Grant Agreement No.: 604656

Project acronym: NanoSim

Project title: A Multiscale Simulation-Based Design Platform for Cost-Effective CO_2 Capture Processes using Nano-Structured Materials (NanoSim)

Funding scheme: Collaborative Project

Thematic Priority: NMP

THEME: [NMP.2013.1.4-1] Development of an integrated multi-scale modelling environment for nanomaterials and systems by design

Starting date of project: 1st of January , 2014

Duration: 48 months

WP N°	Del. N°	Title	Version	Lead beneficiary	Nature	Dissemin. level	Delivery date from Annex I	Actual delivery date dd/mm/yyyy
1	8	Porto Data Storage and Adaptors	0.1	SINTEF	Report	PU	30	01/07/2016

1 Executive Summary

The Porto framework has a focus on semantic interoperability for the offline coupling of simulators, data analysis tools etc. In this lies the challenge of defining an abstract representation of data and context that does not enforce the need for syntactic standardization on file formats, protocols. Porto provides its own "*native*" storage back-ends. It currently supports MongoDB and HDF5. These back-ends allows for storing and retrieving any data that has a schema defined in terms of formal metadata. One challenge is the handling of simulators, which already have syntactic constraints, i.e. they are already using their own input and output formats. In these cases, to adopt to the Porto framework, it is necessary to create a wrapper-application that embeds its own file I/O operations together with the execution of the application. In addition, we will need to customized back-ends that are able to read and/or write the produced data formats and populate Porto-*entities* with content. This will allow other applications to access the available data without the need to know anything about the data's current format. See *Figure 1 - "Workflow component with offline coupling using Porto*". This illustrates a single component wrapper. A complete workflow will connect two or more of these components. The "*custom*" back-end deliveries in the NanoSim project are early versions of CHEMKIN-II, JSON-data and generic file storage. In addition to a Fluent UDF[1] code generator.

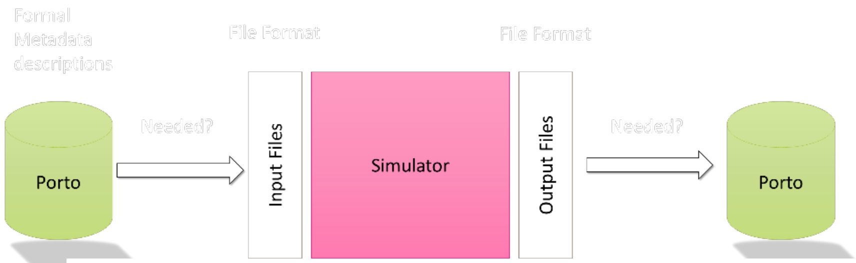

Figure 1 - Workflow component with offline coupling using Porto

2 NanoSim Storage System

2.1 Input from NanoSim consortium

The NanoSim consortium are currently not fully prepared to perform the specified interoperability steps. The main reason is that Porto has not been fully compliant with external storage abilities until recently. The other reason is the lack of work on the representation of the formal metadata needed. Resent feedback from the consortium implies that the current need for connections are as defined by this table:

Need to read input from Porto?	Formal Metadata Description (INPUT)	Input files format(s)	Simulator	Need to write output to Porto?	Output file format(s)	Formal Metadata Description (OUTPUT)
no			CPPPO	yes	JSON	As Specified

[1] User Defined Function

yes	As Specified	CHEMKIN-II, JSON, custom	ParScale	yes	JSON	As Specified
yes	Reference_entity	text,zip,reference	CFDEMCoup	yes	OpenFOAM, misc files, xy-data	OF_*, LIGGGHTS_dump
no			VASP	no		
no			REMARC	yes	CHEMKIN-II	VASP_*.json
No			FLUENT	no		

Note that this does not limit Porto's expendability to support more connections.

2.2 Porto storage API's

The architecture of the Porto storage system consist of three components:

- Storage Strategy - A storage strategy (see Strategy Pattern) implements the syntactic I/O specialization
- Data Model – A generic representation of metadata state
- Plugin Interface – A contract that allows Porto to load the back-end as a plugin

With this, it is possible to extend the palate of backend systems. Currently the Porto is supporting four of these storage systems; a MongoDB-backend, JSON-backend, HDF5-backend and the EXTERNAL-backend. The latter is a special case with is suitable to extend with custom file-format readers.

The current implementations of the IDataModel interface are described in Figure 2. As the Public Member Function illustrates, this is basically just a specialized key-value store. All data models are interchangeable, but serves different purposes. The implementations of the IStorageStrategy inteface in Figure 3 is closely related to the implementations of the IStrategyPlugin interface, as the strategy is the code of the plugin.

