D5.2 Initial demo of tools to transform asset representation

Grant Agreement nr	780470
Project acronym	SAUCE
Project start date (duration)	January 1st 2018 (36 months)
Document due:	31/03/2019
Actual delivery date	31/03/2019
Leader	DNEG
Reply to	William Greenly - wmg@dneg.com
Document status	Final version for submission

Project funded by H2020 from the European Commission

Project ref. no.	780470
Project acronym	SAUCE
Project full title	SAUCE
Document name	D5.2 Initial Demo of Tools to Transform Asset Representation
Security (distribution level)	Public
Contractual date of delivery	31/03/2019
Actual date of delivery	31/03/2019
Deliverable name	D5.2 Initial Demo of Tools to Transform Asset Representation
Type	Document
Status & version	Final version for submission
Number of pages	10
WP / Task responsible	5.1
Other contributors	
Author(s)	William Greenly
EC Project Officer	Ms. Cristina Maier, Cristina.MAIER@ec.europa.eu
Abstract	Demonstration of Asset Types and their capabilities being recognised by the framework for linking together
Keywords	Transformation, Capabilities, Search, Chaning, Compound
Sent to peer reviewer	Yes
Peer review completed	Yes
Circulated to partners	No
Read by partners	No
Mgt. Board approval	No

Document History

Version and date	Reason for Change
1.0 12-03-2019	document created by William Greenly
1.1 17-03-2019	Version for internal review
1.2 27-03-2019	Revisions in response to review: final version submitted to Commission

Table of Contents

1 EXECUTIVE SUMMARY

This document demonstrates how transformation capabilities and transformation options can be represented, combined and chained within the Smart Search Framework. This document provides snippets of working examples that prove the foundation of transformation capability along with the means to combine transformation capabilities and options, to transitively present new options that are materialised as a result of chaining transformation capabilities.

We begin by demonstrating how simple transformation capabilities can be advertised in the search framework using existing conventions and building blocks, and materialised in a search.

We then proceed to demonstrate how transformation capabilities from independent 3rd parties can be chained and combined within the search framework and results materialised thereof.

The advertising transformation capabilities and options builds heavily upon the search framework detailed in D4.1, therefore to fully or even partially appreciate the document, it is essential that it is assessed, evaluated and reviewed cross referencing the previous deliverable.

2 BACKGROUND

This document should be reviewed and evaluated in conjunction with the documented deliverable for WP4.1 (Smart Search Framework) due to the fact that the data architecture and technical architecture described therein, provide some of the basis for the components detailed in this document. This document makes specific references to items in D4.1, so without cross reference, evaluating the merit of this document will be difficult.

It should also be noted that this document very specifically describes a means to chain transformation capabilities on asset types. It does not describe the transformation framework as a whole since this is detailed in D5.3, hence it focuses very specifically on advertising asset transformation capabilities, proving how capabilities can be chained together. D5.3 will further build upon this, demonstrating a broader range of transformation capabilities and covering asset freshness and the actual means to run transformations.

3 INTRODUCTION

This chapter introduces the main goals and objectives and the method by which we demonstrate the deliverable and solution

3.1 Main objectives and goals

The main objectives of this deliverable are to:

1. Demonstrate how a transformation service can advertise it's transformation capabilities to the smart search framework
2. Demonstrate how theses capabilities can be chained together to provide transitive and compound transformation options.

3.2 Methodology

To demonstrate the the tools to transform asset representation we have delivered the following:

1. This document containing code samples and technical specifications
2. Two additional working code projects demonstrating how to advertise transformation capabilities and how they can be chained, incorporated into the existing smart search framework.

3.3 Convention

This document contains many snippets of code, which are pre-formatted accordingly to the most suitable code style with a black background. In some cases bold italics of code snippets has been done for the purpose highlighting of something important.

4 Advertising and Chaining Transformation Capabilities

In this chapter we describe how a transformation service or plugin can advertise it's transformation capabilities to the Smart Search Framework. We demonstrate how using components of the framework we can provide rules that can describe the transformation capabilities of a transformation service, incorporate these rules into the ingestion service and enrich assets that are ingested into the search engine with the transformation options for that asset.

We then proceed to show how rules can be chained together, providing compounded options that can then be exposed in a search result. It should be noted that for the purposes of demonstration only, the transformation options available for assets, will be materialised at ingestion time, not at query time, hence the examples are integrated with the ingestion and classification framework.

4.1 Advertising Transformation Capabilities

We will advertise the transformation capabilities of AccuTrans 3D in order to demonstrate the framework. This piece of software can take a range of input 3D file formats and convert them to a range of output file formats. For the purposes of demonstration we will only take a subset of transformation capabilities so that we can chain the others together using a second transformation service. Specifically these are as follows:

- AccuTrans **can read** Maya rtg files, but **cannot write** them
- AccuTrans **can read and write** Wavefront obj files
- AccuTrans **can write** X3D files, but **cannot read** them

We will show how to represent these transformation rules in a ontology, extending new core ontology transformation terms, and then include this ontology in the general purpose OWL reasoner enricher.

