Delivery date:

04-08-2016

Authors:

Sigve Karolius

MoDeNa

Deliverable D4.4

Documentation Database

Principal investigators:

Sigve Karolius
Henrik Rusche
Dominik Christ
Heinz Preisig

Collaborators:

Cansu Birgen

Project coordinator:

Heinz A Preisig

heinz.preisig@chemeng.ntnu.no

Abstract:

The report describes the role of the database in the MoDeNa software framework. The report summarises the model and data containers for the representation of mathematical models and emphasises on the way in which the framework is linked to the database.

© 2016
MoDeNa

Contents

1 Introduction 1

2 Overview 2

2.1 The database system 2

2.2 Role of the database in the MoDeNa software framework 3

2.3 Connecting to the database 4

3 MoDeNa Data Schema 5

3.1 Surrogate Function 6

3.2 Surrogate Model 7

3.3 Index set 8

4 Data Management 9

5 Discussion 10

Introduction

This report documents the database architecture that is used in the production release of the MoDeNa software framework. The structure has gone through several revisions, but the major improvement is embedding the low level API with a high level Python library, harnessing the speed of C with the abstraction that is possible in python. The production release focuses on hiding the database from the user of the API, and embedding the parameter estimation framework into an intuitive computer representation.

The MoDeNa software framework employs the MongoDB database as the centralised storage system. The database is a necessary part of the software framework for several reasons: common communication interface, storage of simulation results for later use and remote access of data. The database enables the surrogate models to be saved in a self-consistent manner, such that they can be instantiated directly from the database without the need to run new detailed simulation.

The MoDeNa software framework aims at hiding the database from the users, meaning that the software API handles database communication without the user having to take additional action. The goal is that the developers of the single-scale models in the multiscale application only has to deal with the MoDeNa API. Consequently, the recipes for embedding surrogate models into a model becomes consistent across the entire multiscale simulation.

The purpose of this report is to document the production release of the MoDeNa database. However, since the structure of the database has not changed since deliverable Karolius et al. (8 31) the main goal of this report will be to provide a birds-eye view of the role of the database in the MoDeNa software framework.

Even though the database is a central part of the computational workflow it is not being queried often. This is an important part of the software framework and it will be emphasised through the report when and where the database is actually queried. The reason for this is that it is important for users that have thousands of calls to different surrogate models that the database is not involved when the model is evaluated.

The layout of the report is the following:

Overview
: The illustrate the position of the database in the multiscale model, as well as in the computational workflow.

Data containers
: The philosophy of the database architecture is to represent the surrogate model in a way that mirrors notation from mathematics. In practice, this implies merging practical necessities, from a programming perspective, with the compact notation and terminology from mathematics.

Interfaces
: The low-level (C API) and high-level interface (Python) are highly intervened. Surrogate models are defined in Python, and the C API is implemented in part using the Python-C API.

Ultimately, the goal of the report is to document how ideas and philosophy have been abstracted and implemented.

Overview

This section aims at providing an overview about the database system that is used in MoDeNa, and the role of the database in a simulation that employs the MoDeNa software framework. The purpose is to provide a birds-eye view how the database in order to introduce the philosophy of the data structures and data management that is discussed in the later sections.

What will be emphasised throughout the report is that the database is used for data storage, and that its role is to save simulation results, as well as the state of the surrogate models. In other words, the MoDeNa software framework does not use the database to pass data between models.

The database system

The database system used in the MoDeNa framework is the MongoDB database. Generally, a database can be defined as an organized collection of data which enables us to handle large quantities of information by inputting, storing, retrieving and managing them.

Document database A record in MongoDB is a document, which is a data structure composed of pairs (field and value), or (key and value). The values of fields can also include arrays of other documents. MongoDB documents are in BSON ('Binary JSON') data format. Essentially, it is a binary form for representing objects or documents. Using documents is advantageous since in many programming languages they correspond to native data types, embedded documents (subdocuments) and arrays reduce need for expensive joins, and dynamic schema supports fluent polymorphism. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents. MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. Collections are analogous to a table in relational databases.