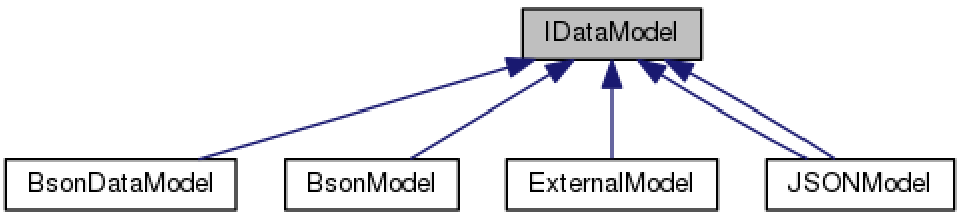

Figure 2 - Class diagram of IDataModel and derivatives

Public Member Functions

virtual **IDataModel** *	**createModel** ()=0

virtual bool	**appendDimension** (const char *, StdUInt)=0
virtual bool	**appendVariant** (const char *, StdVariant const &)=0
virtual bool	**appendString** (const char *, const StdString &)=0
virtual bool	**appendInt8** (const char *, StdInt8)=0
virtual bool	**appendUInt8** (const char *, StdUInt8)=0
virtual bool	**appendInt16** (const char *, StdInt16)=0
virtual bool	**appendUInt16** (const char *, StdUInt16)=0
virtual bool	**appendInt32** (const char *, StdInt)=0
virtual bool	**appendUInt32** (const char *, StdUInt)=0
virtual bool	**appendInt64** (const char *, StdInt64)=0
virtual bool	**appendUInt64** (const char *, StdUInt64)=0
virtual bool	**appendFloat** (const char *, StdFloat)=0
virtual bool	**appendDouble** (const char *, StdDouble)=0
virtual bool	**appendBool** (const char *, StdBool)=0
virtual bool	**appendInt32Array** (const char *, StdIntArray const &)=0
virtual bool	**appendDoubleArray** (const char *, StdDoubleArray const &)=0
virtual bool	**appendDoubleArray2D** (const char *, StdDoubleArray2D const &)=0

virtual bool	**appendDoubleArray3D** (const char *, StdDoubleArray3D const &)=0
virtual bool	**appendByteArray** (const char *, StdBlob const &)=0
virtual bool	**appendStringArray** (const char *, StdStringList const &)=0
virtual bool	**appendArray** (const char *, **IDataModel** const *)=0
virtual bool	**getDimension** (const char *, StdUInt &) const =0
virtual bool	**getVariant** (const char *, **StdVariant** &) const =0
virtual bool	**getString** (const char *, StdString &str) const =0
virtual bool	**getInt8** (const char *, StdInt8 &) const =0
virtual bool	**getUInt8** (const char *, StdUInt8 &) const =0
virtual bool	**getInt16** (const char *, StdInt16 &) const =0
virtual bool	**getUInt16** (const char *, StdUInt16 &) const =0
virtual bool	**getInt32** (const char *, StdInt &) const =0
virtual bool	**getUInt32** (const char *, StdUInt &) const =0
virtual bool	**getInt64** (const char *, StdInt64 &) const =0
virtual bool	**getUInt64** (const char *, StdUInt64 &) const =0
virtual bool	**getFloat** (const char *, StdFloat &) const =0
virtual bool	**getDouble** (const char *, StdDouble &) const =0

virtual bool	**getBool** (const char *, StdBool &) const =0
virtual bool	**getInt32Array** (const char *, StdIntArray &) const =0
virtual bool	**getDoubleArray** (const char *, StdDoubleArray &) const =0
virtual bool	**getDoubleArray2D** (const char *, StdDoubleArray2D &) const =0
virtual bool	**getDoubleArray3D** (const char *, StdDoubleArray3D &) const =0
virtual bool	**getByteArray** (const char *, StdBlob &) const =0
virtual bool	**getStringArray** (const char *, StdStringList &) const =0
virtual bool	**getArray** (const char *, **IDataModel** *) const =0
virtual **IDataModel** *	**getModel** (const char *) const =0
virtual bool	**appendModel** (const char *, **IDataModel** *)=0
virtual void	**setId** (const StdString &id)
virtual void	**setMetaName** (const StdString &metaName)
virtual void	**setMetaVersion** (const StdString &metaVersion)
virtual void	**setMetaNamespace** (const StdString &metaNamespace)
virtual StdString	**id** () const
virtual StdString	**metaName** () const
virtual StdString	**metaVersion** () const

