Data Model Roadmap

CCSI Element 5

Editors: D. Gunter, D. Olson

Disclaimer

1. Motivation

The primary purpose of the data model is to assure integrity of results from using the CCSI simulation suite. It does so by defining standard representations, vocabularies, and models where they are needed and, in areas where the data representations and formats are already defined by existing applications, by documenting existing definitions.

Here are a few excerpts from the Project Summary of CCSI goals that illustrate the importance and capabilities of the data model and data management system:

- produce complete and consistent simulations, which allow seamless migrations among multi-physics models at different scales.

- The CCSI Toolset will ensure that all the simulations within its framework use consistent data formats (e.g., the same thermophysical properties are used in models at various scales) with assured data provenance.

- The CCSI Toolset will ensure the consistency of models across all scales. … One of the objectives of CCSI is to develop a common platform that allows models to readily exchange information in a computationally efficient manner, enabling the development of complete (knowledge preserving) and consistent models.

Figure 1 illustrates the range of CCSI modeling and data acquisition activities. Often these activities will be carried out by different groups of people, so it is important that the data sharing between the activities is not folklore, but well-defined and reproducible. This is accomplished by, for example, specifying consistent use of units, annotating stored data, tagging of related elements, and clear format specifications between simulation elements and the storage system. This also includes adopting applicable industry standards, such as CAPE-OPEN, and detailed specification of interfaces in areas where standards do not exist.

1.1. Background

The Data Model must take into account existing data models, tools and technologies. This section describes those tools and technologies deemed relevant, due to alignment with CCSI needs, adoption by current CCSI collaborators, or both.

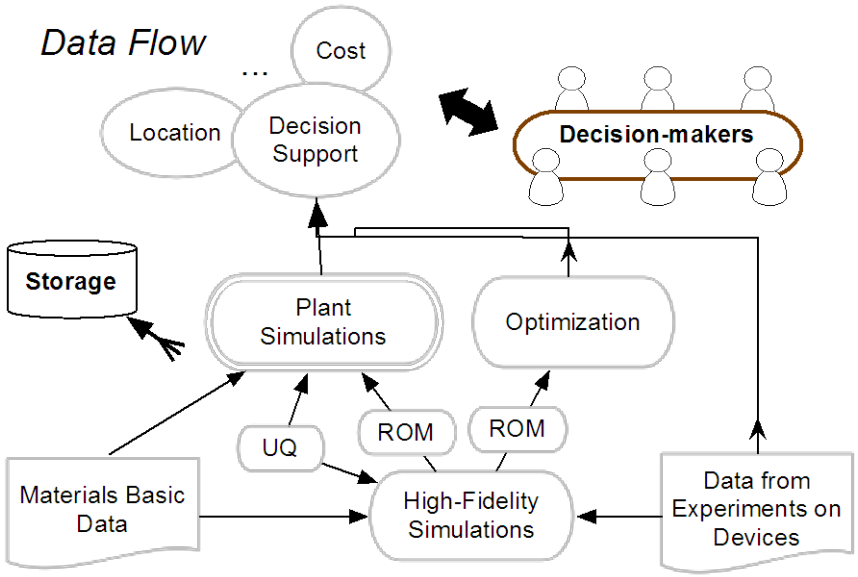

Figure 1. CCSI Data Flow

1.2. Tools

The tools currently used by CCSI fall into two general categories: simulation and modeling, and data management. Simulation and modeling tools include: Aspen Plus, Fluent, and MFiX, each of which is described below.

Simulation Tools

The simulation tools used within CCSI consist of a mix of commercial products and those developed in the R&D and open source communities. The ones listed below are just a sampling of those that have been used so far in CCSI and many more tools are expected to be used.

Aspen Plus is a process modeling tool from Aspen Technologies Inc. that is used in CCSI for plant process optimization and supports the CAPE-OPEN interface specification (see below for a description of CAPE-OPEN) and a COM interface. This COM interface is used to control execution of Aspen Plus in the CCSI integration framework. Aspen Plus has numerous files in proprietary formats used during execution. The backup format (.bkp) for the flowsheet is used by the CCSI framework since it is a portable ASCII format and is independent of the version of Aspen being used. Additional configuration information for a run and the results are handled via the CCSI integration framework.

Fluent is a CFD (computational fluid dynamics) code from ANSYS, Inc. and will be used in CCSI for high-fidelity modeling and simulation of specific components of CC plants.

MFIX is a CFD code developed at NETL that is used for high-fidelity simulation and modeling of specific components of CC plants.

PSUADE is an uncertainty quantification (UQ) tool developed at LLNL (related, but separate from SNL's DAKOTA tool) that is being used by the UQ team.

Data Management Tools

The data management tools specific to CCSI are EKM and Velo. More general data management tools include relational database management systems and non-relational databases such as MongoDB. The ones listed below are just a sampling of those that may be used in CCSI.

EKM (Engineering Knowledge Manager) from ANSYS, Inc. provides a tightly coupled data management solution for the suite of tools provided by ANSYS. While it has an API for integration with other tools we find that the somewhat rigid structure and licensing cost makes it a poor choice for a general purpose data management solution for CCSI. We do recognize, however, that some companies using the CCSI framework may choose to use EKM for parts of their storage service and the CCSI framework should accommodate that.

Velo is a flexible, foundational, collaborative technology that can be used in modeling and simulation projects to:

- Capture, organize, query, and share experimental and observational data along with the tacit knowledge that is used to develop computational models
- Provide versioning of model inputs for specific projects and provenance for simulation results
- Enable simulations to be launched on remote computational platforms
- Support both tight and loose integration of 3rd party tools to facilitate various modeling activities such as model development and visualization

Velo is a knowledge management platform for modeling and simulation. A knowledge management system (KMS) is an information and communication technology system to support and enhance explicit and tacit organizational knowledge. The Velo platform is designed around a novel integration of a collaborative Web-based environment and a scalable enterprise content management system, augmented by a set of APIs that support extension for new data types and tools. Importantly, by leveraging and reusing proven, open source infrastructure technologies at Velo's core, it may be customized for CCSI modeling needs.

RDBMS. The basic features and benefits of RDBMSes such as Oracle, DB2, MySQL and PostgreSQL, are well known. Therefore, we only note here the less-obvious fact that RDBMS products are less effective with a large and *dynamic* set of data models. The reason is, put simply, that without large amounts of effort building and constantly evolving schema definitions, most of the internal structure of the data remains opaque to the sophisticated indexing and querying capabilities of the tools. The degree of this constriction depends of course on the degree and type of dynamism in the system.

NoSQL. The features of non-relational (so-called "NoSQL") data stores such as Cassandra, CouchDB, and MongoDB are less well known. For these data stores, there is much more variation in interface and functionality between products than for the RDBMSes. But, generally speaking, they tend to trade a record-at-a-time interface (no "joins") for easier horizontal scalability, offer less in the way of transactional guarantees, and do not use (or require) a pre-defined schema for the data. Unlike the RDBMS products they do not share a common query language. This schemaless (what Michael Stonebraker likes to call "schema last"), approach has the advantage that it more easily adapts to new or changing data types; though depending on the use-case it may demand an equivalent extra amount of application-level verification.

1.3. Technologies

There are several technologies in use within existing CCSI components. The most important of these are CORBA, COM, JSON and JSON-SCHEMA, and CAPE-OPEN. A description of each is given below.

Common Object Request Broker Architecture (CORBA). OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Although CORBA has been largely superseded in new applications by Microsoft COM/.NET for Windows-only applications, and HTTP/JSON elsewhere, there are still many legacy applications which use it as a primary interface for control and data. One example is the ANSYS toolset which uses CORBA for data interchange between components. For more information, see http://corba.org/.

Component Object Model (COM) and .NET. Microsoft technologies COM/COM+, and .NET enable Microsoft Windows-family of Operating Systems software components to communicate. One can think of it as Microsoft-specific technologies for remote object invocation in the style of CORBA. For more details, see http://www.microsoft.com/com/.

XML. eXtensible Markup Language is designed to transport and store data in a self-descriptive format. The mark-up is similar to HTML and uses tags to define and delineate data. XML's wide usage has led to a proliferation of editors and readers.

JavaScript Object Notation (JSON). JSON is a lightweight data-interchange format that has quickly become the best practice for data exchange in web-based applications, and is highly popular outside this context as well. JSON is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. However, like XML, JSON is a text format that is completely language independent. Unlike XML, it is easy for humans to read and write and it is easy for machines to parse and generate. JSON's formatting conventions are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. For more details, see: http://json.org/. Related to JSON is BSON for binary-encoded JSON-like documents, see http://bsonspec.org.

