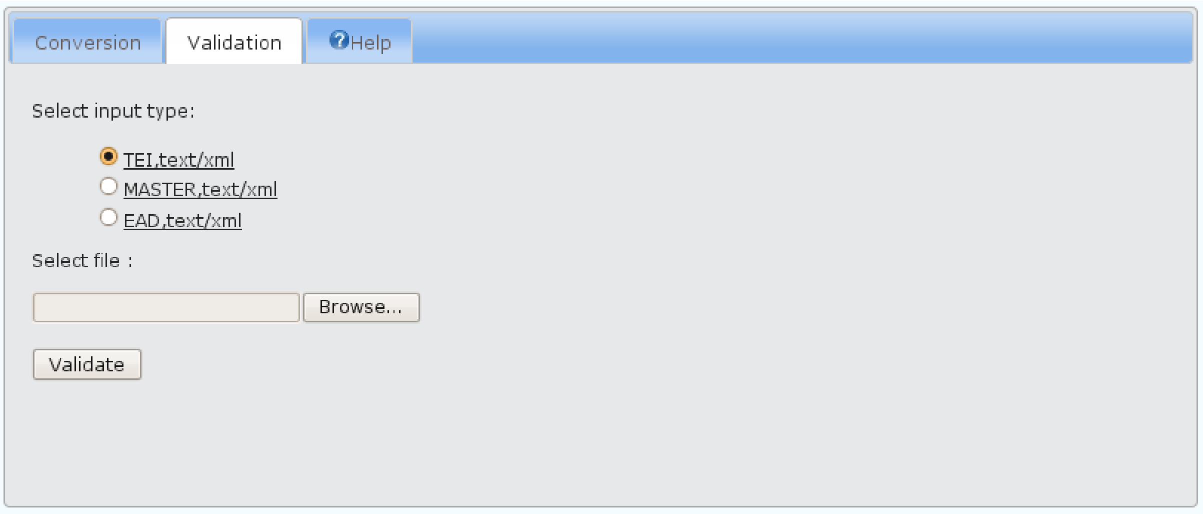
5.2.4 Converting Your File

Underneath where you select the conversion profiles you are prompted to "Specify file to convert:" and there is a 'Browse' but to look through your computer's file system to find a file of the selected input type to convert. Make sure that the document you select is in the correct format, and click on the 'Convert' button.

If your file has been converted successfully, you will be prompted to download it. If it has failed to convert and error message will be displayed. Make a note of this error message for any bug report you want to submit before trying again after ensure the correct input format has been chosen.

5.3 Validating Files with the EGE Web Client

In addition to converting your files, the EGE can simply be used to validate your files to ensure they are the correct input format. This takes place automatically when you upload your files for conversion, but it is also available as a separate feature on a second tab in the EGE Web Client.



Select input type:

[✓ TEI,text/xml]
MASTER,text/xml
EAD,text/xml

Select file :

5.3.1 Selecting Input Type

Similar to the 'Conversion' tab, the EGE Web Client validation service allows you to specify an input document's type from a list of choices predefined by the conversions that have been installed into that instance of the EGE Web Client. In the case of the ENRICH EGE Web Client these are:

- TEI,text/xml
- MASTER,text/xml
- EAD,text/xml

Note that MS Word is not an option for validation because the validation method deails with 'text/xml' files. So in the case of the ENRICH EGE Web Client validation service there are three formats TEI, MASTER, and EAD, and only one IMT's "text/xml" that it knows how to handle as input types. The EGE will test and validate your document after uploading it against its definition for the format you select.

5.3.2 Validating Your File

Underneath where you select the conversion profiles you are prompted to "Select file:" to validate and there is a 'Browse' but to look through your computer's file system to find a file of the selected input type to validate. Make sure that the document you select is in the correct format, and click on the 'Validate' button.

If your file has been validated successfully a message will appear indicating this success. If it has failed to validate the error message will be displayed. Make a note of this error message for any bug report you want to submit and correcting whatever the error is before trying again after ensuring the correct input format has been chosen.

6 Internal mechanisms of the EGE

Having read the overview of the EGE it is now possible to proceed with the idea of the internal mechanism. This will lead to specification of the EGE API which is available for use in external applications. This will also show the potential possibilities of the EGE in case, when there will be a number of implementations of particular components types. Below you will find a simple case study illustrating EGE internal mechanism.

6.1 EGE internal mechanisms: a case study

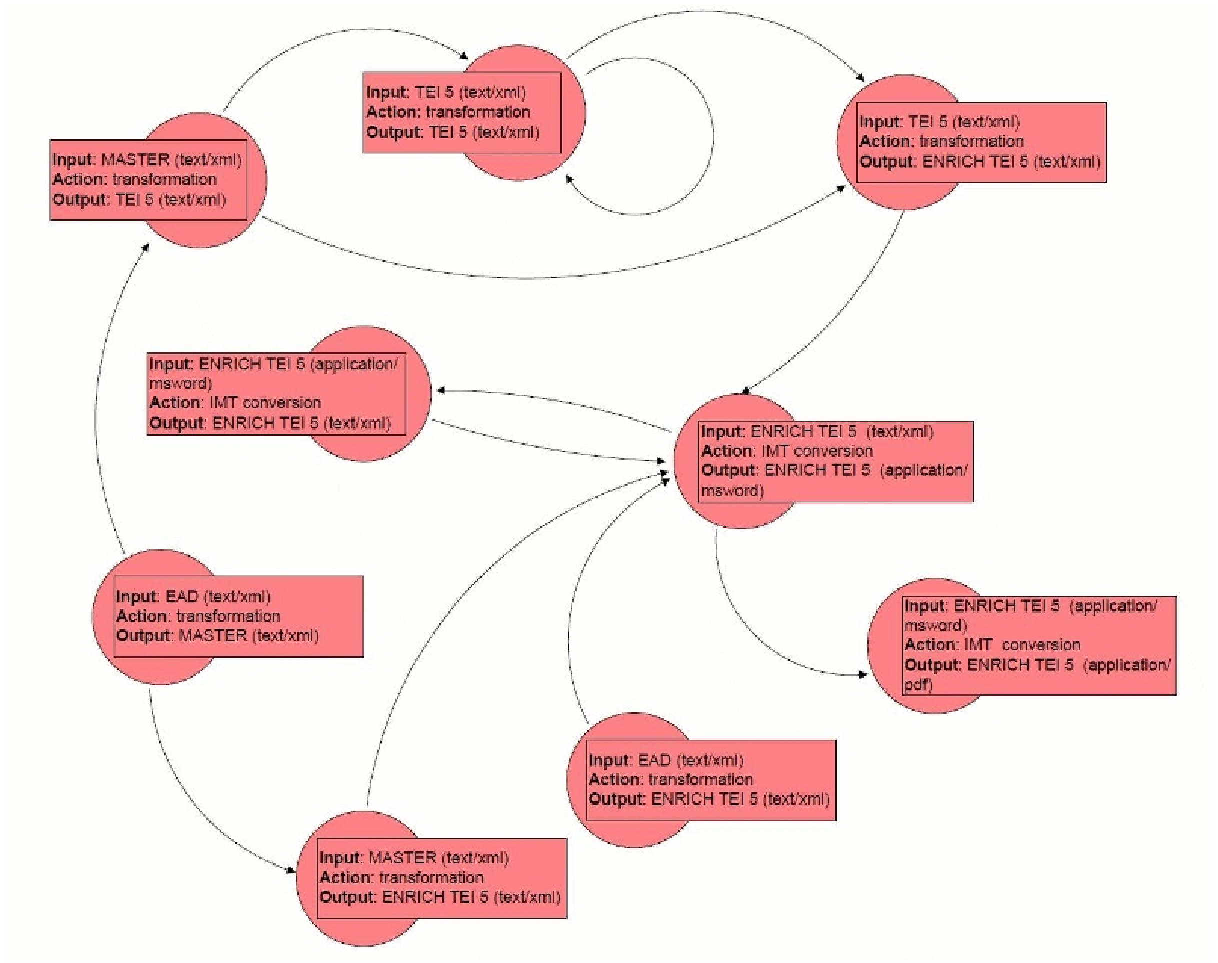
Let us assume that:

1. Initially we have recognizers which are able to recognize text/xml, application/pdf and application/msword. Recognizers allow us to recognize the Internet MIME Type (IMT) of the input data.
2. We assume that we have the following validators: MASTER (text/xml), TEI P5 (text/xml), ENRICH TEI P5 (text/xml), ENRICH TEI P5 (application/pdf), ENRICH TEI P5 (application/msword), EAD (text/xml). Validators allow us to validate recognized data and therefore state the format of the input data.
3. The available set of converters is described in the table below. The X mark in the table means that we have converter which is able to convert from the format specified in the row name to the format specified in the column name.