4.1.1 Creating the Transformation Rules

In D4.1 we described a core set of vocabularies that underpin the framework which provide terms and rules for describing and enriching data (section 5.1). In order to advertise transformation capabilities, we have extended the core vocabulary to include core transformation classes and properties that individual transformation services use in order to advertise their capabilities. The key class that has been added is the 'AvailableTransformation' class which represents that something has an available transformation. It is up to the transformation service to to extend this class with sub-classes which represent available transformations that can be produced as a result of using the transformation service. For the purpose of demonstration only, we shall do this by making additions to the existing sauce-lib-vocabulary project, but it is recommended that separate libraries are provides for individual transformation services since it decouples dependencies and makes versioning and deployment easier.

The resources folder in the sauce-lib-vocabulary project, contains the core terms. We will begin by adding the available transformation options to the core vocabulary (sce.owl).

```
sce:AvailableObjTransformation a owl:Class;
    rdfs:label "AvailableObjTransformation";
```

```
    rdfs:comment "something that can be transformed to an Obj file";
    rdfs:subClassOf sce:AvailableTransformation.

sce:ObjDistribution a owl:Restriction;
    owl:onProperty dcat:mediaType;
    owl:hasValue "text/prs.wavefront-obj";
    rdfs:subClassOf sce:AvailableObjTransformation.

sce:AvailableX3DTransformation a owl:Class;
    rdfs:label "AvailableX3DTransformation";
    rdfs:comment "something that can be transformed to an X3D file";
    rdfs:subClassOf sce:AvailableTransformation.

sce:X3DDistribution a owl:Restriction;
    owl:onProperty dcat:mediaType;
    owl:hasValue "model/x3d+xml";
    rdfs:subClassOf sce:AvailableX3DTransformation.

sce:AvailableRtgTransformation a owl:Class;
    rdfs:label "AvailableRtgTransformation";
    rdfs:comment "something that can be transformed to an Rtg file";
    rdfs:subClassOf sce:AvailableTransformation.

sce:RtgDistribution a owl:Restriction;
    owl:onProperty dcat:mediaType;
    owl:hasValue "application/autodesk.rtg";
    rdfs:subClassOf sce:AvailableRtgTransformation.
```

In the example above we show the transformation options, corresponding to the different 3D file formats. As well as these transformation options, we also state that every asset with an available distribution with a file format, is also a transformation option for that file format, since everything can be transformed into itself.

We now add a new vocabulary for the AccuTrans transformation service saving it in the resources folder as AccuTrans.owl and proceed by adding the transformation capabilities of the services as follows:

```
sce:AvailableRtgTransformation owl:subClassOf sce:AvailableObjTransformation,
sce:AvailableX3DTransformation.
sce:AvailableObjTransformation owl:subClassOf sce:AvailableX3DTransformation.
```

It should be clear from the example above that things which have AvailableRtgTransformations are also AvailableObjTransformations and AvailableX3DTransformations, since the transformation service can read rtg files and write them to obj and X3D files. Files which have AvailableObjTransformations are also AvailableX3DTransformations but not AvailableRtgTransformations since the tool cannot convert to rtg. X3D files cannot be read so there are not AvailableTransformations associated with these.

Now that the rules have been completed we add static variables to the appropriate classes in the library so that we can load the rules and make them available to the reasoner, make the appropriate version changes, and then build, test and publish the new library, as described in D4.1, section 6.1.1

4.1.2 Adding the Rules to the General Purpose OWL Reasoner

Now that the new transformation capabilities have been built test and published, they can be incorporated in the General Purpose OWL Reasoner details in section 6.2.4 in D4.1. We simply include the new version of the library into the dependencies and then add the vocabulary contained in the class to the asset as is ingested, the same way we do with the core vocabulary:

```
class OwlReasonerAction {

	public static JsonObject main(JsonObject args) {

	LdModel asset = new LdModel(SauceVocabulary.prefixes)

	asset.read(new ByteArrayInputStream(args.toString().getBytes()), null,
"JSON-LD")

	asset.add(SauceVocabulary.vocabulary)

	 // add the AccuTrans Transformation rules
	asset.add(AccuTrans3DTransformationVocabulary.vocabulary)

	JsonObject response = new JsonParser().parse(
asset.reason().json().toString() ).getAsJsonObject()

	log.info "Successfully completed reasoning"

	return response

	}

}
```

4.1.3 Demonstrating the results

We can demonstrate the results of the OWL reasoner by writing a integration test to prove that the correct transformation options are materialised, using mock asset data:

```
testAccuTransTransformationCapabilities() {

    def asset = new LdModel(SauceVocabulary.prefixes)

    asset.add   """

        scei:test-asset a sce:Asset;
              dcat:distribution scei:test-asset-obj-distribution,
scei:test-asset-rtg-distribution.

        scei:test-asset-obj-distribution dcat:mediaType
"text/prs.wavefront-obj".
        scei:test-asset-rtg-distribution dcat:mediaType
```

```
"application/autodesk.rtg"

    """

    def result = new LdModel(SauceVocabulary.prefixes)
    result.read(new ByteArrayInputStream(OwlReasonerAction.main( new
JsonParser().parse(asset.json()).getAsJsonObject() ).toString().getBytes()),
null, "JSON-LD")

    assertTrue result.ask("""scei:test-asset-obj-distribution a
sce:AvailableX3DTransformation""")
    assertTrue result.ask("""scei:test-asset-rtg-distribution a
sce:AvailableX3DTransformation""")
    assertTrue result.ask("""scei:test-asset-rtg-distribution a
sce:AvailableObjTransformation""")

}
```

After reasoning about our transformation options for the asset, it should be clear that the obj file can be transformed to an X3D file and the rtg file can be transformed to both a X3D file and an obj file.

