SOFT: A FRAMEWORK FOR SEMANTIC INTEROPERABILITY OF SCIENTIFIC SOFTWARE

Thomas F. HAGELIEN[1*], Andrey CHESNOKOV[2†], Stein Tore JOHANSEN[2 3‡], Ernst A. MEESE[3§], Bjørn Tore LØVFALL[3¶]

[1]SINTEF Ocean, 7465 Trondheim, NORWAY

[2]NTNU, Department of Energy and Process Engineering, 7491 Trondheim, NORWAY

[3]SINTEF Materials and Chemistry, 7465 Trondheim, NORWAY

* E-mail: thomas.hagelien@sintef.no
† E-mail: andrey.chesnokov@ntnu.no
‡ E-mail: stein.t.johansen@sintef.no
§ E-mail: ernst.a.meese@sintef.no
¶ E-mail: bjorntore.lovfall@sintef.no

ABSTRACT

In this paper we present our strategy and implementation of a datacentric modelling framework (SOFT, SINTEF Open Framework and Tools) with focus on information interchange in through-process and multiscale applications. SOFT needs to accommodate for a inhomogeneous set of in-house open source and proprietary simulators, often written in different programming languages, and storing data in different formats. The complexity and diversity of such a system requires that we have formal schemas and structures of metadata that allow for information interpretation regardless of the original storage formats, which application produced the data, and which application processes the data. We propose a standard for data exchange, separately describing metadata specific to different knowledge domains.

SOFT, via a mechanism of plugins, offers the possibility to utilize different tools for storage of such data and metadata. Further, SOFT facilitates scientific software development by clear separation of numerical routines and platform-dependent input, output, and analysis routines. Automated testing and simulation data analysis are also achieved in SOFT via external plugins and interfaces to scripted languages such as Python and Javascript.

The framework has been developed and tested within such flow modelling projects as LedaFlow, NanoSim and SimcoFlow.

Keywords: Metadata, information interchange, semantic interoperability, software framework, JSON .

INTRODUCTION

Modern scientific modelling software often has to operate with large amounts of data coming from different sources. A typical example is flow or process simulators where phenomena on different time and spatial scales has to be connected and data exchanged. If sub-models or relevant experimental information are available, it would be very efficient if we could "plug in" such models or data and have them available inside of a simulator with a minimum of work. In our development of the open source software SimcoFlow, the target is to develop a multi-material simulator which can handle complex multiphase flows with coexisting dynamic materials, free surfaces, and dispersed phases. In such a framework, it would be of great advantage if we could, with a minimum of work, use available sub-models for physics, available Cartesian cut-cell grid generation methods, visualization methods, as well as models/data for thermodynamic and material properties.

Most often such simulation software has to function as part of a pipeline, producing data for post processing, analysis and input for a higher level simulator. The data is often stored in a variety of formats, ranging from proprietary and closed formats, to well-documented open standards. In addition to this, the data often lacks information, such as which unit is used for a data point, how the data was produced, what are the uncertainties of the data etc. — as this is either implicit or simply just "understood" by the systems interpreting the data.

In this paper we describe a framework that is operating on a high level of abstraction, allowing for loose coupling between syntactic data representation (data structures and file formats) and internal representation in a software system. The framework operates within concepts of metadata and metamodels, representing different entities and their domain specific relationship, and how this can be used to connect different scientific modelling tools in a workflow. The framework also eases identification, traceability and reproducibility of simulations, and allows for easier separation of different knowledge domains, while also reducing development time.

HISTORICAL APPROACHES

The information technology has evolved into a world of largely loosely coupled systems and as such, needs increasingly more explicit, machine-interpretable semantics. There have been several approaches to formalizing interoperability, arising from different areas of knowledge.

A good example of an important problem is an effective data sharing across government agencies and other organizations. Such sharing relies upon agreed meanings and representations. A key technological challenge in electronic governance is to ensure that the meaning of data items is accurately recorded, and accessible in an economical, preferably, fully automatic fashion. In response, a variety of data and metadata standards have been put forward: from government departments (DHS, 2012), from industry groups (Chieu *et al.*, 2003), and from organizations such as the ISO (ISO, 2013) and W3C (Lassila and Swick, 1997). A short review of several challenges and initiatives in the area is presented in the works (Obrst, 2003) and (Davies *et al.*, 2008).