To / From	MASTER (text/xml)	TEI 5 (text/xml)	ENRICH TEI P5 (text/xml)	ENRICH TEI P5 (application /msword)	ENRICH TEI P5 (application /pdf)
MASTER (text/xml)		X	X		
TEI 5 (text/xml)		X	X		
ENRICH TEI P5 (text/xml)				X	
ENRICH TEI P5 (application /msword)			X		X
EAD (text/xml)	X		X		

Therefore with the above set of converters it is, for example, possible to convert from MASTER (text/xml) to TEI P5 (text/xml) and from ENRICH TEI P5 (application/msword) to ENRICH TEI P5 (application/pdf), but it is not possible to convert from EAD (text/xml) to ENRICH TEI P5 (application/pdf) or from TEI P5 (text/xml) to ENRICH TEI P5 (application/pdf). Additionally, we may imagine a converter from TEI P5 (text/xml) to TEI P5 (text/xml) which could simply clean the given input data of redundant information or restructure it in some pre-determined manner. Moreover, we could imagine a converter which will be independent of the format and will focus only on the IMT, e.g. converter which converts everything from (application/msword) to (application/pdf), although at this moment such feature is not supported.

Based on these three assumptions, let us build a directed graph of possible conversion, just as it is being built in the EGE. The nodes in the graph indicate the converters (with input format, action and output format specified) and the directed edges (arcs) will connect the converters between each other. The arc from converter A to converter B exists if and only if the output data format and IMT of the converter A is identical as the input data format and IMT of the converter B. Based on that we have the following graph:



With the above graph and the format and IMT of the input data (recognized by one of the recognizers and then validated or given by the end user directly) we may determine the possible conversion paths for the input data. The simplest version of the algorithm to determine possible paths is as follows:

1. Find all the nodes that accept given input data format and IMT.
2. For each found node search for all the paths leading from the node (traverse the processing graph);
3. Found paths (and also their sub-paths) are the conversion possibilities for the given input data format and IMT.

There are some issues that the algorithm (and the EGE user to a certain degree) has to take into account:

- Loops - it is possible that a node has the same input format and output format (e.g. [TEI P5 (text/xml) > TEI P5 (text/xml)]). This may for example indicate a cleaning or restructuring converter.
- Cycles - it is important to be aware of the fact that a cycle may appear, and it cannot (and will not) break the algorithm. For example: [ENRICH TEI P5 (text/xml) > ENRICH TEI P5 (application/msword)], [ENRICH TEI P5 (application/msword) > ENRICH TEI P5 (text/xml)]. We assume that there is no sense to use the cycle in the resulting paths, e.g. convert from ENRICH TEI P5 (text/xml) to ENRICH TEI P5

(application/msword) and then convert back from ENRICH TEI P5 (application/msword) to ENRICH TEI P5 (text/xml).

- It is possible that there will be two different paths that lead from input format and IMT A to output format and IMT B. In our example there are two paths leading from EAD (text/xml) to ENRICH TEI P5 (application/msword): [EAD (text/xml) > ENRICH TEI P5 (text/xml)], [ENRICH TEI P5 (text/xml) > ENRICH TEI P5 (application/msword)] [EAD (text/xml) > MASTER (text/xml)] [MASTER (text/xml) > ENRICH TEI P5 (text/xml)] [ENRICH TEI P5 (text/xml) > ENRICH TEI P5 (application/msword)]

6.2 *Summary*

In the EGE each converter provides a set of conversion actions with specified input and output data, e.g. an action in which converter can use to convert from ENRICH TEI P5 format in text/xml IMT to ENRICH TEI P5 format in application/msword IMT. All conversion actions are connected through their input and output formats and forms convert graph. That representation enables the provision of a mechanism for converting data in a specified format to other another supported format in multiple ways - by searching paths in graph. Paths of conversion can be then be chosen by users through an implemented interface (e.g. GUI or web application forms).

Before converting each input data its IMT may be recognized by Recognizer component. Its format (e.g. ENRICH TEI P5) may then be validated by the Validator component. Alternatively the application which uses EGE may ask the user directly for the input IMT and data format.

7 Overview of the EGE API

The EGE API is a Java based framework that provides basic implementation and interfaces of mechanisms mentioned in the introduction to EGE. The EGE API is written in Java 1.5 (5.0). It directly uses three external libraries:

- JPF ver. 0.12 (http://jpf.sourceforge.net/) for plug-in management.
- JUNG ver. 2.0.1-SNAPSHOT (http://jung.sourceforge.net/) for conversion graph operations.
- Log4j ver. 1.2.12 (http://logging.apache.org/log4j/) for logging.

EGE API contains the following Java packages:

- **pl.psnc.dl.ege** - main package with the implementation of the EGE logic.
- **pl.psnc.dl.ege.component** - contains the interfaces describing the three main EGE components.
- **pl.psnc.dl.ege.exception** - contains the EGE exceptions.
- **pl.psnc.dl.ege.types** - contains the EGE data types used by the components and conversion graph.

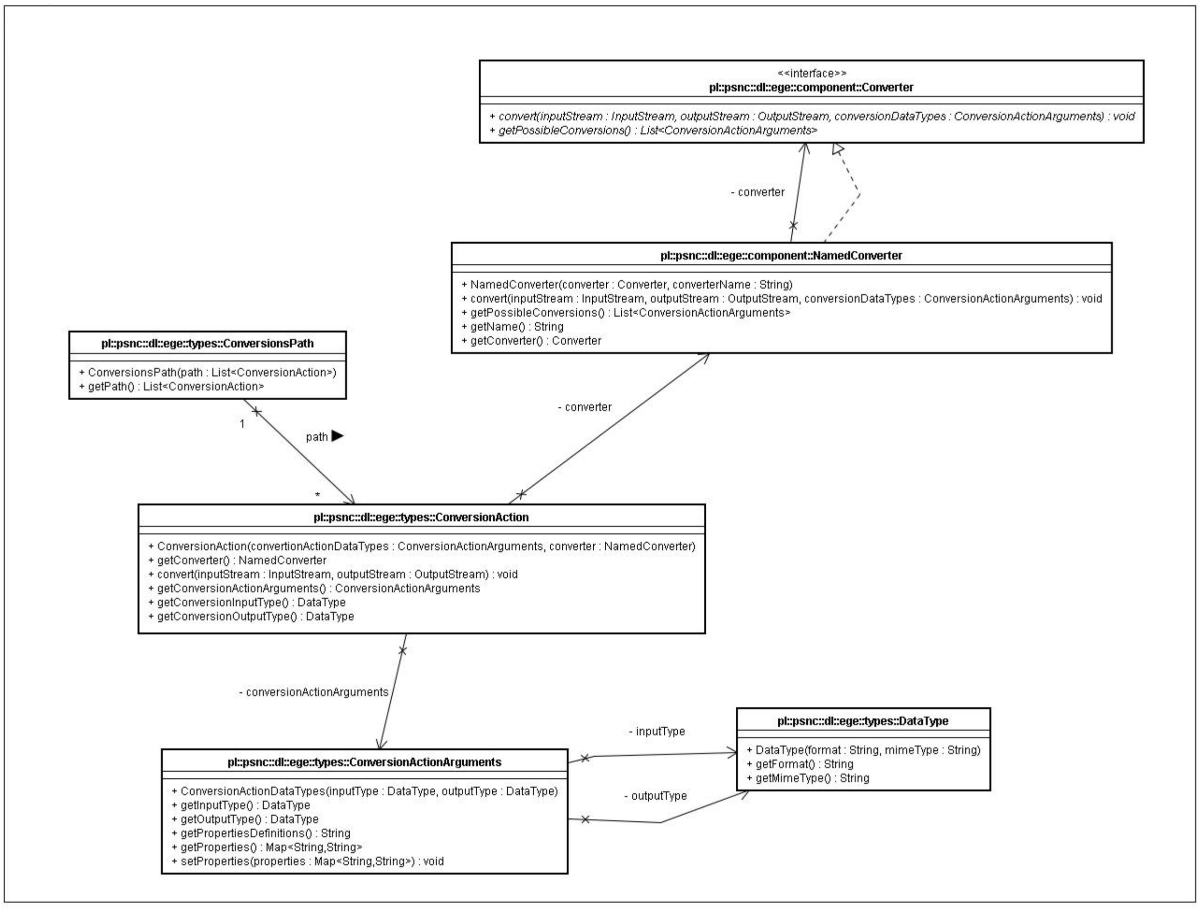
7.1 *EGE API specific data types (pl.psnc.dlteam.ege.types)*