JSON-Schema. JSON defines an interchange format for all data models, whereas JSON Schema can be used to define the structure of a particular data model. Details are in an IETF draft, published at http://tools.ietf.org/html/draft-zyp-json-schema-03. In the words of this draft, JSON Schema "provides a contract for what JSON data is required for a given application and how to interact with it. JSON Schema is intended to define validation, documentation, hyperlink navigation, and interaction control of JSON data."

CAPE-OPEN. Published at www.colan.org. "CAPE-OPEN standards are the uniform standards for interfacing process modelling software components developed specifically for the design and operation of chemical processes. They are based on universally recognized software technologies such as COM and CORBA."

This standard consists of a set of inter-related documents and software covering nearly all aspects of process modeling. Of particular relevance for the CCSI data model are the specifications included in the "Physical Properties Data Bases Interfaces" document, including

notation for units, chemical and compound identifiers based on CAS and naming for equations of property models.

The suite of activities in CCSI goes well beyond process modeling but we can take advantage of this standard when appropriate, many of the commercial tools support aspects of CAPE-OPEN.

1.4. Future tools and technologies

The CCSI Data Model must be flexible enough to accommodate new tools and technologies. Some anticipated tools and technologies include Matlab and R for analysis, CFD codes MFIX and Fluent, and other commercial codes such as gProms, GeoTherm, and COMSOL.

2. Approach

This section describes the general considerations, tools, and techniques in our technical approach to designing and implementing the Data Model.

2.1. CCSI Integration Framework

We divide CCSI/E5 functionality into five layers, numbered Layer 0 through Layer 4. Figure 2 shows the functionality and arrangement of these layers. Each layer is itself subdivided into a set of *interfaces* for associated *services*. At the top of the diagram is Layer 4, the highest level of consumers of the CCSI outputs: Browsing/Analysis and Model Development. Browsing refers to open-ended exploration of data from multiple layers in the stack; Analysis, similarly, may consume data from Layers 1 to 3. Model Development refers to models at all levels of granularity: the models for computational fluid dynamics (CFD), reduced-order models (ROMs), steady-state and dynamic models of power plants, etc.

Layer 3 provides standardized interfaces to services for Uncertainty Quantification (UQ) and the creation of ROMs (ROM Builder). For the purpose of model development, these interfaces may serve as the proxy to the standardized simulation interfaces, or the interaction may occur more directly with the Gateway to the simulation. This is indicated by the interface from Model Development to both Layers 2 and 3.

Layer 2 is the standardized simulation interface, also called the Sinter Gateway. This includes the ability to glue together multiple simulation codes in a workflow. To this point, much of the work in Element 5 has focused on defining this interface to allow the Layer 3 ROM/UQ services to interact with the simulations.

Layer 1 is the interface to the simulation codes, such as MFIX and Aspen Plus. This includes native COM/CORBA interfaces.

Finally, Layer 0 is a cross-cutting service layer for data management and storage. It allows services at Layers 1-4 to use common mechanisms to find, register, and annotate data sets. Much of the data used in CCSI will consist of files in proprietary formats of the various tools and, in these cases, the important aspect of the data model is to characterize those files and store metadata with them in the storage system so the versioning and dependencies on particular versions of tools is captured and maintained.

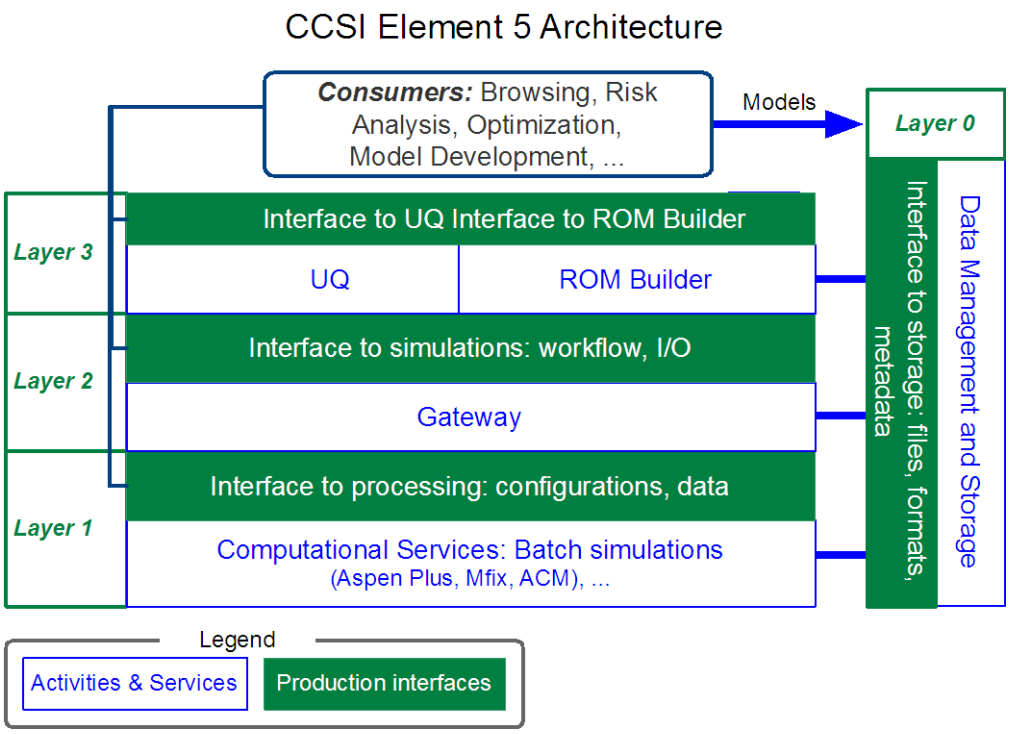

Figure 2. CCSI integration architecture.

Most components of our approach are applicable to multiple layers in the architecture.

2.1.1 Development Process

We will use development processes called “agile” development, which is a group of software development methodologies based on iterative and incremental development. The canonical definition of agile principles is in the *Agile Manifesto*: (1) Individuals and interactions over processes and tools, (2) Working software over comprehensive documentation, (3) Customer collaboration over contract negotiation, and (4) Responding to change over following a plan.

In this context, following the principles of agile development mean that we will not attempt to identify and document all requirements up-front, nor will we develop a complete data model before attempting to deploy working software. Rather, we will start by working one-on-one with other members of the CCSI project and developing prototypes to implement relevant parts of the data model on short (*e.g.*, two-week) cycles. In this way, we will gain experience with technologies and methodologies early, and continuously, while delivering working software at every stage in the process.

2.2. Data representations

Data is represented in the most standard and convenient format for existing consumers. For example, an Aspen Plus process’ flowsheet model will use Aspen’s “backup” file format since that is most portable and independent of versions of Aspen Plus.

Where new representations must be invented, we will use JSON unless there is a strong reason not to (e.g. anticipated consumers for comma-separated values). For example, the Layer 0 interface that queries the EFRC database uses JSON, defined by JSON-Schema.

2.3. Controlled vocabularies

A controlled vocabulary is a standard set of terms and definitions that may include synonyms. Examples include service categories in the Yellow Pages and the subject headings in the Library of Congress. We propose to pursue a bottom-up effort to first standardize the terms used within a limited scope and then merge related vocabularies. An output of this process will be logic to interpret and translate terms from external tools into the controlled vocabulary (and vice-versa). An example of a cross-cutting controlled vocabulary are measurement units, for which we are using the UDUnits library.

2.4. Provenance

Provenance is concerned with recording and tracking the creation of data products. Often this is solved within the smaller scope of a single application by using a particular naming convention for data files. But this solution makes the metadata hard to modify, search, integrate, and move. Our approach to this problem will be to identify data items uniquely (*e.g.*, with a UUID) and then register that unique identifier in a datastore along with meta-data about the application that created it, its location(s), etc.

2.5. Remote access model

Remote access to the data should use standard mechanisms. This includes JSON over HTTP, but some tools have their own remote access methods (COM, .Net) that can be leveraged. The storage management system will also provide built-in remote access mechanisms.

New interfaces will be designed and built based upon the REST principles[1], which provide a scalable and reliable interface between the clients (often a web browser) and the servers, and accommodates intermediate network features such as proxy servers and security services.

2.6. Data verification

The data model must support data verification, particularly with data that is replicated or summarized. It is important to know that different copies of the data "agree", in some scientifically meaningful sense. Attributes of the data model should allow discovery of all extant copies of the original data, and verification of agreement between them.