Standardization activity in software engineering was originally focused upon language and protocol design: upon the intended interpretation of programming statements, and upon

the concrete representation of data and commands. Since then, there has been a significant shift in focus towards metadata standards, such as descriptions of intended functionality and meaning that can be associated with particular items of data, in order to ensure consistent treatment and interpretation.

There could be mentioned several serious failures in industry due to misinterpreting of metadata. Probably the most well known example is the NASA Mars Climate Orbiter (NASA, 1999), that was lost after two different subsystems did not agree on communication. The first was sending values in US Customary units, the second assumed that the values arriving were measured in SI units. In this example the data itself, i.e. number, was passed correctly, but the metadata, i.e. unit, was lost during communication. This kind of mistake is hard to avoid when data is processed and integrated automatically, while its semantic consistency, such as the compatibility of units and intended interpretation, is checked only manually.

Importance of specification and recycling of standard metadata elements has led to development of the ISO 11179 standard (ISO, 2013), an international standard for metadata registration. The standard addresses “the semantics of data, the representation of data, and the registration of the descriptions of that data”.

An important initiative in the domain of material modelling is the SimPhoNy project (Hashibon, 2014). The main concept of the SimPhoNy framework is to augment existing open-source and commercial simulation tools and supplement them with sophisticated interface software libraries that allow for flow of information from one component to the other and from one scale to another. The integrated tools range from those describing the electronic structure and atomistic scales up to those modelling mesoscopic and macroscopic device level scales.

On the low level, the metadata model in SimPhoNy specifies a basic knowledge-based set of keywords that cover all aspects of the models and associated numerical computational methods. For example, temperature is a fundamental concept that is used in numerous models in different contexts. In continuum models it is the macroscopic thermodynamic temperature of a whole domain or on specific mesh elements. While for atomistic systems it may be the local kinetic temperature or a global parameter defining the interaction with a thermal bath. It can be associated therefore with either a parameter of the system or a variable. In either case, one keyword is associated with temperature in SimPhoNy with appropriate basic metadata. This metadata is then augmented with the associations and relations between different model components to obtain readily the exact nature of the particular temperature and its context in the model.

GENERAL IDEAS

In this section we define basic concepts of metadata, semantic modelling and data interchange.

Metadata

To achieve interoperability, the data that is communicated needs to be labeled, categorized and described. This description is generic, and is called metadata. Usually this is a means to define the data syntactically. It is also possible to think about metadata as extra information that can be attached to data. A good example of this is EXIF-info, which is attached to images and stores metadata such as date and time information, camera settings, etc. These two definitions of metadata are one and the same, with just a different set of attributes.

Ways of data interoperability

Interoperability between two or more scientific simulators is the process of taking some input into one simulation tool, producing the output and transforming the data into a suitable form that can be read and interpreted by the second simulator. Often the data is stored to disk, but can also be exchanged directly between two simultaneously running simulations via RPC, MPI etc. Here we focus on file based information exchange.

The data that is written to disk needs to be structured such that data can be retrieved. The format (syntax) of the data files can be either proprietary (closed) or open. To generalize information exchange from proprietary formats is difficult. Using open file formats is better, but it requires that wrappers are written an placed in the pipeline between the simulators in a defined workflow. The wrappers allow for the conversion of an application specific representation to a generic representation understood by a framework. If the formal specification of the generic representation contains enough information to be self-contained and allow for data sharing between different simulators and domains, we call this a schema for semantic interoperability.

A framework for semantic interoperability should allow for the exchange of information with any other "context compatible" system that shares the same attributes, purely based on a formal description of the semantics, without regards to the underlying syntax or file formats. This, however, requires that there exists wrappers that handles application specific file formats. The semantic framework builds on top of a set of syntactic layers, without exposing the details to the scientific simulator.