Format and MIME type - both are represented as standard String data type. Format represents a name of format, like: ENRICH TEI P5 or EAD. MIME type can be for example: text/xml, application/pdf or application/msword. The values of format and MIME type are not validated in any way. This is intentional, because the internal EGE mechanisms are designed to be very general. If one would like to use the EGE in a context different than the original context connected with the ENRICH project, this should also be possible.

The pl.psnc.dl.ege.types includes the following classes:

- **DataType** - contains both format and MIME type information; instances of this class are used to describe both input and output data types in EGE.
- **ConversionAction** - every instance of this class describes one particular conversion operation, which can be performed by specified converter component on concrete input/output data types. A particular converter A can perform conversion of a data in X DataType to Y DataType. The information about X and Y DataTypes is stored in ConversionActionArguments class instance. ConversionAction is also a base object class for a node in conversion graph. Each conversion action can be dynamically configured through properties provided with ConversionActionArguments class instances.
- **ConversionActionArguments** - describes input and output data types for a conversion action. Input and output data types are instances of a DataType class. Each instance provides also arguments for dynamic parameterization of conversion.
- **ConversionsPath** - each instance of this class contains a list (a sequence) of a ConversionAction objects; this list represents path of chained conversion where: input data type from first element of sequence is a source data type and output data type from last element of sequence is a result data type. Adjacent elements of such path have its input/output data types compatible - this is assured during the process of conversion graph construction.
- **ValidationResult** - instances of this class are returned by validation methods. Each instance contains status of validation result (ERROR,SUCCESS or FATAL) and messages.

The UML diagram below shows described relationships between classes.



<<interface>>
pl::psnc::dl::ege::component::Converter
+ convert(inputStream : InputStream, outputStream : OutputStream, conversionDataTypes : ConversionActionArguments) : void
+ getPossibleConversions() : List<ConversionActionArguments>
- converter
pl::psnc::dl::ege::component::NamedConverter
+ NamedConverter(converter : Converter, converterName : String)
+ convert(inputStream : InputStream, outputStream : OutputStream, conversionDataTypes : ConversionActionArguments) : void
+ getPossibleConversions() : List<ConversionActionArguments>
+ getName() : String
+ getConverter() : Converter
pl::psnc::dl::ege::types::ConversionsPath
+ ConversionsPath(path : List<ConversionAction>)
+ getPath() : List<ConversionAction>
1
path ▶
*
- converter
pl::psnc::dl::ege::types::ConversionAction
+ ConversionAction(convertionActionDataTypes : ConversionActionArguments, converter : NamedConverter)
+ getConverter() : NamedConverter
+ convert(inputStream : InputStream, outputStream : OutputStream) : void
+ getConversionActionArguments() : ConversionActionArguments
+ getConversionInputType() : DataType
+ getConversionOutputType() : DataType
- conversionActionArguments
pl::psnc::dl::ege::types::ConversionActionArguments
+ ConversionActionDataTypes(inputType : DataType, outputType : DataType)
+ getInputType() : DataType
+ getOutputType() : DataType
+ getPropertiesDefinitions() : String
+ getProperties() : Map<String,String>
+ setProperties(properties : Map<String,String>) : void
- inputType
- outputType
pl::psnc::dl::ege::types::DataType
+ DataType(format : String, mimeType : String)
+ getFormat() : String
+ getMimeType() : String

Each instance of ConversionsPath contains sequence of ConversionAction instances. ConversionAction instance references instance of a class that implements Converter interface (loaded by the JPF at EGE start-up) and also specifies conversion action data types (through the instance of the ConversionActionArguments class). ConversionActionArguments contains information about input and output data types, so each instance of this class references two instances of DataType class. DataType is a elementary class that contains information about format and MIME type, both kept as String.

Conversion process can be dynamically parameterised with properties described within ConversionActionArguments instances. ConversionActionArguments contains properties definitions written as String. Each definition of property should contain at least : unique id of property and its data type. Syntax of properties definitions should be properly described in converter documentation. With documented syntax client application can translate available properties in order to provide e.g. user interface for conversion configuration. Properties configured through default client application interface can be transferred to converter using map argument of ConversionActionArguments instance, where : key in map is a unique "id" of property and value is value assigned to property. Validation of properties and default settings should be a converters task.

7.2 Components interfaces

Each component in the EGE (validator, converter or recognizer) can have multiple implementations provided through the extension mechanism of JPF. In order to provide this extensibility in the EGE API the three above interfaces were defined in the pl.psnc.dl.ege.component package:

- **Recognizer** - declares major functions of the EGE recognizer, has to be implemented by every external recognizer.
- **Validator** - declares major functions of the EGE validator, has to be implemented by every external validator.
- **Converter** - declares major functions of the EGE converter, has to be implemented by every external converter.
- **ConfigurableConverter** - extends the standard Converter interface with one additional function of configure().

The description of the methods declared in these three interfaces is presented in following sections.

7.3 Recognizer

Recognizers are responsible for the recognition of the MIME type of data. MIME types are media types identifiers registered by IANA (Internet Assigned Number Authority), the EGE however intentionally does not provide any checking mechanism of this standardization. Therefore the MIME types are just instances of the String type.

Declared methods:

- **getRecognizeableMimeTypes(): List<String>** - should return a list of MIME types recognized by this recognizer.
- **recognize(InputStream inputData) : String** - for provided input data a particular recognizer implementation tries to recognize the input MIME type and returns it or throws an exception if the input MIME type was not recognized.

7.4 Validator

Validators are responsible for validating the input data with respect to its format, e.g. if sent data is in ENRICH TEI P5 format.

Declared methods:

- **getSuportedValidationTypes() : List<DataType>** - method returns data types that the implemented validator is able to validate.
- **validate(InputStream inputData, DataType inputDataType) : ValidationResult** - method returns instances of the **ValidationResult** type. Every result contains a status (whether the validation ended with success, error or fatal error) and validation messages (about errors or warnings). If **inputDataType** is not supported by Validator implementation, method should throw ValidatorException.

7.5 Converter

Converters perform conversion of given input stream and store the result of conversion into the given output stream.

Declared methods:

- **getPossibleConversions() : List<ConversionActionArguments>** - should return list of arguments with pairs of input/output data types supported by the converter implementation and the properties definitions associated with those pairs.

- **convert(InputStream inputData, OutputStream outputData, ConversionActionArguments conversionArguments) : void** - performs conversion of input data contained within given input stream and puts the converted data into the given output stream. Both the expected input data type and the output data type are contained within the conversionArguments parameter and they should be compatible with the particular converter possibilities. With the basic input/output arguments method can receive conversion parameters filled according to the parameters definitions syntax (also contained within the ConversionActionArguments instance).

7.6 ConfigurableConverter

This extends the standard Converter interface with additional configure() method.

Declared methods:

- **configure(Map<String,String> params) : void** - converters that implements this interface can receive additional parameters from JPF plugin descriptor. The method is executed from the EGE configuration manager which translates taken parameters into the map argument. The converter is responsible for reading the map and setting up the configuration. The converter can inform EGE configuration manager about configuration errors by throwing an EGEException; improperly configured converter will be disconnected from EGE.