High performance MongoDB provides high performance data persistence. In particular, support for embedded data models reduces I/O activity on database system, and indexes support faster queries and can include keys from embedded documents and arrays.

High availability To provide high availability, MongoDB's replication facility, called replica sets, provide automatic failover and data redundancy. A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

Automatic scaling MongoDB provides horizontal scalability as part of its core functionality. Automatic sharding distributes data across a cluster of machines. Replica sets can provide eventually-consistent reads for low-latency high throughput deployment.

MongoDB queries exhibit the following behaviour:

- All queries in MongoDB address a single collection.
- Queries can be modified to impose limits, skips, and sort orders.
- The order of documents returned by a query is not defined unless a `sort()` method is used.
- Operations that modify existing documents use the same query syntax as queries to select documents to update.
- In aggregation pipeline, the $ match pipeline stage provides access to MongoDB queries.

Table 1: Terminology and concepts in SQL and MongoDB.

SQL	MongoDB
Database	Database
Table	Collection
Row	Document or BSON Document
Column	Field
Index	Index
Table joins	Embedded documents and linking
Primary key (specify any unique column or column combinations as primary key)	Primary key (the primary key is automatically set to the id field in MongoDB)
Aggregation (e.g. by group)	Aggregation pipeline

Role of the database in the MoDeNa software framework

The fundamental purpose of the MoDeNa software framework is to facilitate multiscale modeling through scale coupling employing surrogate models. A birds-eye view of the coupling between the models that make up the multiscale application is shown in Figure 1. The purpose of the database is

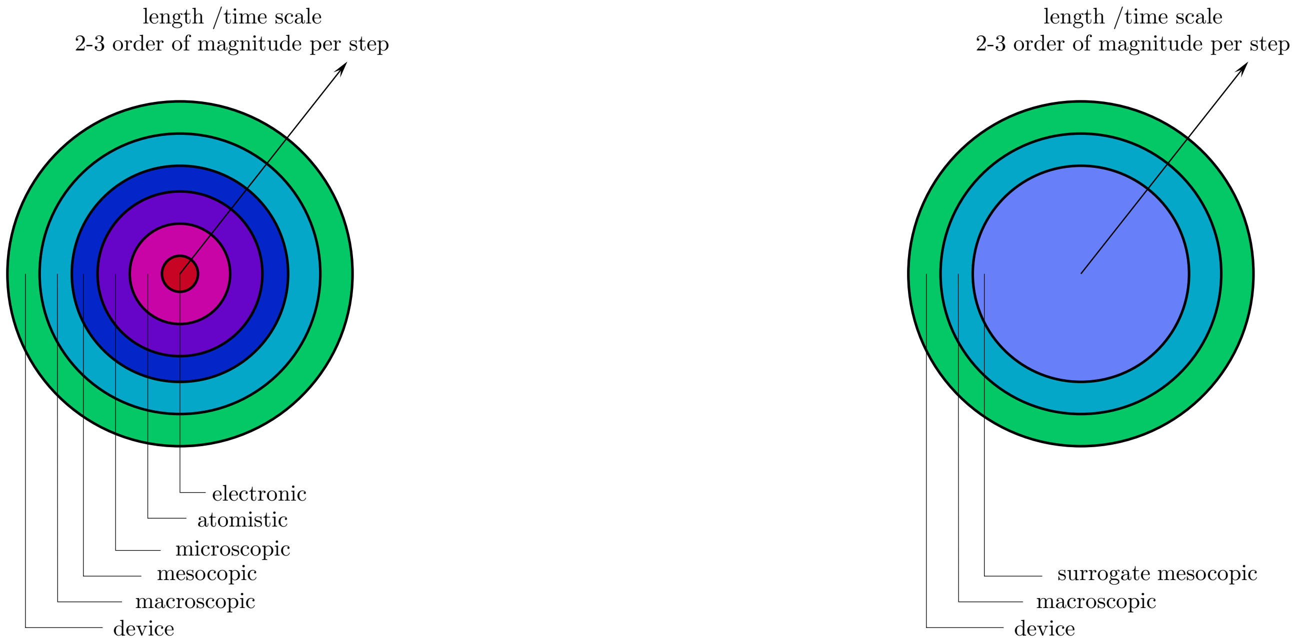

Figure 1: Figure illustrating scale coupling using surrogate models.