2.7. Data Lifecycle Management

The lifecycle of a data product -- from creation, through transformation(s), consumption, archival, and deletion – should be available to CCSI components. Similarly, the related actions performed by the software components to achieve a modeling result, which both generates and consumes multiple data products should be discoverable and accessible. For example, a single calculation of thermodynamic properties of a carbon compound would involve retrieval of crystal structure, calculation of the property, and storage of the result. Our approach to

[1] http://en.wikipedia.org/wiki/Representational_state_transfer

providing this functionality will be to add *instrumentation* to uniquely identify the activities, data products, and links between them. In addition to the instrumentation libraries, CCSI/E5 will develop data models to describe this information.

2.8. Storage model

The details of the CCSI storage model are still under consideration. We have identified the following base requirements:

- **Location.** Given the scope of activities encompassed by CCSI and the end uses of the CCSI Toolset we expect there to be a number of locations and storage systems for various types of data used by CCSI.
- **Security.** It is anticipated that some data is likely available only locally to certain researchers based on the IP security requirements for some proprietary data.
- **Provenance.** A goal of the CCSI storage service will be to handle persistent references to the distributed data stores and items so that the provenance of the results of simulations is maintained.

2.9. Security attributes

Data, programs, documents, and other data used in CCSI will have Intellectual Property (IP) constraints regarding who has access and how it can be used. Users of the completed CCSI Toolset will introduce additional IP constraints. While the IP protected data sits in local storage under control of the data owner it is not necessary for the CCSI framework to provide any additional access control mechanisms. For data and documents with IP restrictions that are put into the CCSI storage service then the storage service must be able to support the necessary access control mechanisms. The aspect of the data model involved with this feature is that the storage service needs to support attributes of stored data that can be used to make the decisions about who is able to access a protected item.

3. Uncertainty Quantification Use-Case

This section walks through a use-case for Uncertainty Quantification in the CCSI framework in some detail. The major components are, in terms of the architecture in Figure 2: the UQ GUI and PSUADE at Layer 3, the Sinter Gateway (or simply Gateway) at Layer 2, and the Aspen Plus simulation tool at Layer 1.

These terms are used to indicate relationships to the data modeling tasks:

CCSI data model. If an item has/needs a *CCSI data model,* then the item should be represented and handled in a standard way by the CCSI framework. Usually, it should be represented in JSON or BSON and described with a JSON-Schema that is stored in the CCSI subversion repository. In addition to standard representation, most items should be identified and tracked by the CCSI Data Management Service.

CCSI interface. All CCSI interfaces should be documented in the subversion repository. Usually they will be RESTful HTTP interfaces.

An overview of the use-case is shown in Figure 3. The user starts by generating the simulation, which is saved in the back-end database of the Gateway. Then the user uses the UQ GUI to choose the simulation (from all available ones), get a set of runs from PSUADE, and submit the

runs to the Gateway for processing on EC2. When some or all of the runs have completed, the UQ GUI can request the results from the Gateway and feed them back to PSUADE to generate an analysis graph. This analysis graph is displayed to the user.

Figure 3. Overview of UQ Use-Case

3.1. Generate the Simulation

An overview of the simulation generation process is shown in Figure 4. The detailed steps are below.

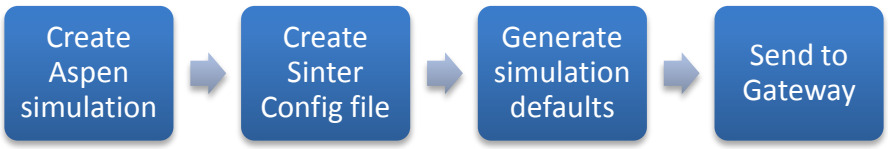

Figure 4. Overview of Simuation Generation

1. A modeler creates an Aspen simulation, and saves a Backup (.bkp) file.
    - The backup file (though not in JSON) is a CCSI data model
2. The modeler uses the Aspen variable explorer to determine which input/output variables are of interest.
3. The modeler creates a Sinter config file (by hand)
    - The Sinter config file is a CCSI data model which conforms, in syntax, to a general JSON-Schema (for all sinter config files).
4. The modeler uses *DefaultsGenerator*, which takes the sinter config and Aspen .bkp, and generates a JSON file of defaults for the simulation.
    - The defaults file is a CCSI data model, and should conform to the same JSON-Schema as the sinter config file from #3, above.
5. The modeler uses the HTTP REST API to send the simulation to the Gateway.
    - The REST API is a CCSI interface.
6. The gateway creates a new Simulation (adding it to the Gateway's RDBMS).
7. The modeler submits the .bkp file to the new Simulation.
    - This operation uses the Gateway CCSI interface.
8. The modeler submits the sinter config file to the new Simulation.
    - This operation uses the Gateway CCSI interface and Sinter Config data model.
9. The modeler submits the defaults JSON file to the new Simulation.
    - This operation uses the Gateway CCSI interface and Defaults data model.

3.2. Perform UQ Runs

An overview of how the user uses the UQ GUI and Gateway to run *Aspen+* simulations on EC2 is shown in Figure 5. The detailed steps are below.

Figure 5. Overview of Performing UQ Runs

1. The user starts the UQ GUI.
2. The GUI queries the gateway for a list of simulations available to this user.
    - The query uses the Gateway's CCSI interface
3. The user chooses a simulation from the list.
4. The GUI downloads the selected simulation's sinter config file from the gateway.
    - The sinter config file is a CCSI data model.
5. The GUI parses the config file to extract the list of input and output variables.
6. The user selects inputs, outputs, ranges, distribution, and sample generation algorithm.
7. The GUI generates a PSUADE input file from the user's selection.
    - The PSUADE input files (though not in JSON) are CCSI data models.
8. The GUI runs PSUADE.
9. PSUADE generates a set of samples based on the input file.
    - The PSUADE samples are a CCSI data model.
10. The GUI reads the set of samples, and displays it to the user.
11. The user clicks "Run" to run the samples.
12. The GUI generates a TurbineClient config file.
13. The GUI calls the TurbineClient scripts.
14. The Turbine client requests a new session on the gateway. This returns a *sessionID*.
    - This uses the Gateway's CCSI interface.
15. The TurbineClient uses the PSUADE file and TurbineClient config information to generate a set of runs for the gateway.
    - A "run" is a CCSI data model (currently a JSON file), where each run includes Simulation name, Inputs, etc.
16. The TurbineClient sends the runs (JSON file) to the Gateway, using the sessionID.
17. The Gateway creates a new Job in the database for each submitted job.
18. The TurbineClient requests the gateway move the newly created jobs to the submit queue.
    - This operation uses the Gateway's CCSI interface.
19. The Gateway consumer is alerted to the jobs on the submit queue.
20. The gateway consumer begins launching EC2 consumer nodes in relation to how many jobs are on the submit queue.
21. A consumer spins up on EC2, and requests a job to run from the gateway.
22. The consumer receives the job's input data: Aspen .bkp file, sinter config, defaults JSON file, and other parameters.
    - All these files are CCSI data models
23. The consumer starts Aspen Sinter, and provides the path to the simulation files.
    - The interface to Aspen/Sinter is a CCSI interface

24. Aspen Sinter: starts Aspen with the .bkp file, reads the defaults file and inputs file., merges in the inputs to the defaults, inserts the inputs+defaults into Aspen, then runs the Aspen simulation to completion.
25. Aspen Sinter extracts the outputs and errors.
    - The simulation result (outputs + status) is a CCSI data model.
26. The consumer returns the result (JSON) to the Gateway, with the return code. (Success or Error.)
27. The Gateway saves the result.
28. Goto Step #21 until all jobs complete.
29. The GUI: periodically checks how many jobs remain uncompleted by querying the gateway, and displays a status bar.

3.3. Analysis of UQ Results

This section describes the collection and analysis of results from the UQ runs in 3.2. An overview of the process is shown in . The detailed steps are below.

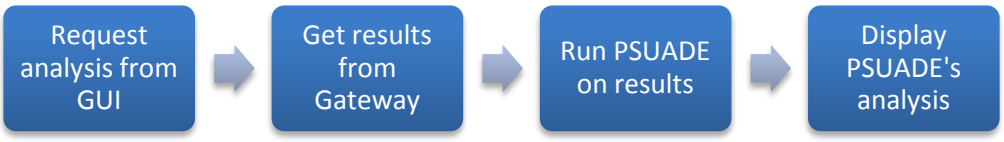

Figure 6. Overview of the Analysis of UQ Results

1. When the user requests an analysis from the GUI, the GUI requests TurbineClient retrieve the results.
2. TurbineClient requests all the results of a session from the Gateway. (The session may not be complete.)
    - The sessionID is part of the CCSI data model.
3. The Gateway collects all the results.
4. The Gateway returns the results
    - The results (a JSON Array) are a CCSI data model.
5. TurbineClient uses the information from the original PSUADE input file and TurbineClient config to save the results to a new PSUADE input file.
    - The PSUADE input files are a CCSI data model.
6. The GUI runs PSUADE on the resulting PSUADE file, and requests an analysis.
7. PSUADE generates an analysis graph (in Matlab format).
8. The GUI displays the graph.