7.7 EGE interface and implementation

The main EGE interface is the **pl.psnc.dl.ege.EGE** . It describes functionality for complex and multiple conversions of data. The intention of the EGE is to construct a graph of conversions, where every possibility of conversion is describes by graph paths. The graph structure and its basic algorithms are implemented through external JUNG library by the **EGE framework** class - **pl.psnc.dl.ege.EGEImpl** which is also the main implementation of the EGE interface.

The main methods of **pl.psnc.dl.ege.EGE** interface are:

- **findConversionPaths(DataType sourceDataType) : List<ConversionsPath>** - finds all possible ways of conversion for the given data type; all those ways (conversion paths), are returned as a List of ConversionsPath instances. ConversionsPath instances received from this method can be used as the input parameter of a performConversion() method.
- **findConversionPaths(DataType sourceDataType, DataType resultDataType) : List<ConversionsPath>** - finds all possible ways of conversion from the specified sourceDataType to resultDataType. Depending on the set of loaded converters, there can be several parallel paths.
- **performConversion(InputStream inputData, OutputStream outputStream, ConversionsPath path) : void** - performs multiple conversions described by the ConversionsPath parameter. Converted input data - provided through the given input stream is written to the output described by the given output stream.
- **performValidation(InputStream inputData, DataType inputDataType) : ValidationResult** - this method performs validation using all loaded through the extension mechanism Validator interface implementations. The method returns ValidationResult instance which contains the status of a result and the validation

messages. If no validator recognizes inputDataType as supported then an exception will occur.

- **performRecognition(InputStream inputData) : String** - This method performs the recognition of the MIME type of an input data using all loaded EGE recognizers. If any of the loaded EGE recognizers decodes the MIME type of the input data, the method returns the String value of the MIME type, otherwise the method throws an exception.

The EGE interface methods are implemented by the EGEImpl class of the EGE framework. Internally the conversion graph is initialized in the EGEImpl constructor from the loaded external plugins - implementations of the Converter interface. JPF extensions are managed through an instance of **pl.psnc.dl.ege.EGEConfigurationManager** class. From each loaded converter its supported ConversionActionArguments are read and used for the creation of nodes for the conversion graph. During the graph construction the nodes are connected with the directed edge by rule: an arc from node A to node B can only be added if at least one of the node A output data types is compatible with at least one of the node B input data types. For each compatible input/output pair an arc in the graph is added.

Important Note: EGE API assumes conversion of the data by usage of the streams - one input stream for the input data and one output stream for the output data. In order to make it possible to provide input data and output data consisting of multiple files/directories, EGE implementation requires that every EGE converter accepts data and outputs data by means of a ZIP archive. This requirement is crucial not only for appropriate conversion of data consisting of multiple files/directories, but also for conversion results consisting of multiple files/directories. To have a simple rules for EGE converter creation, EGE implementation requires every converter to obey this requirement. Additionally, for developers' convenience EGE implementation provides functionality for compressing multiple files into a ZIP archive and decompressing ZIP archive. These functions are provided by the ZipIOResolver class. An instance of this class is returned by the *getStandardIOResolver()* method of the *EGEConfigurationManager* instance. Please, note that this requirement is stated by this specific EGE implementation and not by the EGE API itself.

8 Creating EGE extensions

The extension mechanism used in EGE is based on the Java Plugin Framework (JPF). More information about which can be found at: http://jpf.sourceforge.net . This section explains how to create sample implementation of a converter.

Currently the EGE project has to offer implementations of extensions :

- EGE TEI Converter - conversion of TEI format to other formats.
- EGE XSL Converter - conversion based on XSLT.
- EGE Validator - validator of XML based formats.

Important: EGE API assumes conversion of the data by usage of the streams - one input stream for the input data and one output stream for the output data. In order to make it possible to provide input data and output data consisting of multiple files/directories, EGE implementation requires that every EGE converter accepts data and outputs data by means of a ZIP archive. This requirement is crucial not only for appropriate conversion of data consisting of multiple files/directories, but also for conversion results consisting of multiple files/directories. To have a simple rules for EGE converter creation, EGE implementation requires every converter to obey this requirement. Additionally, for developers convenience EGE implementation provides functionality for compressing multiple files into a ZIP archive and decompressing ZIP archive. These functions are provided by the ZipIOResolver class. An instance of this class is returned by the *getStandardIOResolver()* method of the *EGEConfigurationManager* instance. Please, note that this requirement is stated by this specific EGE implementation and not by the EGE API itself.

8.1.1 Step 1: Java code

Each extension project has to import at least the **EGE API and EGE framework libraries (ege-api-X.jar,ege-framework-x.jar where X is the library version)** in order to be able to use the EGE interfaces, data types and extensions mechanism.

In this case the implemented class, named SampleConverter, will implement the **pl.psnc.dl.ege.Converter** interface. Implementing every method of this interface should result in code that looks like this:

```
package com.myapp.converter;


import java.io.InputStream;

import java.io.OutputStream;


import pl.psnc.dl.ege.component.Converter;

import pl.psnc.dl.ege.exception.ConverterException;

import pl.psnc.dl.ege.types.ConversionActionArguments;

import pl.psnc.dl.ege.types.DataType;


public class SampleConverter implements Converter

{

	public void convert(InputStream input, OutputStream output, ConversionActionArguments
conversionArguments) throws ConverterException, IOException

{

	// if conversionArguments are correct

		// perform proper conversion:

		// handle properties (if they were defined) taken from conversionArguments

		// read data from input stream

		// transform data according to implemented logic

		// write data into output stream

	}

	public List<ConversionActionArguments> getPossibleConversions()

{

		// return the list of ConversionActionArguments

		// that describes possibilities

		// of this particular converter implementation.

		return .........;

	}
}
```

Through the **getPossibleConversions()** method the converter informs the external application in which it is embedded about the possibilities of the converter. It is done by returning a list of pairs of data types and conversion properties definitions (instances of ConversionActionArguments). Method **convert()** should contain the necessary conversion logic, which checks the specific ConversionActionArguments, handles received parameters, performs conversion on data from the input stream and writes the result of conversion to a given output stream.

IMPORTANT: The input and output streams will be opened and closed by EGE. Therefore the plug-in code should not try to open or close those streams.

Creating converter with input and output data compression.

EGE implementation requires that every EGE converter accepts data and outputs data in form of a ZIP archive. In order to obey this requirement EGE converter may use functions available in the EGE implementation package. These functions are provided by the ZipIOResolver class. An instance of this class is returned by the *getStandardIOResolver()* method of the *EGEConfigurationManager* instance. IOResolver provides two simple methods for compression and decompression of data :

- decompressStream(InputStream is, File dir) : void - unpacks ZIP archive transferred through the given InputStream to a target 'dir' folder.
- compressData(File dir, OutputStream os) : void - packs specified source 'dir' to a ZIP archive and sends it to the OutputStream.

Example code showing the usage of EGE IOResolver is presented below:

```
package com.myapp.converter;


import java.io.InputStream;

import java.io.OutputStream;


import pl.psnc.dl.ege.component.Converter;

import pl.psnc.dl.ege.exception.ConverterException;

import pl.psnc.dl.ege.types.ConversionActionArguments;

import pl.psnc.dl.ege.types.DataType;


public class SampleConverter implements Converter

{

	private final IOResolver ior = EGEConfigurationManager.getInstance().getStandardIOResolver();


	public void convert(InputStream input, OutputStream output, ConversionActionArguments
conversionArguments) throws ConverterException, IOException

	{

		File inputTempDir = new File("in_foobar");

		File outputTempDir = new File("out_foobar");

		try{
```

```
                        ior.decompressStream(input, inputTempDir); // unpack transferred data into temporary
directory
                        // if conversionArguments are correct
                        // perform proper conversion:
                        // handle properties (if they were defined) taken from conversionArguments
                        // read data from temporary directory
                        // transform data according to implemented logic
                        // write data into output temporary directory
                        ior.compressData(outputTempDir, output); // compress converted data from output
temporary directory and transferr it to output stream.
                }finally{
                        // cleanup temporary data
                }
        }

        public List<ConversionActionArguments> getPossibleConversions()
        {
                // return the list of ConversionActionArguments
                // that describes possibilities
                // of this particular converter implementation.
                return .........;
        }
}
```

8.1.2 Step 2 : JPF Descriptor

Before creating a .jar file with the plug-in, it is necessary to create the JPF plug-in descriptor to mark the converter class as a JPF plug-in.