Defining semantic interoperability

Semantic interoperability can be achieved by formally describing all properties of the data that is to be exchanged, along with domain specific relationships between categories of data (entities). Necessary information includes attributes such as property names, types, units, rank (dimensionality), etc. This allows for reading and writing the data at the syntactic level. The schema for describing the semantics needs to be formal such that information can be shared between platforms and domains. Relationships between different entities need to be described explicitly as to define how they are connected. Any given relationship between two or more entities is bound to a certain domain, and might not be true for another, as such it is important to separate the relationship descriptions from the data descriptions. With this, it is possible to have interoperability across domains, where the semantically equivalent data points are used in two different contexts.

There are certain desired requirements to a formal schema-describing language that would be used as a building block for domain-specific schemas. It should be well-defined, minimalistic, be able to describe itself and provide versioning for handling of changes during the development process. There already exist several industry-wide languages that supersede these requirements, such as XML, YAML, JSON and others.

Concepts in semantic modelling

An entity is a fundamental concept in semantic interoperability approach. The entity is a generic concept that can represent anything, and according to Merriam-Webster is defined as "Something that exists by itself, something that is separate from other things". This means that entities have unique identities and sets of attributes. Such identities and attributes are described using formal schemas in a selected language. The schemas represent self-contained information about some object or concept.

While during well-known entity-relationship modelling (Chen, 1976) a model is composed of entity types, which classify the things of interest, and specifies relationships that can exist between instances of those entity types, we allow an entity to also include information about the state data, accompanying metadata, entity relations, workflow semantics and more.

Similarly to the Resource Description Framework data model (Lassila and Swick, 1997), we define facts as binary relations between entities. Then a system that is being modelled can be represented as a set of entities combined into collections by a fact-based semantic data model. Since a collection is an entity, collections can define relationships between collections as well as data-entities and model-entities.

While entity definitions are shared between knowledge domains, relationships between entities change between the domains. It is therefore important to separate these two concepts. Relationships in SOFT are expressed using triples. Each triple has a subject-predicate-object structure. To illustrate this, let us define a person as an entity, and give it properties such as name, date of birth, social security number etc. Instances of the entity person will represent an actual human being. To add relationships we can consider an example of a family business named FamCo, where the oldest son, Peter, is running the firm, while his sister Susan and father John are employees. In this domain the relationships can be defined like this:

"Peter"	"works at"	"FamCo"
"Susan"	"works at "	"FamCo"
"John"	"works at"	"FamCo"
"Peter"	"title"	"CEO"
"Peter"	"is-the-boss-of"	"Susan"
"Peter"	"is-the-boss-of"	"John"

In a different domain — say genealogy, we can consider the same entities, but the relations are completely different:

"John"	"is father of"	"Peter"
"John"	"is father of"	"Susan"

Instantiation of semantic models

To reduce dependencies to a specific piece of computational software, it is useful to classify the mathematical models that is realized in the software, and create a generic description that allows for the computational software to be interchangeable.

A mathematical model defines a transformation of a set of values (Input → Transformation → Output). Let these inputs, outputs and state data be described by a semantic data model. Then such a mathematical model can be considered as yet another abstraction level, namely, as a metamodel. By using this concept, we can describe a complete solution of a given business case, where the actual realization will not depend on any one given piece of computational software or file format. A suitable set of software simulators and data sources will be chosen then at runtime, which is called as an instantiation of a business case.

An important application of these ideas in the SOFT framework is an ability to build data agnostic simulators, that is, simulators working with different formats and storage technologies. With standardized formal metadata-schemas, we can allow reusable backend storage systems to handle I/O, thus letting computational models know only about the semantic data model and not the implementation details.

The current representation of the semantics in SOFT is targeted towards scientists and programmers with domain knowledge. The choice of representation syntax is intentionally kept minimalistic and pragmatic, without loosing the mapping to other representations, such as OSF.

EMMC initiative

The work presented here lies in heart of the EU cluster EMMC CSA (European Materials Modelling Council, `https://emmc.info/`), where framework development, semantic interoperability and metadata are key enablers for the coupling of new and existing materials modelling tools.