3.4. Summary of CCSI Data Models and Interfaces

In previous sections, the steps involved in running a UQ analysis in the CCSI framework were described in detail. The steps were annotated with data models and interfaces. Table 1 provides a summary of these data models and interfaces.

Table 1. Summary of CCSI Data Models and Interfaces for UQ Use-Case

Data Model	Description	Format	Component	Section
AspenBackup	Aspen Plus backup file (.bkp)	ASCII, Aspen Plus native	Aspen Plus	3.1-1
SinterConfig	Variables of interest for a simulation	JSON	Sinter/Aspen-Plus	3.1-3
SinterDefaults	Defaults for simulation	JSON	Sinter/Aspen-Plus	3.1-4
PsuadeInput	Input file for PSUADE algorithms, describing inputs, outputs, ranges, distribution, and sample generation algorithm	PSUADE format (ASCII)	PSUADE, UQ GUI	3.2-7
PsuadeSamples	Samples used to define a set of UQ runs	JSON (maybe)	PSUADE, UQ GUI	3.2-9
GatewayRun	Job run by the Gateway. Each run includes simulation name, inputs, etc.	JSON	(Sinter) Gateway	3.2-15
SinterResult	Outputs and errors from a simulation run.	JSON	Aspen/Sinter, Gateway	3.2-25
GatewaySession	sessionID and other metadata to identify a "session" of runs.	JSON	Gateway, TurbineClient	3.3-2
SessionResult	Set of results from all runs in a Gateway session. A collection of SinterResult-s plus other metadata.	JSON	Gateway, TurbineClient	3.3-4
Interface	Description	Protocol	Component	Section
GatewaySimulation	Create, run, monitor, and retrieve results of simulations.	HTTP (REST)	Gateway	3.1-5
AspenSinter	Run Aspen Plus simulations.	HTTP (REST)	Aspen Plus	3.2-23

4. Existing definitions

This section describes components of the data model that have already been defined within CCSI. The components are sorted by interface Layer (see Figure 2), from highest to lowest. Each component describes use-cases and existing models as well as CCSI standard Data Models.

4.1. ROM builder (Layer 3)

CCSI will use multiphase computational fluid dynamics (CFD) to simulate the device scale performance in terms of flow properties, outlet composition, temperature and pressures. The CFD predicted device scale performance under different operating conditions will be used in creating device scale Reduced-Order Models (ROMs) for the subsequent tasks in process synthesis, control and uncertainty quantification.

The current ROM development work uses the MFIX CFD code and targets Aspen Plus as the process modeling tool for executing the ROMs.

4.1.1 Use-Cases

The following are the original use-cases proposed for the ROM data model.

- Status of running simulation based on ROM ID.
- Summary of design space used to run the simulation.
- Individual query for sampling method, interpolation method and cross validation method used to run the simulation.
- Machine information identifying where the simulation ran.
- Multiple simulations run for a single model.
- Comparative data of simulations run using same input and output space but different sampling and interpolation methods.

4.1.2 Data Files

The files needed for the ROM builder include the input and output files for MFIX cases and the input and output files for the ROM builder itself. For each MFIX case, there is at least one input file called "mfix.dat". If the computational mesh is generated by a CAD file, the CAD file called "geometry.stl" is required. If the model requires user defined Fortran, the Fortran source code can also be treated as input file. The output files of MFIX include files with extensions "RES", "SP1", "SP2",..., "SP9", "SPA", "SPB","vtk", "OUT", and "LOG". Those files are needed for ROM builder. Details of the files and formats used for MIFX are documented at https://mfix.netl.doe.gov/documentation.php.

The input files for the ROM builder itself are not finalized yet. However, we need user inputs for the selected parameters and their default, lower and upper limit values, an input file for post-processing the MFIX results, and the input for Latin Hypercube Sampling. The output file of the ROM model may include a file for yROM and a file for xROM. We may also combine them to one output file.

Since multiple MFIX cases need to be run on multiple machines, we need a good data model to store those files. The ROM builder needs to read the MFIX output files either in the same directory or in multiple directories with directory name numbered in sequence. Note that the

"SP*" and "vtk" files for one case are named in sequence by MFIX based on residence time since MFIX solves a unsteady (transient) flow problem.

4.1.3 Data Store

The TS5 team has designed a relational schema to store the ROM builder input and output data in a more general and accessible form, shown in Figure 7. Access to the data in this schema will be standardized as we make progress towards accessing these components from Layer 4 simulations.

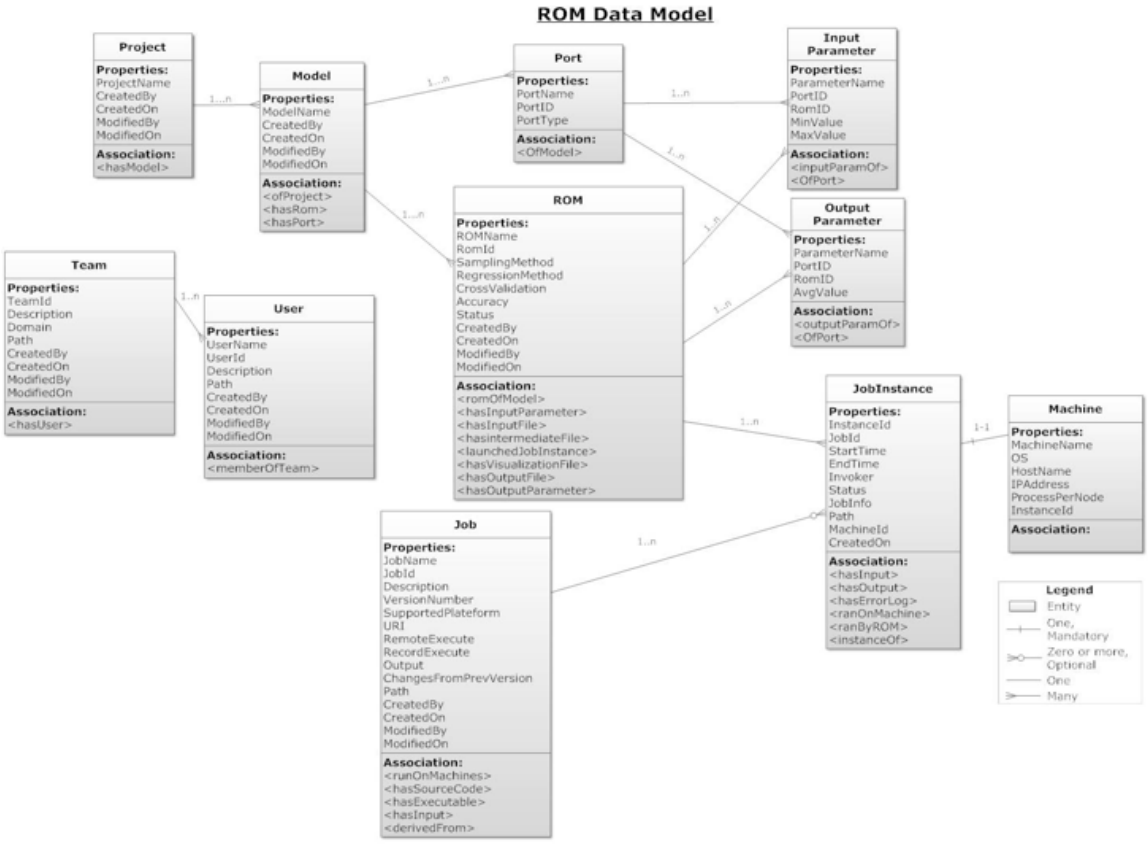

Figure 7. Relational schema for ROM builder.

4.2. Uncertainty Quantification (Layer 3)

The initial framework development for performing Uncertainty Quantification (UQ) analysis connects the uncertainty analysis software with the process simulation software. An MEA process model (in Aspen Plus) is the initial simulation used in the development of the prototype. The prototype connects the PSUADE UQ software through a coordination gateway to the AspenPlus simulation.

Section 3 provides a detailed use-case for UQ analysis.