that of a centralised data storage, and the role in the computational workflow is shown in Figure 2. Even though the database is central to the computational workflow, the MoDeNa software framework aims at making it invisible to the user.

The philosophy of the communication is therefore that the software infrastructure handles communication and data management. Moreover, the data containers have been designed to be self-contained. I.e. the database contains all necessary information to instantiate a surrogate model.

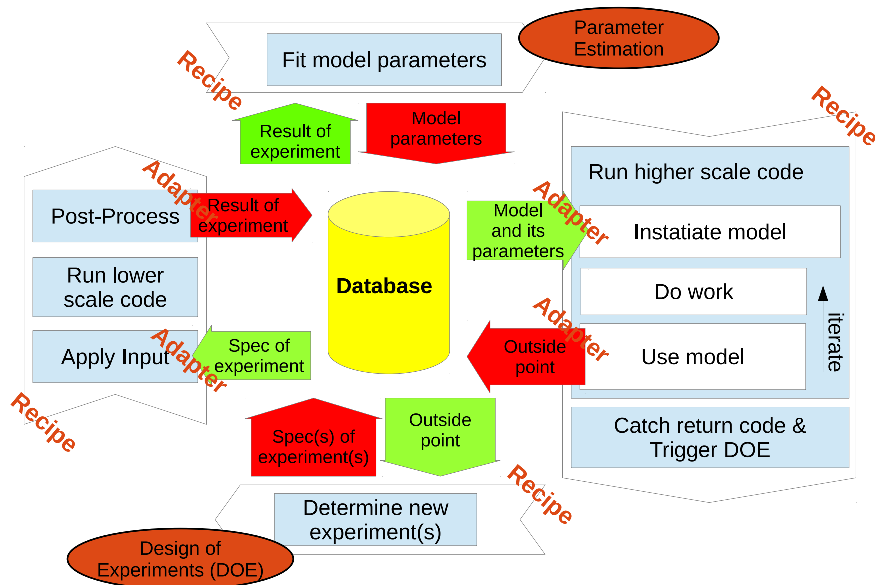

Figure 2: Illustration of the data flow of one scale connection using a MoDeNa surrogate model.

Connecting to the database

The MoDeNa framework connects to the MongoDB database using a uniform resource identifier (URI). By default the MoDeNa framework will attempt to connect to `localhost` and use the `test` database. The corresponding URI is: `mongodb://localhost:27017/test`.

However, if a user wants to use a different database, i.e. a different name than **test**, the MoDeNa framework will read the environment variable `MODENA_URI`. The environment variable can either be set from the shell:

```
export MODENA_DIR=mongodb://localhost:27017/<DATABASE-NAME>
```

Similarly, if the database is protected by a username (`USER`) and password (`PWRD`):

```
export MODENA_DIR=mongodb://<USER>:<PWRD>@localhost:27017/<DATABASE-NAME>
```

Also, if the database is hosted remotely, e.g. `www.mlab.com`, the URI becomes:

```
export MODENA_DIR=mongodb://<USER>:<PWRD>@ds011840.mlab.com:11840/<DATABASE-NAME>
```

However, the URI can also be set within the python script that starts the computational workflow, but in that case it must be set **before** the modena module is imported as follows:

```
import os
os.environ["MODENA_URI"]="mongodb://localhost:27017/<DATABASE-NAME>"
import modena
...
```

The MoDeNa framework is configured to use the same database for surrogate models and workflow management. Therefore, the URI also determines the database in which FireWorks stores the computational workflow.