SOFT FRAMEWORK: PRACTICAL IMPLEMENTATION OF GENERAL CONCEPTS

Scientific software development — a datacentric approach

It is a common practice to start out the development of scientific software with implementing core functionality around numerical methods or other algorithms. We propose an alternative, namely, to start with data modelling of the system in the context of a framework such as SOFT (Domain Driven Design). We have experienced that this way the development will greatly benefit from the architectural quality attributes maintainability, interchangeability/interoperability and modifiability.

The modelling process starts with building a taxonomy of the domain for which the software system should be built. This will require expertise from both domain experts and data modelling experts. The taxonomy will be a blueprint of what will later be defined as entities and relationships. The taxonomy should include all relevant actors and properties the scientific software will address, without regards for what will be the workflow, transformations, inputs and output of the system. The next step in the process is formalizing the model into a recognized/standardized format suitable for metadata interchange.

The associations and dependencies defined in the taxonomy are candidates for the decoupled structuring of the information in data collections. The first step here is to formally define the entities. Then code generators can be employed to generate classes, templates, wrappers, serialization code, documentation, etc. In addition, the developed schemata can be published such that others working in a similar knowledge domain can benefit from the classification work, without having to redo all the work.

SOFT includes an autogenerator which can take any JSON data as input, and transform it into a custom text output, by

defining a template that mixes pure text and Javascript code identified by a special markup. One of the greatest benefits of this is that an entity can be changed anytime by changing only its corresponding JSON description. The changes are then applied to the rest of project automatically, without having to worry about updating all dependent code and documentation. Code autogeneration happens at compile time.

While developing formal schemata for the entities it seems to be beneficial to employ namespaces in the descriptions of the entities for avoiding name-clashes between domains, as well as version numbers for supporting a semi-automatic versioning system. This is the way the entities are named in the SOFT system.

SOFT supports I/O through a storage plug-in mechanism, which hides the I/O machinery from the modelling code. In cases where data needs to be communicated between multiple simultaneously running processes, the entities can be used to generate code used to serialize the data between the processes. Examples of this can be MPI (`http://pages.tacc.utexas.edu/~eijkhout/pcse/html/mpi-data.html`) and protobuf (`https://developers.google.com/protocol-buffers/`) messages. In these cases, the interfacing of the simulation software components is written or autogenerated based on entities, as part of the modelling code.

Exchanging data only requires that data is described in the formal semantic description and there exists support for the given syntax (data format) in the SOFT framework as a plugin. The APIs for entities and collections are used in the modelling code (and are usually embedded in the autogenerated code) to get access to the data. Except from this, no other APIs are necessary.

Example 1: Describing an entity

In SOFT, we have selected to use JSON as this is widely supported, human readable/writable and integrates nicely with the scripting engine of SOFT which is based in JavaScript. The proposed JSON schema for defining entities is currently being reviewed in the EMMC-CSA project and is subject for future standardization. The standard will be open and publicly available. An example of the JSON definition of an entity is shown in Figure 1. The entity is uniquely defined with a name, namespace and a version. Attributes are listed under the *properties* keyword. In this case there is only one property *foo*.

```
{
 "name": "example",
 "namespace": "com.example",
 "version": "0.1-SNAPSHOT-1",
 "properties": [
 {
 "name": "foo",
 "type": "string",
 "description": "A metasynt. variable"
 } ]
}
```

Figure 1: An example of a formal schema for defining an entity

Example 2: Taxonomy of a simplified simulation case

An example of an instantiation of a semantic model is a simulation case. Let us consider some imaginary case that has an inlet condition and an outlet condition. The case also has a set of predefined user inputs, such as physical constants, and computational results, such as velocity fields. The case also has a numerical solver that requires a matrix structure and some settings, such as tolerances, that can be changed by the user. In Figure 2 we have defined the relations between six different *Entities* (grey ovals) and two *Collections* (blue ovals).

All *Entities* are described by their formal JSON-metadata. These JSON-files are registered in the SOFT system and their counterpart in the desired programming language is autogenerated. Before running the simulation a Case Creator instantiates these *Entities* with input data and other parameters and stores them in a Domain *Collection* with a certain ID.