virtual StdString	**metaNamespace** () const

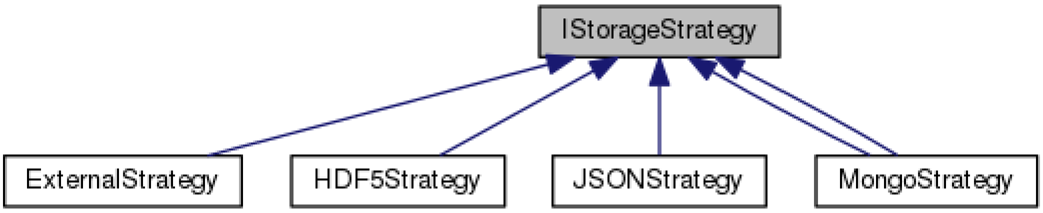

Figure 3 - Class diagram of IStorageStrategy and derivatives

Public Member Functions

	IStorageStrategy (char const *uri, const char *options=nullptr)
	IStorageStrategy (IStorageStrategy const &)=delete
IStorageStrategy &	**operator=** (IStorageStrategy const &)=delete
virtual const char *	**metaType** () const =0
virtual IDataModel *	**dataModel** () const =0
virtual void	**store** (IDataModel const *)=0
virtual void	**startRetrieve** (IDataModel *m) const =0
virtual void	**endRetrieve** (IDataModel *) const =0

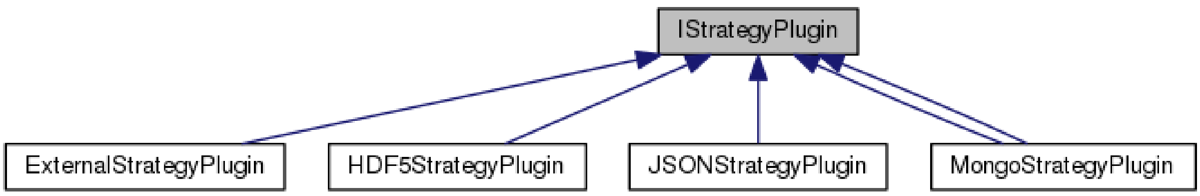

Figure 4 - Class diagram of IStrategyPlugin and derivatives

Public Member Functions

virtual void	**registerStrategy** ()=0

2.3 External Plugins API

External plugins are user defined libraries that anyone can implement to support a new custom file format. The only prerequisite to defining these libraries are knowing which formally defined entities to support (in addition to knowing how to map this to the contents of the external file format). The key functions are the load and save functions which are used to either populate a datamodel (softc_datamodel_t is the C-API equivalent to the IDataModel interface in C++). There is a struct contract between the definition of the metadata and how this datamodel should be populated. How the actual reading and writing of the data is done is up to the programmer.

The current version 1 of the C-API is defined in <softc/softc-storage-plugin.h>:

```
#define SOFTC_CAPABILITY_NONE 0
#define SOFTC_CAPABILITY_READ 0x01
#define SOFTC_CAPABILITY_WRITE 0x02

#define SOFTC_STATUS_OK 0
#define SOFTC_STATUS_FAILURE 1

int softc_plugin_identify( char* name, int maxlen );
int softc_plugin_capabilities();
int softc_plugin_load( softc_datamodel_t* datamodel, const char* uri, const
 char* options );
int softc_plugin_save( const softc_datamodel_t* datamodel, const char* uri,
 const char* options );
```

Implementing external plugins requires that the specific Formal Metadata Schema is known (this will dictate what is to be expected from the contents of the *datamodel*.

2.4 External Plugins

In this section the currently supported external plugins implemented in Porto is described. The most important attributes of these plugins are its representation through formal metadata schemas.

2.4.1 Chemkin-II

Porto now have a support for reading the CHEMKIN-II format through a provided chemkinReader. The output from REMARC can be read into the following entity:

Name	Namespace (context)	Version

chemkin_reaction	eu.nanosim.vasp	0.1

Description

Description of a thermodynamical reaction with rate constant: k(T) = A * T**b * exp(-Ea/(R*T)) where A, b and Ea are parameters, T the temperature and R the molar gas constant (8.31451 J/(mol K)). See http://www.frad.t.u-tokyo.ac.jp/public/chemkin/CKm_inp.html.en for more details.

Dimensions

Name	Description
nreactants	Number of reactants (Chemkin requires 0 < nreactants < 4).
nproducts	Number of products (Chemkin requires 0 < nreactants < 4).
ntroe	Number of parameters for evaluating the pressure dependence using Troe's formula. Can be 0 (not used), 3 or 4.
nenhancement_factors	Number of enhancement factors. Zero indicates that they are not used.
nplog	Number of intervals the pressure dependency of the rate coefficients is described. May be zero for no pressure dependency.