Descriptor file name is **plugin.xml** and it should look like this:

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.4"
"http://jpf.sourceforge.net/plugin_0_7.dtd">
(1) <plugin id="com.mycompany.converter" version="1.0">
(2)     <requires>
(3)             <import plugin-id="pl.psnc.dl.ege.root"/>
(4)     </requires>
(5)     <extension plugin-id="pl.psnc.dl.ege.root" point-id="Converter" id="SampleConverter">
(6)             <parameter id="class" value="com.myapp.converter.SampleConverter"/>
(7)             <parameter id="name" value="Sample Converter"/>
(8)     </extension>
(9) </plugin>
```

Every plug-in in JPF has its unique **id** , which is defined in the **plug-in** element by an **id** attribute (in our case it is **"com.mycompany.converter"** , see line 1). Every JPF extension is connected to a specified extension point, which in this case is **"Converter"** (described by a point-id attribute of the extension element in line 5, imported in lines 2 to 4). Extension points are defined within the JPF descriptor of EGE API and they are: Converter, Validator and Recognizer. More information about the JPF plug-in descriptors are available at http://jpf.sourceforge.net .

The extension element (lines 5 - 8) contains two additional parameters:

- In line 6 the id "class" and the value containing full name of the class that implements Converter interface, in this example it is "com.myapp.converter.SampleConverter".
- In line 7 the id "name" and value containing the name of the converter.

This descriptor has to be inserted in the root directory of extension .jar file or in its direct sub-directory named **"META-INF"** . Finally to use the created example converter in EGE client application, its .jar file(s) has to be added to the classpath of the EGE client.

8.1.3 Creating other extensions

The process of creating Recognizer and Validator extensions is very similar. First of all we need to create a class that implements one of the chosen interfaces - Reconizer or Validator and secondly change few things in the **plugin.xml** file:

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.4"
"http://jpf.sourceforge.net/plugin_0_7.dtd">
(1) <plugin id="com.mycompany.validator" version="1.0">
(2)     <requires>
(3)         <import plugin-id="pl.psnc.dl.ege.root"/>
(4)     </requires>
(5)     <extension plugin-id="pl.psnc.dl.ege.root" point-id="Validator" id="SampleValidator">
(6)         <parameter id="class" value="com.myapp.validator.SampleValidator"/>
(7)         <parameter id="name" value="Sample Validator"/>
(8)     </extension>
(9) </plugin>
```

This example shows the extension of Validator. First line shows that the **id** attribute of **plugin** element has changed into different name - every plugin must have a unique name. Line (5) shows the differences within extension element : **point-id** parameter is now of **"Validator"** value and the **id** parameter has changed into **'SampleValidator'** . Finally, it is necessary to change the parameter with id **"class"** in (6) line, where in the attribute value we have to point to our class name - that implements the Validator interface and also change the value attribute of the next parameter - name in line (7). Creating the Recognizer extension is almost the same. The difference is that we need to create the class that implements the Recognizer interface and particular attributes have to be changed in **plugin.xml** :

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.4"
"http://jpf.sourceforge.net/plugin_0_7.dtd">
```

```
(1) <plugin id="com.mycompany.recognizer" version="1.0">
(2)     <requires>
(3)         <import plugin-id="pl.psnc.dl.ege.root"/>
(4)     </requires>
(5)     <extension plugin-id="pl.psnc.dl.ege.root" point-id="Recognizer" id="SampleRecognizer">
(6)         <parameter id="class" value="com.myapp.recognizer.SampleRecognizer"/>
(7)         <parameter id="name" value="Sample Recognizer"/>
(8)     </extension>
(9) </plugin>
```

Once again it is necessary to change the following attributes :

- **id** attribute within the plugin element (line 1);
- **point-id** attribute within the extension element to 'Recognizer' (line 5);
- **id** attribute of the extension element to a unique value which will identify the extension (line 5);
- **value** attribute of both the parameter elements (lines 6-7).

9 XSL converter configuration

9.1 Introduction

The standard EGE API implementation includes two extensions by default:

- **TEI** format converter;
- **XSL** converter.

The XSL converter is able to transform an XML file using a given XSL transformation file. By default there are 2 XSL transformation files defined in the XSL converter plugin descriptor (see below) :

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.4"
"http://jpf.sourceforge.net/plugin_0_7.dtd">
<plugin id="pl.psnc.dl.ege.xsl.converter" version="1.0">
        <requires>
                <import plugin-id="pl.psnc.dl.ege.root"/>
        </requires>
        <extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter"
id="MasterToEnrichConverter">
                <parameter id="class" value="pl.psnc.dl.ege.MultiXslConverter"/>
                <parameter id="name" value="MASTER to Enrich Converter"/>
                <parameter id="xsluri" value="http://tei.oucs.ox.ac.uk/ENRICH/XSLT/xsl/master2enrich.xsl"/
>
                <parameter id="iMimeType" value="text/xml" />
                <parameter id="iFormat" value="MASTER" />
```

```
          <parameter id="oMimeType" value="text/xml" />
          <parameter id="oFormat" value="ENRICH TEI P5" />
     </extension>
     <extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter" id="EadToEnrichConverter">
          <parameter id="class" value="pl.psnc.dl.ege.MultiXslConverter"/>
          <parameter id="name" value="EAD to Enrich Converter"/>
          <parameter id="xsluri" value="http://tei.oucs.ox.ac.uk/ENRICH/XSLT/xsl/ead2enrich.xsl" />
          <parameter id="iMimeType" value="text/xml" />
          <parameter id="iFormat" value="EAD" />
          <parameter id="oMimeType" value="application/zip" />
          <parameter id="oFormat" value="ENRICH TEI P5" />
     </extension>
</plugin>
```

In order to add additional XSL transformations you have to modify the XSL converter plugin descriptor - 'plugin.xml' file located in the META-INF directory of the ege-xsl-converter.jar.

To add a new XSL transformation, please follow these 3 steps:

1. Unzip the **ege-xsl-converter.jar**.
2. Modify the **'plugin.xml'** file located in the META-INF directory.
3. Repack the files back to **ege-xsl-converter.jar**.

9.2 Modifying the 'plugin.xml' descriptor file (adding a new XSL transformation).

In order to add a new XSL transformation, you have to add a new extension to the 'plugin.xml' descriptor file.

There are 5 required properties of the extension that have to be provided for an XSL transformation :

- **xsluri** where the converter developer can specify URI address of xsl transformation scheme;
- **iMimeType** - defines the mime type of input data;
- **iFormat** - defines the format of input data;
- **oMimeType** - defines the mime type of output data;
- **oFormat** - defines the format of output data.