When the simulation is run, one gives this unique ID of the instance of the Domain *Collection* and the URI of a place where the raw data is stored, to SOFT. This is all that is needed; SOFT takes care of the rest. Thus, the simulation core does not have to take into account how the data is stored, provided that it complies to a given semantic, i.e. JSON, description. A *Collection* can be dynamic, such that, for example, the solver tolerance or inlet type can be changed at a runtime.

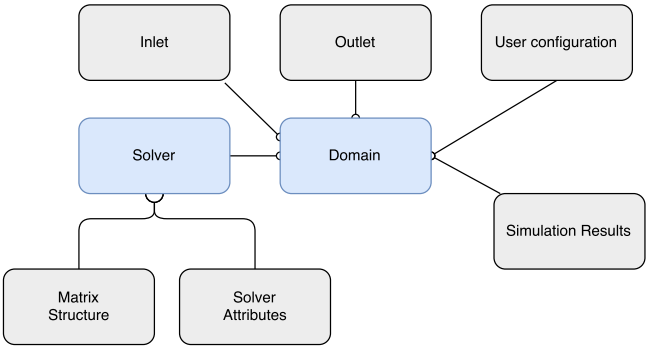

Figure 2: Taxonomy of a simulation case

Example 3: Entities and couplings in the NanoSim project

The NanoSim project (Singhal *et al.*, 2016) employs SOFT in a multiscale coupling workflow. In this example we demonstrate data transformations and exchange of information from an atomic scale simulation to a resolved CFD simulation. Figure 3 Illustrates a workflow, which starts with available intermolecular scale data (DFT), that was generated using VASP software (`https://www.vasp.at`). Raw data files are represented by using SOFT entities named *Reference*. These are instantiated and registered in a *Collection*. The next step of the workflow is to feed this to the REMARC software to generate information for chemical kinetics models. The REMARC (owned by SINTEF) package already reads the VASP data formats, so there is no need to transform the data further. In addition, the DFT files can be very large, so it makes no sense to store duplicates or transfer these files.

Additional input files to REMARC are handled via the entity named *File*, which stores information about an arbitrary block of data, a name, size and a cryptographic hash

string for consistency checking. REMARC is embedded in SOFT by using a wrapper that converts information from the *Collection* into the native file format and back again. In this case the CHEMKIN-II data files are read and written using a SOFT storage plugin. This enables the REMARC wrapper to work purely on semantic information along with a URI to the location of the actual data. One of the output entities from the REMARC wrapper is called *chemkinReaction*, whose corresponding JSON-description could be accessed online at `https://github.com/NanoSim/Porto/blob/master/porto/src/entities/chemkinreaction.json`. A part of it is shown on Figure 4. The data generated from REMARC is stored in a MongoDB database.

The URI to the MongoDB location, along with the unique case ID is then passed to a SOFT code generator, which generates a plugin for ANSYS Fluent. The code generator uses a general template for generating FLUENT UDFs as well as data that came from VASP through REMARK. The entire workflow is put together as a single SOFT script. The simulation is then run simply by passing the input files (VASP DFT data and REMARC configuration files) to the script, which, in its turn, generates the Collection, and attaches all entities automatically.

On Figure 3 we show the complete workflow. The DFT-prepare step creates a *Collection*, a *File* and a *Reference* entity, and instantiates these with data given from input. The output from the preparation step is the ID of the Collection, which is passed to the REMARC wrapper. The wrapper makes call to the REMARC simulator and appends REMARC data to the *Collection* using relevant entities, such as *chemkinReaction*. The output from the wrapper has the same ID as was given as input. The code generator receives the ID and builds a model based on the relevant entities found in the Collection. It then uses the SOFT code generator utility to produce a User Defined Function (UDF) that can be compiled and read by ANSYS Fluent.

Example 4: Workflow in the SimcoFlow project

A typical run of a SimcoFlow simulation could be divided into three parts: Case Creator, Simulator and Postprocessor. Simulator operates on initial data provided by Case Creator and, in its turn, generates the data for the Postprocessor. All data exchange happens via the SOFT framework, that takes care of several low-level details. Figure 5 shows data flow chart of the project. The datacentric architecture is noticeable by the data storage that is initially declared, and later used during the simulation run.