4.3. Sinter Gateway (Layer 2)

The Sinter Gateway presents a generalized Data Model for all simulations. Figure 8 below shows the relationship of the gateway to the clients such as UQ and simulations; it also shows the

three web resources -- session, simulation, and job -- which are manipulated by the Sinter Gateways. These form the basis of the current data model, detailed below.

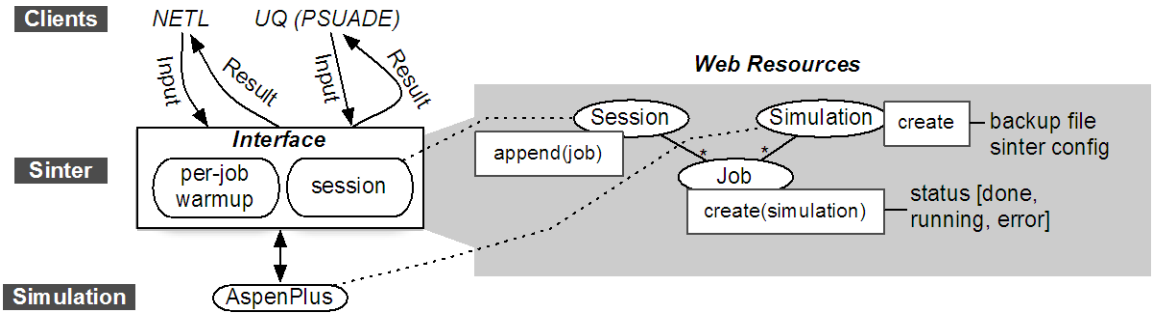

Figure 8. Sinter Gateway

4.3.1 Data Model

The Sinter Gateway currently stores its state and history in a relational database. Figure 9 below is a snapshot of the relational schema (note: the size of the VARCHAR and VARBINARY columns has been artificially set to 255 for the diagram). It tracks simulations throughout their lifecyle. A few of the tables and columns are specific to Aspen Plus, but most of them apply to all types of simulations. Comparing Figure 8 and Figure 9 one can see how the functionality of the Sinter Gateway is supported by the data store.

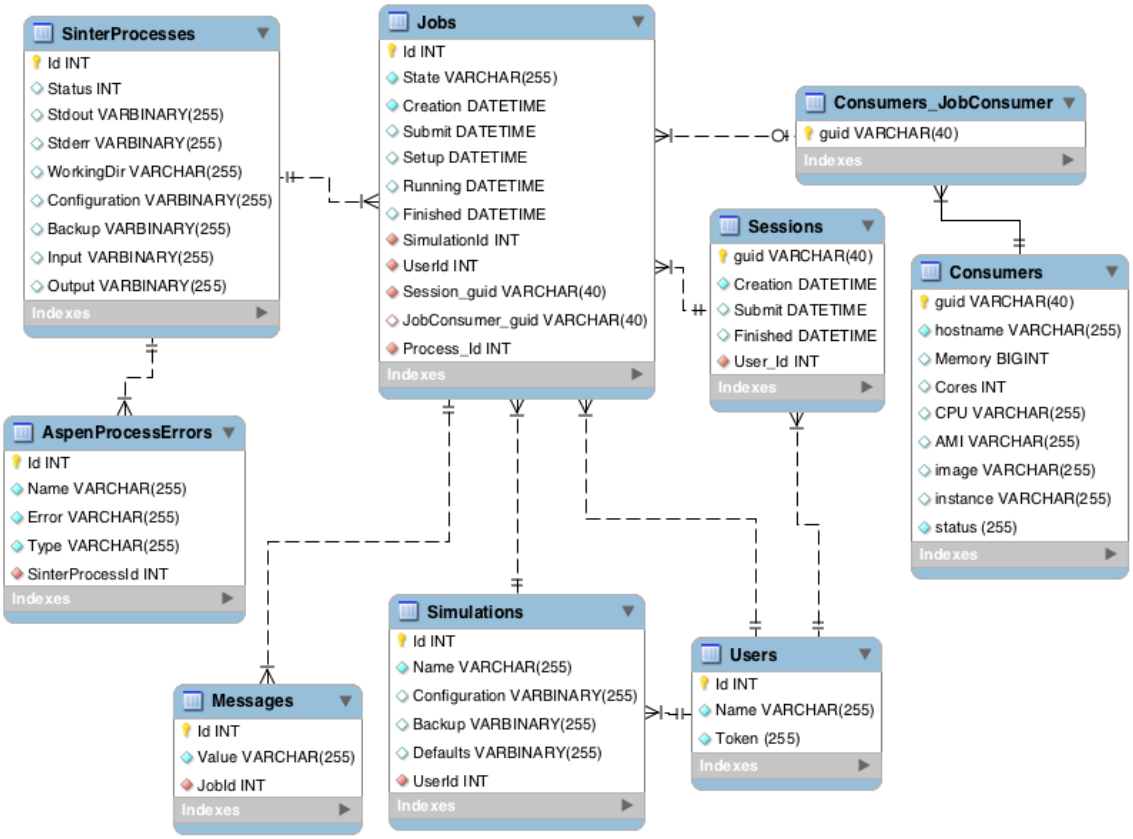

Figure 9. Sinter Gateway relational schema.

4.4. Sinter/AP (Layer 1)

The interface used by the Sinter Gateway to invoke functionality of the Aspen Plus simulation software was originally called *Sinter*. Somewhat confusingly, this term has also been used in CCSI Element 5 documents to refer to the general interface to a simulation. Therefore, we will use the term Sinter/AP (for Aspen Plus' Sinter).

4.4.1 Use-cases

Sinter/AP was designed to allow Aspen Plus simulations to be used as a component in a larger simulation scenario. This of course is a subset of the goals of the CCSI Element 5. An example using modeFRONTIER[2] optimization software to drive Aspen Plus is shown in Figure 10. In the figure, Sinter/AP is referred to simply as "sinter".

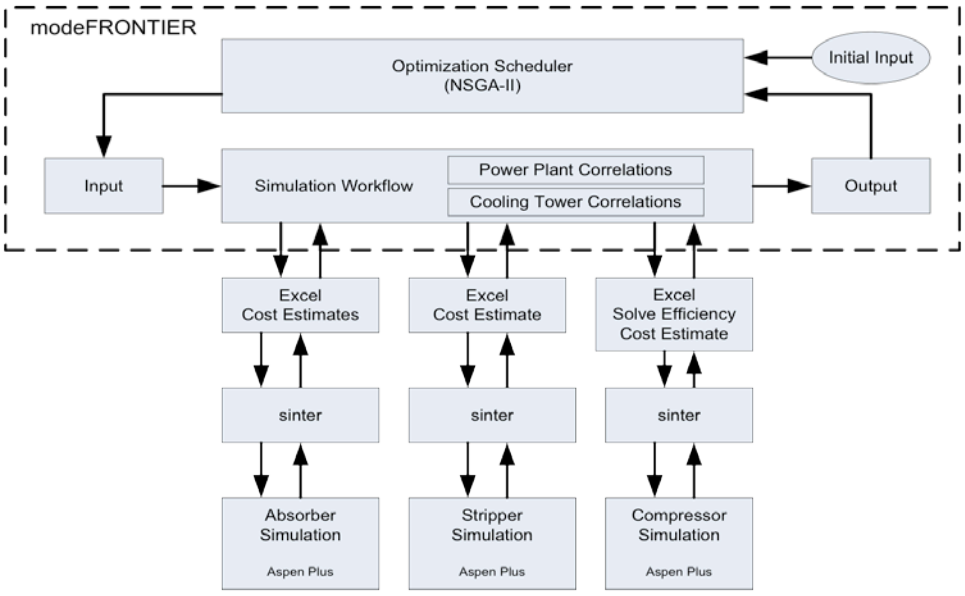

Figure 10. Example of driving AspenPlus with Sinter and modeFRONTIER

4.4.2 API

Using Sinter/AP, Aspen Plus functions can be invoked either through Windows COM, .Net, or C++ libraries. This was originally developed to allow, e.g., Excel and Matlab to drive an Aspen Plus simulation.

4.4.3 Data Model

Sinter/AP uses a configuration file called SinterConfig to describe a simulation. This file is stored in the Sinter Gateway schema as the `Configuration` column in the `Simulations` table

[2] Commercial software from ESTECO: http://www.esteco.com/

(see Figure 9). The file itself is a text file, which can be easily transmitted inside JSON documents. Given this, the current decision is that translating SinterConfig files to/from a JSON format is not sufficiently useful to merit the additional code, time, and maintenance overhead.