For instance, in order to add an XSL transformation that transforms data from FOO(text/xml) format to BAR(text/xml) format using foobar.xsl transformation scheme, available under http://pl.psnc.ege.pl/foobar.xsl, the following definition of an additional extension have to be included into the 'plugin.xml' descriptor file:

```
<extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter" id="FooToBarConverter">
          <parameter id="class" value="pl.psnc.dl.ege.FooToBarConverter"/>
          <parameter id="name" value="FOO to BAR converter"/>
          <parameter id="xsluri" value="http://pl.psnc.ege.pl/foobar.xsl" />
          <parameter id="iMimeType" value="text/xml" />
          <parameter id="iFormat" value="FOO" />
          <parameter id="oMimeType" value="text/xml" />
          <parameter id="oFormat" value="BAR" />
</extension>
```

Finally, the **'plugin.xml'** file would look like this:

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.4"
"http://jpf.sourceforge.net/plugin_0_7.dtd">
```

```
<plugin id="pl.psnc.dl.ege.xsl.converter" version="1.0">
	<requires>
		<import plugin-id="pl.psnc.dl.ege.root"/>
	</requires>
	<extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter"
id="MasterToEnrichConverter">
		<parameter id="class" value="pl.psnc.dl.ege.MultiXslConverter"/>
		<parameter id="name" value="MASTER to Enrich Converter"/>
		<parameter id="xsluri" value="http://tei.oucs.ox.ac.uk/ENRICH/XSLT/xsl/master2enrich.xsl"/
>
		<parameter id="iMimeType" value="text/xml" />
		<parameter id="iFormat" value="MASTER" />
		<parameter id="oMimeType" value="text/xml" />
		<parameter id="oFormat" value="ENRICH TEI P5" />
	</extension>
	<extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter" id="EadToEnrichConverter">
		<parameter id="class" value="pl.psnc.dl.ege.MultiXslConverter"/>
		<parameter id="name" value="EAD to Enrich Converter"/>
		<parameter id="xsluri" value="http://tei.oucs.ox.ac.uk/ENRICH/XSLT/xsl/ead2enrich.xsl" />
		<parameter id="iMimeType" value="text/xml" />
		<parameter id="iFormat" value="EAD" />
		<parameter id="oMimeType" value="application/zip" />
		<parameter id="oFormat" value="ENRICH TEI P5" />
	</extension>
	<!-- new FOO to BAR converter plugin declaration →
	<extension plugin-id="pl.psnc.dl.ege.root" point-id="XslConverter" id="FooToBarConverter">
		<parameter id="class" value="pl.psnc.dl.ege.FooToBarConverter"/>
		<parameter id="name" value="FOO to BAR converter"/>
		<parameter id="xsluri" value="http://pl.psnc.ege.pl/foobar.xsl" />
		<parameter id="iMimeType" value="text/xml" />
		<parameter id="iFormat" value="FOO" />
		<parameter id="oMimeType" value="text/xml" />
		<parameter id="oFormat" value="BAR" />
	</extension>
</plugin>
```

10 EGE Conversion and Validation RESTful Web Service interface

10.1 *Introduction*

EGE RESTful Web Service is a simple web application which exposes the EGE functionality as a set of RESTful online services.

Assuming that the EGE RESTful Web Service has been installed under 'ege-webapp' name in the application/servlet container, the following URL syntax should be used to perform operations:

```
http://[server_adress]:[port]/ege-webapp/[resources]
```

The EGE RESTful Web Service supports the following operations (request parameters are specified in the [resources] part):

- **Conversion operations**
  - **List input data types :** http://[server_adress]:[port]/ege-webapp/Conversions/ - lists all the supported input data types.
  - **List conversion paths:** http://[server_adress]:[port]/ege-webapp/Conversions/[input data type] - lists all the possible conversion paths for the given [input data type].
  - **Convert:** http://[server_adress]:[port]/ege-webapp/Conversions/[conversion path] - performs conversion of the given input data according to the given [conversion path].
- **Validation operations**
  - **List input data types :** http://[server_adress]:[port]/ege-webapp/Validation/ - lists all the supported input data types.
  - **Validate:** http://[server_adress]:[port]/ege-webapp/Validation/[input data type] - performs validation of the given input data.

The following sections describe in detail the above operations, in particular the syntax of the [input data type] and the [conversion path] parts.

10.2 Conversion

10.2.1 'List input data types' operation

This operation lists all data types (called input data types) which can be converted by this particular instance of the EGE RESTful Web Service. The operation should be invoked using the GET method of the HTTP protocol. In order to invoke this operation, use the following URL (we assume that the EGE RESTful Web Service is installed under 'ege-webapp' path of the application/servlet container):

```
http://[server_address]:[port]/ege-webapp/Conversions/
```

where [server_address] is the address of a server on which the EGE RESTful application is running, and the [port] is the port number of the server.

The following responses are possible:

- **status code 200** - when operation has been performed successfully and there are available conversions (there are certain input data types);
- **status code 204** - when no input data types are found - there are no conversions available;
- **status code 405** - 'wrong method' error, when requested URL suggests conversion operation.

For example, for the following request:

```
http://localhost:8080/ege-webapp/Conversions/
```

the following XML response is provided:

```
<?xml version="1.0" encoding="UTF-8"?>
<input-data-types xmlns:xlink="http://www.w3.org/1999/xlink">
	<input-data-type id="TEI,text/xml" xlink:href="http://localhost:8080/ege-webapp/Conversions/TEI
%3Atext%3Axml/" />
	<input-data-type id="MASTER,text/xml" xlink:href="http://localhost:8080/ege-
webapp/Conversions/MASTER%3Atext%3Axml/" />
	<input-data-type id="EAD,text/xml" xlink:href="http://localhost:8080/ege-
webapp/Conversions/EAD%3Atext%3Axml/" />
</input-data-types>
```

Response data includes a list of nodes describing each supported input data type. The following attributes are available for each input data type:

- **id** - the name of the input data type which should be used in further requests
- **xlink:href** - reference to the EGE RESTful operation - this operation will list all possible conversion paths for this input data type (identified by the 'id' value).

10.2.2 List conversion paths' operation

This operation provides a list of all conversions paths for the specified input data type. This operation is also invoked by the GET method of the HTTP protocol. To obtain a list of conversions paths use the following URL :

```
http://[server_address]:[port]/ege-webapp/Conversions/[input_data_type]
```

where [input_data_type] is the input data type identifier enclosed in the response of the "list input data types" operation.

Note: the [input_data_type] section of the URL contains code '%3A', which is an encoded ':' - colon sign. Colon sign separates every key part of data type from each other.
For example :

- In URL address - http://localhost:8080/ege-webapp/Conversions/TEI%3Atext %3Axml/, we can see that [input_data_type] has data type of TEI(text/xml) in a specially encoded format to fit the URI syntax requirements. First part is the format name, which is in our case 'TEI', then we have the first section of the mime type - 'text' and the second section - 'xml'.

The following responses are possible :

- **status code 200** - when the operation has been performed successfully and there exist conversions paths for a given input data type;
- **status code 204** - when no conversions paths are found for a given input data type;
- **status code 405** - 'wrong method' error, when the requested URL suggests "conversion" operation;

For example, for the following request:

```
http://localhost:8080/ege-webapp/Conversions/TEI;text,xml/
```

the following XML could be a response :

```
<?xml version="1.0" encoding="UTF-8"?>
<conversions-paths xmlns:xlink="http://www.w3.org/1999/xlink">
	<conversions-path xlink:href="http://localhost:8080/ege-webapp/Conversions/TEI%3Atext
%3Axml/TEI%3Aapplication%3Ax-latex/" >
		<path-name><![CDATA[
			I:TEI,text/xml/O:TEI,application/x-latex(TEI Converter)
		]]></path-name>
		<conversion index="0" >
```

```
                <property id="pl.psnc.dl.ege.tei.profileNames">
                    <value><![CDATA[default,enrich,iso]]></value>
                    <type>array</type>
                    <property-name>Profile</property-name>
                </property>
            </conversion>
        </conversions-path>
</conversions-paths>
```

The response contains a list of conversion paths available for a given input data type. Each conversion-path have the following attribute:

- **xlink:href** - reference to EGE RESTful operation of conversion,

and additional sub-nodes :

- **path-name** - which is a name of a conversion path;
- **conversion** - which represents one conversion action (being a part of the conversion path).

Each 'conversion' node (that represents conversion action) contains attribute 'index' - which is a number of the conversion action in sequence of conversions.

Additionally, every 'conversion' node can have one or more sub-nodes named 'property' which describe the possible parameters for a conversion action. Each 'property' node contains an attribute 'id' which identifies the property, and the following sub-nodes:

- **definition** - possible parameters, e.g. accepted property values;
- **type** - data type of the property;
- **property-name** - label of the property.

10.2.3 'Convert' operation.

This operation performs the conversion of a given file using the selected conversion path. This operation can be executed only through the POST method of the HTTP protocol. To perform the conversion of a file the client application has to send the POST request using the URL:

```
http://[server_address]:[port]/ege-webapp/Conversions/[conversion path]
```

where [conversion path] is a sequence of data types (the first one is the input data type), so

```
[conversion path]=[input data type]/[data_type_1]/[data_type_2]/.../[data_type_N]/ .
```

The request has to enclose the file for conversion and the optional request parameter named 'properties' which specifies the properties for concrete conversion actions in the conversion path. The [conversion path] should be understood as a sequence of conversion actions. For instance, the following conversion path:

```
TEI%3Atext%3Axml/TEI%3Aapplication%3Amsword/
```

defines one conversion action - conversion of the file in the TEI(text/xml) data type into a file in the TEI(application/msword) data type which is the result of the conversion operation.

Another example :

```
EAD%3Atext%3Axml/TEI%3Aapplication%3Amsword/TEI%3Aapplication%3Apdf/
```

defines two conversion actions: the first one converts the EAD(text/xml) input file into a TEI(application/msword) file, the second one converts the resulting TEI(application/msword) file into a TEI(application/pdf) file which is the final result of the conversion operation.

Basically, each pair of data type elements in the conversion path is a conversion action to be performed on the given file.

Optional parameter 'properties' should contain XML data similar to the following:

```
<conversions>
    <conversion index="[position of conversion action]" >
            <property id="[id of property]">
                [assigned value of this property]
            </property>
    </conversion>
</conversions>
```

where every 'conversion' node has the attribute 'index' - which represents the number of the conversion action within the conversion sequence.

Each 'conversion' node can contain sub-nodes named 'property' with the attribute 'id' and the value assigned to it. All possible properties for particular conversion action can be obtained from the "list conversion paths" operation.

For the 'conversion' operation, the following responses are possible:

- **200 status code** - conversion was performed without any problems and converted file was returned;
- **400 status code** - returned when requested conversion path does not exist;
- **405 status code** - 'wrong method' error, when trying to perform conversion operation over GET method of HTTP protocol;

10.3 Example use case.

Let us assume that:

- [server_addres] is 'localhost',
- [port] is 8080,

and we want to perform conversion of a file from TEI(text/xml) to TEI(application/msword) format using the default conversion profile.

To achieve our goal we have to :

10.3.1 Step 1 : list of data types

We perform a "list of data types" operation by sending HTTP request with GET method on URL : http://localhost:8080/ege-webapp/Conversions/ .

In response we receive the following XML data :

```
    <?xml version="1.0" encoding="UTF-8"?>
    <input-data-types xmlns:xlink="http://www.w3.org/1999/xlink">
        <input-data-type id="TEI,text/xml" xlink:href="http://localhost:8080/ege-
webapp/Conversions/TEI%3Atext%3Axml/" />
        <input-data-type id="MASTER,text/xml" xlink:href="http://localhost:8080/ege-
webapp/Conversions/MASTER%3Atext%3Axml/" />
        <input-data-type id="EAD,text/xml" xlink:href="http://localhost:8080/ege-
webapp/Conversions/EAD%3Atext%3Axml/" />
    </input-data-types>
```

Please note, that there is one node named 'input-data-type' which matches the input file data type - "TEI(text/xml)".

10.3.2 Step 2 : list conversions paths

- We now have a list of conversion paths for the input data type which matched with our input file data type:

  From the previous response we have to select the 'xlink:href' attribute (http://localhost:8080/ege-webapp/Conversions/TEI%3Atext%3Axml/) of the 'TEI,text/xml' input data type and use it to obtain the conversion paths.

  In order to receive the list of conversion paths, we perform 'list conversion paths' operation of the EGE RESTful Web Service by sending HTTP request with GET method on the extracted URL : http://localhost:8080/ege-webapp/Conversions/TEI%3Atext%3Axml/.

10.3.3 Step 3 : handling "list of conversions paths" response.

In the response we will receive the following XML data :

```
<?xml version="1.0" encoding="UTF-8"?>
<conversions-paths xmlns:xlink="http://www.w3.org/1999/xlink">
    <conversions-path xlink:href="http://localhost:8080/ege-webapp/Conversions/TEI%3Atext
%3Axml/TEI%3Aapplication%3Amsword/" >
        <path-name><![CDATA[
            I:TEI,text/xml/O:TEI,application/msword(TEI Converter)
        ]]></path-name>
        <conversion index="0" >
            <property id="pl.psnc.dl.ege.tei.profileNames">
                <definition><![CDATA[default,enrich,iso, ]]></definition>
                <type>array</type>
                <property-name>Profile</property-name>
            </property>
        </conversion>
    </conversions-path>
</conversions-paths>
```

XML contains only one conversion path, represented by 'conversions-path' node that gives us the possibility to perform conversion from TEI(text/xml) format to TEI(application/msword) format.

10.3.4 Step 4 : setting conversion properties.

XML response returned by the "list conversions paths" operation contains a list of conversions. In the above case there is only one conversion node. Each conversion node defines the properties for the conversion action it represents. In our example we have the 'pl.psnc.dl.ege.tei.profileNames' property with three possible values : default, enrich and iso. We will use 'default' value for the 'pl.psnc.dl.ege.tei.profileNames' parameter by setting the 'properties' parameter (for the 'convert' operation) to the following value:

```
    <conversions>
        <conversion index="0" >
            <property id="pl.psnc.dl.ege.tei.profileNames">
                default
            </property>
        </conversion>
    </conversions>
```

10.3.5 Step 5 : performing conversion.

We have to perform POST request on the the following URL: http://localhost:8080/ege-webapp/Conversions/TEI%3Atext%3Axml/TEI%3Aapplication%3Amsword/ , with a filled request parameter 'properties' and an input file to convert.

As a result we will receive the converted file (in the TEI(application/msword) format - '.docx' extension).

10.4 Validation

10.4.1 'List input data types' operation

This operation lists all the data types which can be validated by EGE RESTful Web Service. The operation should be invoked using the GET method of the HTTP protocol. In order to invoke this operation, use the following URL (we assume that the EGE RESTful Web Service is installed under 'ege-webapp' path of the application/servlet container):

- **http://[server_address]:[port]/ege-webapp/Validation/**

where [server_address] is the address of a server on which the EGE RESTful application is running, and the [port] is the port number of the server.

The following responses are possible:

- **status code 200** - when operation has been performed successfully and there are available data types for validation;
- **status code 204** - when there is no supported data types;
- **status code 405** - 'wrong method' error, when requested URL suggests validation operation.

For example, for the following request:

- **http://localhost:8080/ege-webapp/Validation/**

the following XML is received as a response:

```
<?xml version="1.0" encoding="UTF-8"?>
<validations xmlns:xlink="http://www.w3.org/1999/xlink">
	<input-data-type id="TEI,text/xml" xlink:href="http://localhost:8080/ege-webapp/Validation/TEI
%3Atext%3Axml/" />
	<input-data-type id="MASTER,text/xml" xlink:href="http://localhost:8080/ege-webapp/Validation/
MASTER%3Atext%3Axml/" />
	<input-data-type id="EAD,text/xml" xlink:href="http://localhost:8080/ege-webapp/Validation/EAD
%3Atext%3Axml/" />
</validations>
```

The response data includes a list of nodes describing each supported data type. The following attributes are available for each data type:

- **id** - the name of the data type which should be used in further requests;
- **xlink:href** - reference to the operation of validation.

10.4.2 'Validation' operation

This operation performs the validation of a given file using the selected conversion data type. This operation can only be executed through the POST method of the HTTP protocol. To perform the validation of a file the client application has to send the POST request using the URL:

```
http://[server_address]:[port]/ege-webapp/Validation/[input data type]
```

where [input_data_type] is the input data type identifier enclosed in the response of the "list input data types" operation.

The request has to enclose the file for validation.

For the 'conversion' operation, the following responses are possible :

- **200 status code** - validation was performed without any problems and response XML was returned;
- **400 status code** - returned when either request was formed unproperly or provided data type is not supported by any of the EGE validators;

With code '200' of result client application receives validation result encoded in XML format.

For example, for the following request:

- **http://localhost:8080/ege-webapp/Validation/EAD%3Atext%3Axml**

the following XML response is provided:

```
<?xml version="1.0" encoding="UTF-8"?>
        <validation-result>
                <status>ERROR</status>
                <messages>
                        <message>
                                <![CDATA[
                                        1) Error in line (3), column (8) : Document root element "TEI.2", must match
DOCTYPE root "ead".
                                ]]>
                        </message>
                </messages>
</validation-result>
```

Response data contains one "validation-result" section with :

- **status** - which may take one of three values : ERROR - when validator found some errors, SUCCESS - when no errors were found (warnings possible), FATAL - when fatal errors occured;
- **messages** - which contains multiple "message" sections, each with text values : error or warning messages.

10.5 Error response

During any operation an unexpected error may occur which will result in 500 status code of HTTP response, in that case application should also return xml data:

```
<?xml version="1.0" encoding="UTF-8"?>
<error msg="sample error message" exclass="com.my.SampleException">part of stack trace</error>
```

The response contains only one 'error' element with part of a stack trace as content and with the following attributes :

- **msg** - which contains the exception message;
- **exclass** - which contains the exception class.

11 Download

The EGE web package is available for download at:

http://dl.psnc.pl/software/EGE/download.html

Earlier versions of the EGE are also made available there as well.

12 Installing and configuring EGE web package

12.1 Installing the EGE

1. Download the Servlet/JSP container Apache Tomcat (version 5.5 and 6.x supported) from this page - Apache Tomcat and install it. Or install it through another method depending upon your operating system.
2. Unzip the downloaded **ege-webapp.zip** . Copy both "ege-webservice" and "ege-webclient" directories into the *'APACHE_TOCMAT_HOME/webapps/'* folder.
3. Configure the EGE web client - instructions can be found at : Download the EGE web client
4. Start Apache Tomcat from command line by typing **'startup'** from the *'APACHE_TOMCAT_HOME/bin/'* folder.
5. Run the EGE RESTful client application in your browser by going to the following URL: http://localhost:[port]/ege-webclient/, where [port] is your configured tomcat HTTP port (8080 by default).

12.2 Configuring of the EGE

Before starting EGE web client it is necessary to configure URL address of EGE RESTful web service in web.xml.

web.xml file can be found in /WEB-INF folder of EGE web client project. There is XML entry in configuration :

```
<context-param>
	<description>
		EGE RESTful web service URL.
	</description>
	<param-name>webservice.url</param-name>
	<param-value>http://localhost:8080/ege-webservice/</param-value>
</context-param>
```

Address to web service is specified inside **param-value** entity. By default there is "http://localhost:8080/ege-webservice/" .

Important: EGE web client extensively uses AJAX requests and therefore it has to be run over the same domain as EGE RESTful web service.

13 EGE Case Studies and Testing

The EGE conversions were initially tested with the same testbed of MASTER and EAD samples that were used in the investigation of the development and validation of migration tools as case studies. See http://tei.oucs.ox.ac.uk/ENRICH/Migration/index.xml for more information on these migration case studies.

The *Manuscript Access through Standards for Electronic Records* (MASTER) was an EU project funded to create a single XML-based standard for computer-readable descriptions of manuscripts. In many ways this project was the pre-cursor to ENRICH. The MASTER data format was updated and modified and eventually incorporated as a module into the Text Encoding Initiative (TEI) P5 Guidelines. MASTER itself was based on an earlier (P4) version of the Guidelines. The TEI P5 module has since evolved further and in turn has been used as the basis of the ENRICH specification. The ENRICH project has contributed its resolutions on the creation of manuscript descriptions back to the TEI and they have been ratified by the TEI Technical Council and adopted back into their Guidelines. Owing to the history of relationship between MASTER, TEI, and ENRICH, it was an obvious candidate for a case study on the development of migration tools. Since the development of transformation stylesheets for that investigation into migration tools (ENRICH D3.3), these stylesheets were included in the EGE default setup.

The test bed of MASTER files included some from the following :

1. **AMI**: About 500 full records relating to Icelandic manuscripts created by the Stofnun Árna Magnússonar (Arní Magnússon Institute) in Reykjavik
2. **BMR**: About 30 full records from the Manuscritos de América en las Colecciones Reales(American manuscripts in the Royal Collections) portal created at the University of Alicante
3. **IRHT**: About 350 short records relating to French manuscripts, extracted from the Médium database maintained at the Institut de Recherche et de l'Histoire des Textes, Paris
4. **KB**: About 90 records relating to Dutch and Flemish mediaeval manuscripts, from the collections of the Koninklijke Bibliotheek (Royal Library), The Hague
5. **NLP**:About 50 records relating to mediaeval manuscripts in the collections of the Národní knihovna České republiky (Czech National Library), Prague
6. **Well**: a small collection from The Wellcome Trust

In addition, the previous investigation had also tested Encoded Archival Description (EAD) as a format. EAD is a set of guidelines describing the intellectual and physical aspects of archival finding aids so that the information they contain may be easily searched, retrieved, displayed, and exchanged in a predictable platform-independent manner. It was produced by the Society of American Archivists and the Library of Congress. Many of the EAD concepts are based upon early work with the TEI and there is a large crossover of users between EAD and TEI. EAD is sometimes preferred for archival collection metadata and TEI is generally preferred for textual content, though increasingly TEI is becoming a de facto standard. The EAD format used by the Bodleian Library, University of Oxford, is based on version 1.0 of the standard. This was then superseded by a second version of the EAD standard in 2002. No successive versions of the EAD standards have since been released, though it is still used in many libraries.

The files provided by the Bodleian for this included:

- additional-a.xml (Additional A Manuscripts)
- additional-b.xml (Additional B Manuscripts)
- barlow.xml (The Barlow Manuscripts)
- don.xml (MSS Don., Donations)
- The input files to the conversion as a whole are available at: http://tei.oucs.ox.ac.uk/ENRICH/XSLT/testbed/EAD/

The EGE also provides conversions to/from MS Word's DOCX format (sometimes misleadingly called the 'Office Open' format), and to/from the standard TEI format. These have been tested with a number of documents, some containing manuscript descriptions and the results are impressive. However, there is certainly room for further development and improvement in this area.

14 Conclusions and Recommendations

The ENRICH Garage Engine as a RESTful Web Service with user-friendly web front-end is a good demonstrator of the development of useful service in a distributed 'web of data' method. It far outstrips the original task that it replaces and offers great scope for future development. Further development and maintenance is not funded as part of the ENRICH project, but it is hoped that it will be further developed in future funding bids. It provides an extremely good starting point for future work in format migration in service-oriented architectures.

This report would like to make the following five recommendations:

1. The stylesheets used for transformations in the EGE are sufficient for their usage in the ENRICH project. However, for greater use by other projects additional development should be encouraged of these transformations including further development for additional use cases and greater documentation on the extensibility and customisation of the underlying stylesheets. It is recommended that any additional development work in format migration and conversion be used to improve the existing EGE solution.
2. Increased development should also be considered for new stylesheets for additional formats (such as MARC) since the addition of a single new stylesheet then plugs into the EGE to allow a number of news transformation paths. It is recommended that where new format migration and conversion stylesheets are being developed for future projects, that these do so in a manner compatible with the EGE framework so that they may hopefully be incorporated into it.
3. The EGE is a coordinated RESTful web service framework that uses a documented API for interaction. Projects producing web-based software should use approaches

that allow access to such APIs for building services on them, here demonstrated by the ENRICH EGE Web Client. Linking together of these individual web services to a greater whole is a roadmap for richer and more robust web applications. It is recommended that projects developing web-based software should produce a RESTful web service API and build and front-end systems on top of this.

4. The EGE is an open extensible system which one is able to customise to individual project needs. It is also open source, released under the GPL version 3 licence. Where possible it is recommended that software be open source and built in a manner that encourages extended and improving that software.
5. TEI P5 XML, here used as the ENRICH TEI P5 Specification, is not only a good format for the representation of textual materials and metadata, but is also sufficient as an intermediate format in a conversion pathway between two disparate formats. TEI P5 XML is recommended for any similar endeavours undertaken at a later date.