The Case Creator generates a Collection with all information describing a full simulation case. This will be typically written in a high level (scripting) language, or produced with a graphical user interface. The SOFT system provides the autogenerated code in this selected language for the Entities described with JSON-schemas. The process of working with metadata is shown on Figure 6. The language can be potentially different from what is used for the core numerical method. The *Entities* are then instantiated with the user provided input data and united in a *Collection*. This instance of a *Collection* is stored in a storage solution provided by SOFT, resulting in a unique UUID referring to the simulation case.

We assume that the main Simulator module has the following structure:

1. Fetch data from storage
2. Initialize simulation
3. Time stepper
   (a) Initialize time step
   (b) Do time step
   (c) Finalize time step
4. Finalize the simulation

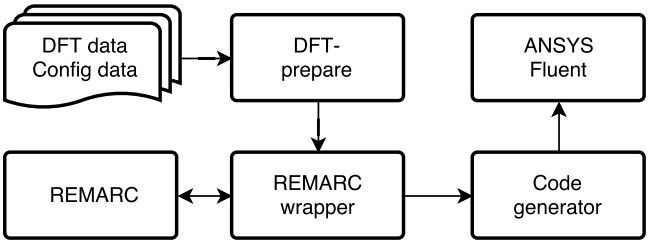

Figure 3: A workflow going from an atomic scale simulation to a CFD simulation using SOFT

```
{"name": "chemkinReaction",
 "version": "0.1",
 "namespace": "eu.nanosim.vasp",
 "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)).",
 "dimensions": [
  {
   "name": "nreactants",
   "description": "Number of reactants"
  },
  {
   "name": "nplog",
   "description": "Number of intervals
      the pressure dependency of the
      rate coefficients is described."
  } ],
 "properties": [
  {
   "name": "reactants",
   "type": "string",
   "dims": ["nreactants"],
   "description": "Name of each reactant
       species."
  },
  {
   "name": "P_plog",
   "type": "double",
   "dims": ["nplog+1"],
   "unit": "Pa",
   "description": "Pressures defining
      the borders of the nplog pressure
       intervals for defining pressure
      dependency of the rate constant."
  } ]
}
```

Figure 4: An extract of *chemkinReaction* entity.

This workflow is represented on Figure 5.

Each of the steps corresponds to a specific function, that can be written in any supported programming language. The only requirement is that its function pointer has to be registered in the system by Case Creator.

During step 1 the Simulator reads in input data using SOFT drivers, given the UUID from Case Creator, and initializes the native programming language structures, using metadata description autogenerated from JSON. During step 2 some physical parameters could be set. During step 3 the main computation is done. In principle, different programming languages can be used for different functions in step 3, if their argument structures follow the documented C API.

Each of the functions on steps 1-4 can in principle dump some data to the storage, using SOFT drivers and metadata description. However, we feel that it is natural to restrict dumping to steps 3c (store some information available at a working time step) and 4 (store the final result or the whole simulation with all intermediate steps). A generated UUID will refer to the stored simulation results.

Given this UUID and metadata descriptions, different applications can access the data storage independent from the simulation module, either in the process of the simulation or when it is finished. A natural use case here is postprocessing of data for visualization.

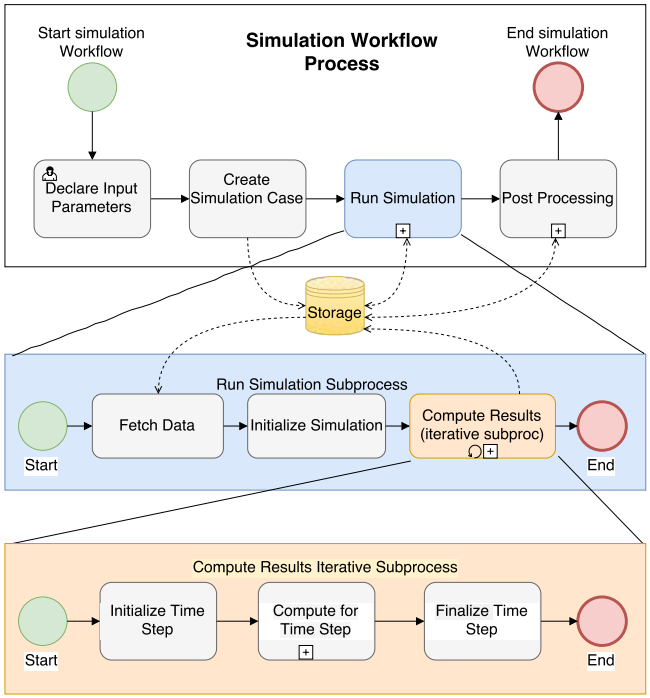

Figure 5: BPMN diagram illustrating the process of running a simulation with SimCoFlow