SinterConfig has one configuration directive per-line. The supported directives are:

- file: specifies simulation file. Example: `file | mea.bkp`
- dir: specifies the working directory. Example: `dir | c:\dir`
- title: specifies a title for a simulation. Example: `title | A Simulation Title`
- author: specifies the authors of a simulation. Example: `author | Jane Smith`
- date: the revision date of the simulation. Example: `date | January 1, 2000`
- description: a description of the simulation. Example:
  `description | A simulation of something`
- limits: specify the minimum and maximum for a scalar variable. Example:
  `limits | x | 0 | 10`
- min: specifies the minimum of a scalar, vector, or table. Example: `min | x | 0`
  - Matrix example:
    ```
    min | x | 0 | 0 | 0
    0 | 0 | 0
    0 | 0 | 0
    ```
- max: specifies the maximum value of a scalar, vector, or table; same form as min
- default: specifies the default value of a scalar, vector, or table; same form as min
- rStrings: specify a list of row strings used in constructing a table. Example:
  `rStrings | streams.out | STEAM01 | STREAM01 |STEAM03`
- cStrings: specify a list of column strings used in constructing a table. Example:
  `cStrings | streams.out | TEMP | PRES`
- rLabels: specify row descriptions. Example:
  `rLabels | streams.out | inlet stream | mid stream | out stream`
- cLabels: specify column descriptions. Example:
  `cLabels | streams.out | Temperature (F) | Pressure (psia)`
- rstrings_forEach: set the row names and strings to the names of streams or blocks of a specific type. Examples:
  ```
  rstrings_ForEach| stream.mixed.output | MATERIAL
  rstrings_ForEach| heater.table | Heater
  ```

4.5. EFRC Basic Data (Layer 0)

The Carbon Capture Materials DB (CCMDB; see https://mof1.cchem.berkeley.edu/ccmdb/) provides a comprehensive data collection on materials for carbon dioxide capture. CCMDB is a collaborative effort between Berkeley Labs Computational Research Division, UC Berkeley's Energy Frontier Research Center for Gas Separations and Electric Power Research Institute. In CCSI, the thermodynamic and other atomic properties in the CCMDB are part of "Basic Data".

Note that this component is placed at Layer 0, data storage and management, even though there is a simulation component to the EFRC. Although simulations are used to create this data, from the CCSI perspective these simulations are a black box. Access is only provided to the data after the simulations have been performed.

4.5.1 Use-cases

The primary use-case for the Basic Data model is as input and output data for queries from external, more coarse-grained, simulations in CCSI. For example, the CCMDB will be queried by Task 3 to retrieve important properties of carbon-containing materials. The most common queries are expected to be:

1. Retrieve data by *material name*, which is a well-known code for a given material
2. Retrieve data by *properties of the simulation*, such as the simulation name, temperature (or temperature range), simulation type, materials involved, and the gases present.

The CCMDB is expected to grow towards millions of computed properties, so these queries may return many results. A mechanism is required to limit and/or browse the results returned by any given query.

The current CCMDB is a relational database (MySQL). Below are some example queries that can be applied to this database.

- Search by *material name*. This name is standard within the chemistry domain. The search returns material ids, see (3) for further data that can be retrieved for each id.
- Search by *simulation properties* (material name, gas id, temperature, and simulation type). This search returns values related to properties of the matching simulations and associated gas molecules.
- Retrieve data by *material id*. This is the identifier within the database for this material. Table 1 lists several examples of the kinds of data that can be retrieved.

4.5.2 REST API

The REST API for EFRC Basic Data is still being completed. It will be built on top of the “native” CCMDB HTTP interface, which provides queries using an HTTP GET interface. The CCSI API will, by contrast, use a sequence of calls to create a “resource” for a given query, then retrieve rows of the result set by issuing HTTP GETs to this resource; this is a more RESTful style of API. The important high-level point to note is that even in cases where new remote interfaces are being created, it is likely that the CCSI interface may be layered over a simpler native interface to simplify conformance with more general interface approaches.

A partial example of the REST API is given in Appendix A: EFRC REST API.

4.5.3 Data Model

The EFRC basic data actually has two data models: one for the items present in a query, and another for the result. These of course will use the same terms where possible. The current query data model is given in Appendix B: EFRC Query Data Model.

4.6. Measurement Units

One area of general importance is the set of conventions for units of measure. Currently, all CCSI software supports some form of SI (international standard) units internally. For data exchange, we will model details of the representation. The base functionality of unit definition and interpretation will be performed by the open-source UDUNITS software, which was recently ported to Windows.

4.6.1 UDUNITS

We began with UDUNITS version 2.1.22, henceforth UDUnits2. The URL for the release is:

http://www.unidata.ucar.edu/blogs/news/entry/udunits_version_2_1_22

The UDUnits2 library was also copied to the CCSI source code repository at the following URLs:

- https://svn.acceleratecarboncapture.org/svn/ts5/external/UDUnits2/trunk
  - This directory will hold the evolution of the vanilla UDUnits2 distribution. When the distributors make another official release or release official patches, we will place it here.
- https://svn.acceleratecarboncapture.org/svn/ts5/external/UDUnits2/tags/2.1.24
  - This is the official 2.1.24 release tag. It is the unmodified form from the UDUnits2 distribution. The Unix/Linux sources will build against this revision of UDUnits2.
- https://svn.acceleratecarboncapture.org/svn/ts5/external/UDUnits2/branches/windows
  - This is a port of UDUnits2 to the Windows platform. It has a number of changes made by the CCSI TS5 team to work on Windows. As of 12/2011, the highlights are:
    - A new CMake based build which can generate Windows Studio project files to build the DLLs and headers required for CCSI. In particular, CCSI needs expat.dll (an XML parser), udunits2.dll (the units parser), convert.h, udunits2.h, udunits2_dll.h, and getopt.h.
    - Changes to header files to add __declspec(dllexport)/__declspec(dllimport) to properly export/import functions from/to DLLs.

The unit definitions for UDUnits2 are encoded in XML. The definitions of these units are at:

`http://www.unidata.ucar.edu/software/udunits/udunits-2/udunits2-{base, prefixes, common, accepted, derived}.xml`

4.6.2 Data Model

To integrate UDUNITS into the JSON documents exchanged in the CCSI integration framework, we have defined a standard way to include the UDUNITS definitions. In a nutshell, we add a standard “units” sub-section to any existing schema. This subsection has a list of keys and values, where the keys refer to attributes in the main document and the values are the units for that attribute. For example, given a JSON document like:

```
{ “power_plant” : {
           { “floor_plan” : {
           { “ width” : 1000,
              “depth” : 250 }
      }
}
```

To add units of meters (m) for the `width` and `depth` attributes, one would modify the document as:

```
{ “units” : {
      “power_plant.floor_plan.width” : “m”,
      “power_plant.floor_plan.depth” : “m”
  },
  “power_plant” : {
      { “floor_plan” : {
           { “ width” : 1000,
              “depth” : 250 }
      }
}
```

As shown in Appendix B: EFRC Query Data Model, the default units for items can be defined in the schema. It is an implementation decision whether units must be defined in every instance of a given data set, or the units can be inferred from the schema, or the units can be inferred from previous data.

5. Plan

The process for developing the data model is to work with the various groups involved in CCSI to identify the data needs and workflows and document the appropriate data characteristics of those workflows. This section describes the steps of this process in more detail then lays out the schedule for carrying out this process for CCSI components.

5.1. High-level Tasks

In the spirit of agile development, the high-level tasks will be performed in an iterative fashion. Each *Data Model Iteration* performs the following high-level tasks:

Data Model Iteration

1. **Enumerate required models**. Use documented workflow tool sets for each ICP milestone as basis of defining (in later iterations, refining) the list of data models to consider.
2. **Prioritize models**. Group the data models based on urgency and anticipated difficulty of implementation.
3. **Choose targets**, based on the prioritization in (2), for this iteration.
4. **Catalog data characteristics** for target data models in (3).
5. **Identify applicable engineering standards** for data elements in (4). Where conflicting or overlapping standards exist, work with CCSI collaborators on best choice.
6. **Implement data models** for targets in (3). This task involves a number of sub-areas that together form a complete view of the data model:
   - 6.1. **Description**. Create and/or modify human-readable and machine-readable description of the models.
   - 6.2. **Translation**. Create or modify software components to translate between these models and existing tools and formats.
   - 6.3. **Aggregation**. Work with application teams to identify requirements for aggregated datasets forming a consistent, verified and validated set for a simulation scenario.
   - 6.4. **Persistence**. Work with data persistence team on technical requirements for annotation capabilities, versioning, access controls (supporting IP requirements) and access methods.
7. **Identify interactions**. The new data models from (3) transformations needed to connect tools to data standards and inter-tool communications.
8. **Test and release**. Test the implementation, and release it.
9. **Iterate**. Return to Step 1.