4.2 Chaining Transformation Capabilities

We will now take another transformation tool and advertise it's capability to the framework, to demonstrate how transformation capabilities can be chained together. For the purpose of the demonstration, we will use a subset of the i3DConverter transformation capabilities, listed below:

- i3DConverter **can read and write** X3D files
- i3DConverter **can read and write** FBX files

We proceed to extend the core Sauce vocabulary with these transformation options in exactly the same way as was described in section 4.1.1. In this instance X3D is already a transformation option.

```
sce:AvailableFbxTransformation a owl:Class;
    rdfs:label "AvailableFbxTransformation";
    rdfs:comment "something that can be transformed to an Fbx file";
    rdfs:subClassOf sce:AvailableTransformation.

sce:FbxDistribution a owl:Restriction;
    owl:onProperty dcat:mediaType;
    owl:hasValue "application/autodesk.fbx";
    rdfs:subClassOf sce:AvailableFbxTransformation.
```

We then create a new ontology for the transformation rules above.

```
sce:AvailableFbxTransformation owl:equivalentClass
sce:AvailableX3DTransformation.
```

Since the converter can read and X3D and FBX files, both options are interchangeable hence we can use the equivalent class axiom.

Finally we integrate the new transformation rules with the AccuTrans transformation rules, in the General Purpose OWL Reasoner in exactly the same way as 4.1.2

```
class OwlReasonerAction {

	public static JsonObject main(JsonObject args) {

	LdModel asset = new LdModel(SauceVocabulary.prefixes)

	asset.read(new ByteArrayInputStream(args.toString().getBytes()), null,
"JSON-LD")

	asset.add(SauceVocabulary.vocabulary)
	asset.add(AccuTrans3DTransformationVocabulary.vocabulary)

	// add the I3DConverterTransformationVocabulary
	asset.add(I3DConverterTransformationVocabulary.vocabulary)

	JsonObject response = new JsonParser().parse(
asset.reason().json().toString() ).getAsJsonObject()

	log.info "Successfully completed reasoning"

	return response

	}

}
```

4.2.1 Demonstrating the results

Again an integration test, similar to the former, can be used to demonstrate the chained transformation options that are materialised as a result of the new transformation rules compounding the existing transformation rules:

```
testAccuTransWithi3DConverterTransformationCapabilities() {

    def asset = new LdModel(SauceVocabulary.prefixes)

    asset.add   """

        scei:test-asset a sce:Asset;
              dcat:distribution scei:test-asset-obj-distribution,
scei:test-asset-rtg-distribution.

        scei:test-asset-rtg-distribution dcat:mediaType
"application/autodesk.rtg"

    """
```

```
    def result = new LdModel(SauceVocabulary.prefixes)
    result.read(new ByteArrayInputStream(OwlReasonerAction.main( new
JsonParser().parse(asset.json()).getAsJsonObject() ).toString().getBytes()),
null, "JSON-LD")

    assertTrue result.ask("""scei:test-asset-rtg-distribution a
sce:AvailableFbxTransformation""")

}
```

In the test above we can see that the rtg is available as an fbx file since the new transformation service has advertised is capability to transform from X3D to fbx. A compound capability of being able to transform from rtg to fbx is now available since the former transformation service can write to X3D from rtg.

5 Conclusion

The examples provided in this document clearly demonstrate how transformation capabilities can be chained together, compounding the capabilities of different transformation services. Whilst the examples provided in this document are arbitrary, any number of classes of transformation can be created. It should be apparent that we can use the Description Logic provided in OWL2 to cover concepts not just around file formats, but also more abstract things such as action types, semantic descriptions and provenance. In fact it is possible to describe transformation options for all the concepts contained in the data architecture outlined in D4.1 and in the next deliverable we will describe in full how transformation capabilities can be applied to different asset types and different aspects of an asset.

The examples above demonstrate the materialisation of transformation capabilities at ingestion time, but in many cases it is more applicable to determine these at query time since there may be some implicit relationship to other assets which may not be decidable until a complete dataset of assets is provided. Furthermore, one of the goals of the framework is to ensure asset 'freshness', so it is imperative that transformation options are kept up to date, and this too will be demonstrated as part of the complete transformation framework.

6 Written references

D4.1 Smart Search Framework

7 Web references

RDF

https://www.w3.org/standards/techs/rdf

RDFS

https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

OWL2

https://www.w3.org/TR/owl2-overview/