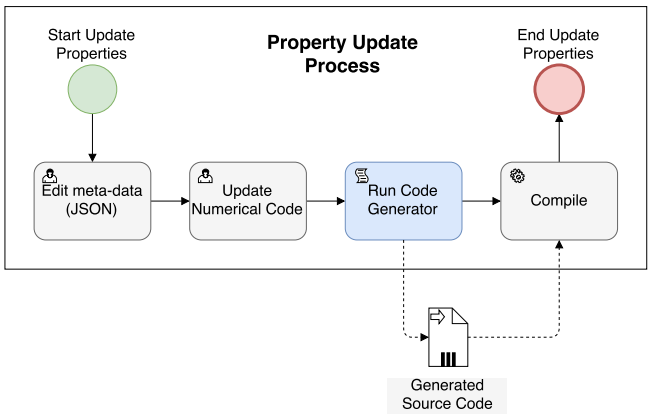

Figure 6: BPMN diagram illustating the process of updating properties/state data

CONCLUSION

In this paper we presented a datacentric approach for designing and developing scientific software that enables interoperability, and illustrate this approach with examples based on the SOFT framework and several projects that are employing it. By starting with the classification of the domain and transferring this into formal definitions of entities and associations/relations, a researcher creates a solution model that is both maintainable and flexible.

Employing a framework like SOFT makes it uncomplicated to share data between different areas of knowledge, by bridging the different domains with a semantic layer. This is done by the exchange of formal definitions of entities. The framework itself takes care of low-level data input and outputs in reusable modules, letting the physicians focus on the physics and software scientists focus on the data modelling within the same project.

ACKNOWLEDGEMENTS

The Norwegian Research Council, through the research program SimcoFlow (project 234126), is acknowledged for supporting this work.

REFERENCES

CHEN, P.P.S. (1976). "The entity-relationship model—toward a unified view of data". *ACM Trans. Database Syst.*, **1(1)**, 9–36.

CHIEU, T.C., FU, S.S., PINEL, F. and YIH, J.S. (2003). "Unified solution for procurement integration and b2b stores". *Proceedings of the 5th International Conference on Electronic Commerce*, ICEC '03, 61–67. ACM, Pittsburgh, Pennsylvania, USA.

DAVIES, J., HARRIS, S., CRICHTON, C., SHUKLA, A. and GIBBONS, J. (2008). "Metadata standards for semantic interoperability in electronic government". *Proceedings of the 2nd International Conference on Theory and Practice of Electronic Governance*, 67–75. ACM.

DHS (2012). *National Information Exchange Model, 3.0.* Dept. of Justice and Dept. of Homeland Security.

HASHIBON, A. (2014). "SimPhoNy: Simulation framework for multiscale phenomena in micro and nano systems".

ISO (2013). *ISO 11179. Information Technology Specification and Standardization of Data Elements.* International Organization for Standardization (ISO).

LASSILA, O. and SWICK, R. (1997). "Resource Description Framework (RDF) Model and Syntax". Tech. rep., W3C.

NASA (1999). "Mars Climate Orbiter Mishap Investigation Board Phase I Report". Tech. rep., NASA.

OBRST, L. (2003). "Ontologies for semantically interoperable systems". *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, 366–369. ACM, New Orleans, LA, USA.

SINGHAL, A., CLOETE, S., RADL, S., FERREIRA, R. and AMINI, S. (2016). "Cfd-dem predictions of heat transfer in packed beds using commercial and open source codes". *MAYFEB Journal of Chemical Engineering*, **1(1)**, 1–17.