5.2. Verification and Maintenance

It is important that the output of a given Data Model Iteration be verifiably correct for the major known use-cases. This verification must be repeatable, as it is expected that these use-cases will evolve over time, e.g. as new components begin to use the interface. To ensure these properties hold for all APIs and data models in the CCSI framework, we must build a testing process into the workflow. Therefore, we also associate a specific set of software engineering tasks with the implementation of each Data Model and associated REST API.

- **Document** model, API, and use-cases. This is essentially the output of Steps 6 and 7 in the Data Model Iteration. The output of this documentation will be placed under version control in a common repository.
- **Develop** a *mockup* of the functionality for major use-cases. The mockup will be able to imitate the behavior of the Data Model, in a limited way, without the presence of any actual computational back-end.

- **Test** mockups. Write clients that invoke the functionality in the mockup, and check results for correctness. These tests will also be added to the common repository.
- Once the tests pass, **tag and release** the data model. All version control systems provide the ability to associate a given snapshot of the files with a "tag". This can be used to provide a stable reference point for a given version of the data model and interfaces.
- When changes are made to the model, API, or use-cases, **repeat** this process to modify the mockups, run all tests again, and release a new version.

As noted above, making this testing and verification process efficient requires a software framework to automate the repetitive tasks. Ideally, this framework will be in the same language and even use the same libraries as the actually running implementations, allowing the interface for the mockup and interface for the functional version to share code. The reason this is desirable is that it makes it much harder for the implementation and the mockups to drift apart – changes in the first will tend to break the mockup tests, causing differences to be more quickly noticed and fixed. The same principle applies to the documentation and mockup/test implementations, although some amount of manual synchronization is likely to remain here.

This work will be coordinated with the software development support in CCSI Element 8 (E8), in particular the JIRA issue and bug tracking system. The details of the "tag" and "release" process will also be created in coordination with the E8 team.

In addition to internal developer verification through mockups and tests, we will work with the Element 8 team to implement periodic *code reviews*, a software engineering best-practice, that can provide some quality control on the data model and API implementations.

5.3. Schedule

The 5-year timeline of tasks for the CCSI simulation activities, taken from the FY12 Project Execution Plan, are shown in Figure 11 below. These targets have been guiding the initial work on data model development (see Section 3). We will continue this process of identifying which set of tools and datasets are used for each of the Industrial Challenge Problems (A0..A3) and then specify and document the appropriate data models over the course of the CCSI project. It is expected that this process occurs in the early stages of work for each of the ICPs with the schedule being driven the groups developing the various workflows and working with them in the early stages of those activities to see that the appropriate data models are documented.

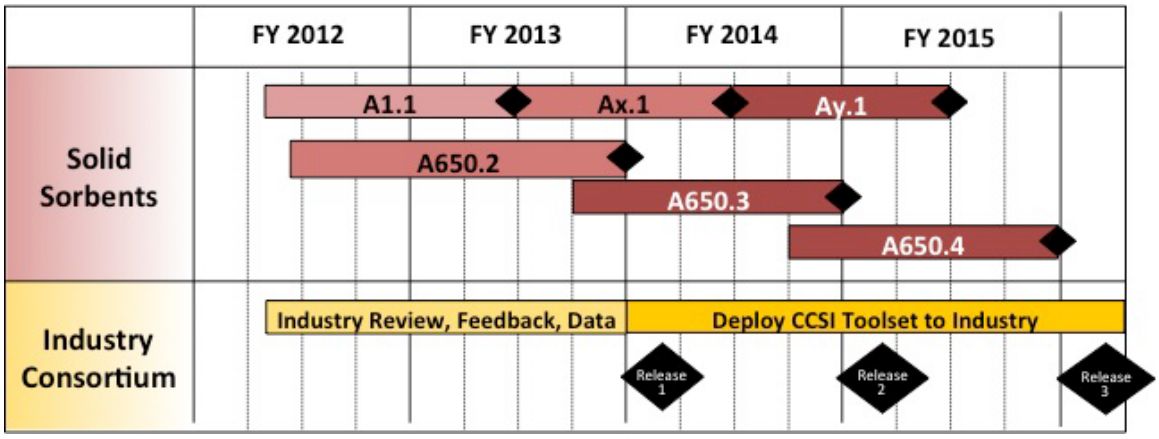

Figure 11. Timeline for CCSI Five-Year Plan

Appendix C gives the set of milestones for CCSI deliverables. These are the primary drivers for the schedule of data model development. The table below lists the dates for these milestones and identifies which data models are needed (earlier) in order to meet those milestones.

Date	**Appendix C Milestones**	**Data Models needed**
3/31/2012	3.c, 4, 6	V1 Sinter Gateway, UQ-Sinter Gateway, ROM Builder
9/30/2012	3.a, 3.b, 4, 5	CFD ROM, Analysis of integration for Superstructure-based models, UQ-Sinter using CCMDB, V1 Risk Analysis
3/31/2013	2.a, 2.b, 3.c, 6	Validation data and provenance, V2 Sinter Gateway incl. ROM, ROM with errors
9/30/2013	1.c, 3.a, 3.b, 4, 5	CFD ROMs, validation data and provenance, V3 Superstructure data, UQ-Sinter Gateway + ROMs, V2 Risk Analysis incl. ROM uncertainties
3/31/2014	2.a, 2.c, 3.c, 6	Validation data and provenance, CFD ROMs, V3 Sinter Gateway incl. UQ API, Dynamic ROMs
9/30/2014	3.b, 4, 5	V4 Superstructure data, UQ-Sinter Gateway with dynamic models, V4 Risk Analysis incl. dynamic models
3/31/2015	2.a, 2.d, 3.c	Validation data and provenance, CFD ROMs, V4 Sinter Gateway
9/30/2015	3.b, 5	V5 Superstructure data, V4 Risk Analysis

5.4. Data Model Review Process

The *verification* process outlined in Section 6.2 helps to assure that the data models (and associated API, if any) meet their specifications. The complementary activity to verification is *validation*, which assures that the specifications of the data models themselves satisfy the original CCSI requirements. In other words, verification checks that the model is right; validation checks that we have the right model.

The validation process will be performed by periodic Data Model Reviews. These will occur on a longer timescale than the release-time verification. The reviews will be focused on answering, for each data model, the following questions:

- Does the data model specification satisfy the known use-cases of the associated component(s)? If not, prioritize the unsatisfied use-cases
- Is the documentation for the data model clear and correct?

To avoid conflict with IAB reviews, they will be scheduled roughly twice per year in the intervening quarters. For convenience, the Data Model Reviews may be co-scheduled with code reviews (these were discussed as part of the verification process); however, these remain separate processes. The Data Model Reviews will produce a documented set of prioritized recommendations for changes. The first Data Model Review will happen in Q2 or Q3 of FY12.

6. Summary

This document provided a roadmap for creation, implementation, and deployment of the CCSI data models and associated infrastructure. Current progress was described in detail (with even more detail in the appendices), followed by a high-level description of the agile iterative process, and a timeline for completion of the currently understood data model requirements. Processes for verification and validation of the data models were also described. We intend for this roadmap to be a living document, amended periodically to capture new details about progress and amendments to process.

Appendix A: EFRC REST API

The following is an example of the Documentation of a REST API, in this case for the EFRC Basic Data. The only documented functionality, below, is creating the query; another REST API call would retrieve the data.

The Carbon Capture Materials DB provides a comprehensive data collection on materials for carbon dioxide capture. CCMDB is a collaborative effort between Berkeley Labs Computational Research Division, UC Berkeley's Energy Frontier Research Center for Gas Separations and Electric Power Research Institute.

Base URL = https://mof1.cchem.berkeley.edu/ccmdb/

1) POST /create

Create a new cursor resource from which to fetch results.

Parameters

- `query` Object conforming to EFRCBasicData_query JSON-schema.