Properties

Name	Type	Dims	Description
reactants	string_list	[nreactants]	Name of each reactant species.
products	string_list	[nproducts]	Name of each product species.
third_body	bool		Whether the reaction occurs in precense of catalytical third-body (e.g. a surface).
A	double		Preexponential factor in the rate constant. [FIXME define the unit. As formulated in the documentation of the CHEMKIN II file format, it depends on b and the reaction order... consider use a saner expression for the reaction constant for this entity.]

b	double		Parameter in the rate constant, see entity description.
Ea	double		Activation energy.
A_low	double		Preexponential factor for the low-pressure limit. Support fillvalues. FIXME: define the unit.
b_low	double		Value of b in the low-pressure limit. Support fillvalues.
Ea_low	double		Activation energy in the low-pressure limit. Support fillvalues.
troe	double	[ntroe]	Parameters a, T***, T* and T** when using Troe's formula to express pressure dependency. The last parameter T** is optional.
enhancement_species	string_list	[nenhancement_factors]	Name of species in the buffer gas corresponding to the enhancement_factors.
enhancement_factors	double	[nenhancement_factors]	Enhancement factors for describing dependency of the rate parameters on the buffer gas.
P_plog	double	[nplog+1]	Pressures defining the borders of the nplog pressure intervals for defining pressure dependency of the rate constant. Should be increasing.
A_plog	double	[nplog]	Preexponential factors for pressure dependency of the rate constant. FIXME: define the unit.

b_plog	double		Values of b for pressure dependency of the rate constant.
Ea_plog	double		Activation energy for pressure dependency of the rate constant.

In addition to this the following supported Entites are defined:

Name	Version	Namespace
kmos_model_parameters	0.1	eu.nanosim.vasp
kmos_meta	0.1	eu.nanosim.vasp
kmos	0.1	eu.nanosim.vasp
kmos_fillvalues	0.1	eu.nanosim.vasp
kmc_process_parameters	0.1	eu.nanosim.vasp
kmc_process	0.1	eu.nanosim.vasp
kmc_layers	0.1	eu.nanosim.vasp
dftdata	0.1	eu.nanosim.vasp
chemkin_thermo	0.1	eu.nanosim.vasp
chemkin_reaction	0.1	eu.nanosim.vasp
chemkin_meta	0.1	eu.nanosim.vasp
chemkin	0.1	eu.nanosim.vasp
chemkin_fillvalues	0.1	eu.nanosim.vasp

2.4.2 XY-plot

XY-plots are supported by using this Entity:

Name	Namespace (context)	Version	
XY	sintef	0.1	
Description			
Dimensions			
Name	**Description**		
I	undefined		
J	undefined		
Properties			
Name	**Type**	**Dims**	**Description**
data	double	[I,J]	

2.4.3 Reference

File references are useful where we want to propagate information about static files on a given location. This can be very large data that isn't practical to store in a duplicated form.

Name	Namespace (context)	Version	
reference	http://info.emmc.eu	1.0-SNAPSHOT-1	
Description			
Generic data source reference entity			
Properties			
Name	**Type**	**Dims**	**Description**
uri	string		Resource locator
created	string		Date and time of the resource creation
owner	string		Owner of the resource
lastModified	string		Date and time when the resource was last modified
sha1	blob		SHA-1 hash checksum

2.4.4 File

Occasionally we want to store a file (binary or text) in the database as-is. This can be documents, compressed files, videos etc. In this case we can use the defined File entity which stores everything as a binary large object (BLOB)

Name	Namespace (context)	Version	
file	http://info.emmc.eu	0.1-SNAPSHOT-1	
Description			
Entity to represent files in			
Properties			
Name	**Type**	**Dims**	**Description**
filename	string		Name of file
suffix	string		The suffix of the file
size	int64		Size of file
data	blob		The file contents

2.4.5 JSON (Proprietary Schema)

We have input from ParScale on the JSON structure that ParScale creates after simulating. A simplified JSON writer is demonstrated, but due to the lack of formal metadata schemas we have not yet been able to complete this backend.

2.5 ANSYS Fluent UDF Support

A demonstration of the generation of ANSYS Fluent is available in Porto. It uses the SOFT.MVC framework to generate the context of prewritten UDF file.

3 Conclusion

The Porto Storage System does now support custom file support. With the provided C-API anyone can now create their own custom backend and have Porto use it. We have demonstrated the flexibility of how to use the storage system through the different options of storing and reading raw files as-is, storing and reading references to files, storing and reading data as entities in the "*internal*" storage and lastly storing and reading from proprietary file formats through the use of the *external* data storage plugins.