Results

Status 200 – Document with an identifier for the cursor:

```
{ "cursor_id" : "string-identifier" }
```

Status 400/500 – Document with error information:

```
{ "error": {
    "type" :  "<token>",
     "msg" : "<English description>",
     "details" : { "<key>" : "<value>", … }
  } }
```

Examples

1. Successfully creating a query

   ```
   HTTP POST <base>/create
   { "query" : {
        "density" : {
        "min" : 2000
     } }
   RESULT (200)
   { "cursor_id" : "5BB099B0-2720-4D13-8ADF-07453E279006"  }
   ```

2. Failure creating a query, due to a malformed request

   ```
   HTTP POST <base>/create
   { "query" : {
   ```

```
    "foobar" : {
     "min" : 2000
   } }
```

```
RESULT (400)
 { "error": {
     "type" : "unknown_parameter",
      "msg" : "parameter 'foobar' is not recognized",
      "details" : { "parameter" : "foobar" }
   } }
```

3. Failure creating a query due to a server error such as DB being down

```
HTTP POST <base>/create
 { "query" : {
    "density" : {
    "min" : 2000
  } }
```

```
RESULT (500)
{ "error": {
   "type" : "db_error",
   "msg" : "cannot connect to database",
   "details" : { }
}
```

Appendix B: EFRC Query Data Model

The following is an example of a JSON-Schema data model, including default units of measurement. This schema describes terms that may be present in a query of the EFRC Basic Data interface. This is a simple filter, with each query term is specified as a “range” with a minimum and/or maximum value.

```
{
    "name" : "EFRCBasicData_query",
    "description" : "Request for data from EFRC database",
    "additionalProperties" : false,
    "properties" : {
        "units" : {
            "description" : "Names and default values for all
properties",
            "properties" : {
                "density" : {
                    "type" : "string",
                    "default" : "Kg/m^3"
                },
                "unit_cell_volume" : {
                    "type" : "string",
                    "default": "Angstroms"
                },
                "unit_cell_volume" : {
                    "type" : "string",
                    "default": "Angstroms"
                },
                "surface_area_He" : {
                    "type" : "string",
                    "default": "m^2/g"
                },
                "Di" : {
                    "type" : "string",
                    "default": "Angstroms"
                },
                "Df" : {
                    "type" : "string",
                    "default": "Angstroms"
                },
                "Dif" : {
                    "type" : "string",
                    "default": "Angstroms"
                },
                "kh_co2" : {
                    "type" : "string",
                    "default" : "mol/kg/Pa"
                },
                "hoa_co2" : {
                    "type" : "string",
                    "default" : "mol/kg/Pa"
                },
                "Pe" : {
                    "type" : "string",
```

```
                    "default" : "kJ/kg",
                    "description":"of CO2"
                },
                "Wc" : {
                    "type" : "string",
                    "default" : "wt %"
                },
                "purity" : {
                    "type" : "string",
                    "default" : "molar"
                }
            }
        },
        "range" : {
            "description" : "Base properties for a numeric range",
            "properties" : {
                "min" : {
                    "type" : "number" },
                "max" : {
                    "type" : "number" }
            },
            "additionalProperties" : false
        },
        "density" : {
            "description" : "Density of the material",
            "extends" : "range"
        },
        "unit_cell_volume" : {
            "description" : "Volume of the material",
            "extends" : "range"
        },
        "surface_area_He" : {
            "description" : "He based surface area",
            "extends" : "range"
        },
        "Di" : {
            "description" : "Maximum included sphere",
            "extends" : "range"
        },
        "Df" : {
            "description" : "Maximum free sphere",
            "extends" : "range"
        },
        "Dif" : {
            "description" : "Maximum included sphere along the path of
maximum free sphere",
            "extends" : "range"
        },
        "kh_co2" : {
            "description" : "Henry Coefficient (CO2)",
            "extends" : "range"
        },
        "hoa_co2" : {
            "description" : "Heat of Adsorption (CO2)",
            "extends" : "range"
        },
        "Pe" : {
```

```
            "description" : "Parasitic energy",
            "extends" : "range"
        },
        "Wc" : {
            "description" : "Working capacity",
            "extends" : "range"
        },
        "purity" : {
            "description" : "Purity at minimum energy",
            "extends" : "range"
        }
    }
}
```

Appendix C: CCSI Milestones

1. **Validated sorbent submodels for candidate sorbents that can be used within both high fidelity simulations and process simulations.**

    a. **A simple kinetic model for amine-based solid sorbents** which accurately captures the competing uptake/release of CO_2 and water (*completion date: 30 Sept. 2011*).
    b. **A high-fidelity multi-scale kinetic/diffusion model for amine-based solid sorbents** which also accounts for the microstructure of the sorbent particle (*completion date: 30 Sept. 2012*).
    c. **Reduced order model implementation of the high-fidelity multi-scale sorbent model** for computationally efficient incorporation into high-fidelity equipment models and process models (*completion date:* 30 Sept. 2013).

2. **Validated, high-fidelity models of solid sorbent carbon capture equipment at various scales, including 1 MW pilot scale, intermediate scales and full scale.**

    a. **High fidelity full scale model of solid sorbent adsorber and regenerator**. V1 model (*completion date: 31 March 2012*); V2 improved model which incorporates detailed sub-grid model and input from 1 MW high fidelity simulations; partially validated (*completion date: 31 March 2013*); V3 improved model which incorporates input from intermediate scale high fidelity simulations; partially validated (*completion date: 31 March 2014*); V4 Final validated model (*completion date: 31 March 2015*).
    b. **High fidelity 1 MWe scale model of solid sorbent adsorber and regenerator**. V1 model (*completion date: 30 Sept. 2012*); Validation complete (*completion date: 30 March 2013*).
    c. **High fidelity intermediate scale (~25 MWe) model of solid sorbent adsorber and regenerator**. V1 model (*completion date: 30 Sept. 2013*); Validation complete (*completion date: 31 March 2014*).

d. **High fidelity intermediate scale (~100 MWe) model of solid sorbent adsorber and regenerator**. V1 model (*completion date: 30 Sept. 2014*); Validation complete (*completion date: 31 March 2015*).

3. **Models, computational tools, user interfaces and accompanying methodology for optimal process synthesis and process design.**

   a. **1-D process models of solid sorbent adsorbers and regenerators.** V1 of initial model set (*completion date: 30 Sept. 2011*); V2 model incorporating initial CFD simulation results (*completion date: 30 Sept. 2012*); V3 model incorporating initial validated CFD simulation results (*completion date: 30 Sept. 2013*).
   b. **Superstructure-based optimization framework to determine optimal process configurations**. V1 Initial demonstration of algebraic surrogate generation tools and superstructure optimization (*completion date: 30 Sept. 2011*); V2 Superstructure includes heat integration within carbon capture system (*completion date: 30 Sept. 2012*); V3 Superstructure includes heat integration across full plant (boiler, steam cycle, capture process, and compression system) (*completion date: 30 Sept. 2013*); V4 Superstructure includes simultaneous process synthesis, heat and mass integration (*completion date: 30 Sept. 2014*); V5 Generalized superstructure to support other carbon capture system technologies and configurations (*completion date: 30 Sept 2015*).
   c. **Heterogeneous simulation-based process optimization framework**. V1 Framework accommodates process models (*completion date: 31 March 2012*); V2 Framework also accommodates reduced order models (*completion date: 31 March 2013*); V3 Framework incorporates uncertainty quantification/propagation and optimization under uncertainty (*completion date: 31 March 2014*); V4 Framework supports multi-scale optimization using reduced order models (*completion date: 31 March 2015*).

4. **Uncertainty Quantification (UQ) Framework to assess the effect of uncertainty in underlying parameters and models on the predicted performance of equipment and processes in a carbon capture system.** V1 Framework demonstrating initial integration and application (*completion date: 31 March 2012*); V2 Framework for application to solid sorbent carbon capture process simulations connected with basic physicochemical properties (*completion date: 30 Sept. 2012*); V3 Framework with added application to high-fidelity simulations (*completion date: 30 Sept. 2013*); V4 Framework with added application to dynamic simulations (*completion date: 30 Sept. 2014*).

5. **Risk Analysis & Decision Making Framework, which incorporates multiple risk contributors including technology readiness, uncertainty in models and simulations, economic uncertainty, and technical scale-up risk.** V1 Framework incorporating risk contributors from process models and their associated uncertainty (*completion date: 30 Sept. 2012*); V2 Framework incorporating additional risk contributors from high-fidelity equipment simulations and their associated uncertainty (*completion date: 30 Sept.*

2013); V3 Framework incorporating additional risk contributors from dynamic process simulations and control assessments and their uncertainty (*completion date: 30 Sept. 2014*); V4 Framework incorporating additional feedback from users (*completion date: 30 Sept. 2015*).

6. **Reduced order model** (**ROM**) **development tools to automatically develop ROMs from high-fidelity models in a form suitable for incorporation into larger scale models and simulations.** V1 Initial tools (*completion date: 31 March 2012*); V2 adds ability to estimate maximum error, i.e., percent deviation from high-fidelity model (*completion date: 31 March 2013*); V3 adds ability to create dynamic ROMs (*completion date: 31 March 2014*).