MoDeNa Data Schema

The database containers that are relevant to modena are described in this section. Since the database is used by the computational workflow management software FireWorks, as described in Jain et al. (2015), the overall database also contains non-modena entries. An example illustration of databases, where the `test` database is specific to MoDeNa, in MongoDB is shown in Figure 3.

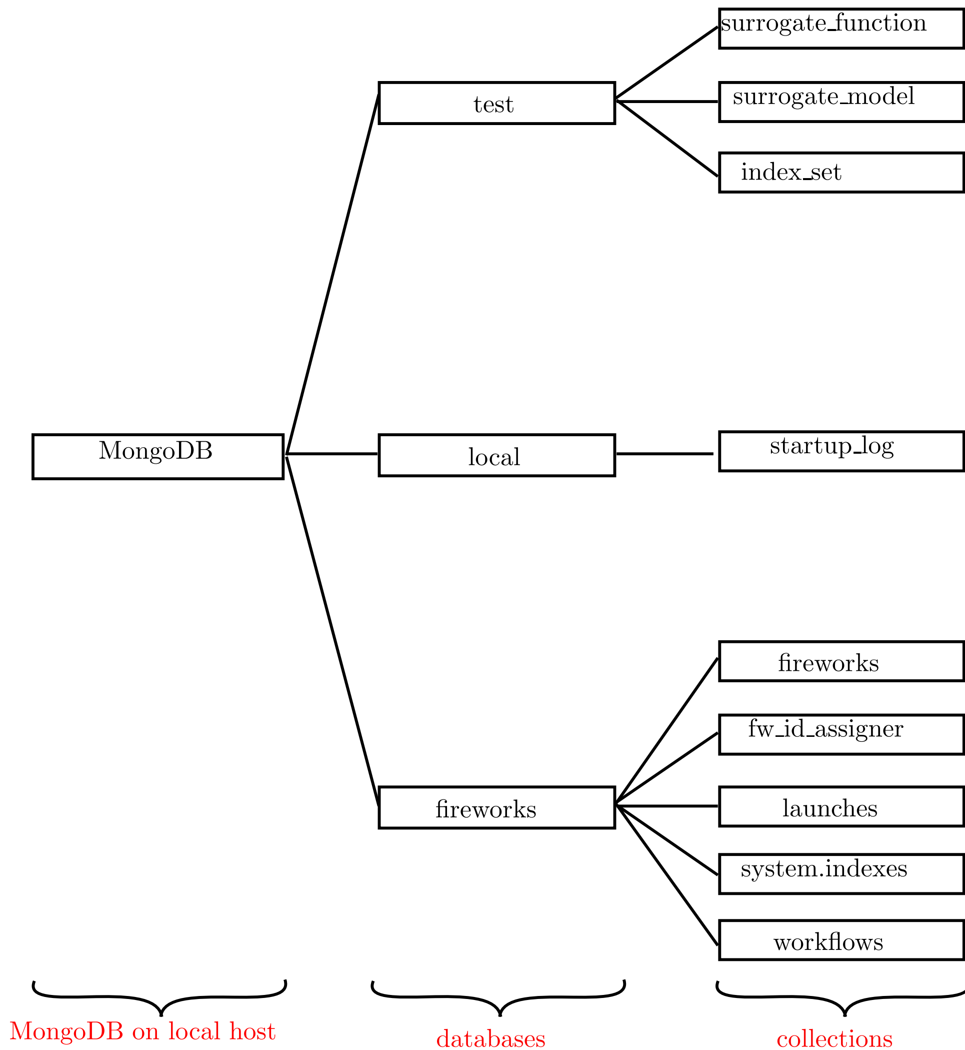

Figure 3: Illustration of database organization in MongoDB on a local host.

This section summarises the purpose and structure of the modena-specific collections: surrogate model, surrogate function and index set. Moreover, it will emphasise the reasons why the current, decentralised, data structure was chosen instead of a more unified approach. The philosophy of the data representation was to introduce enough abstraction in order to to minimise the amount of data, whilst ensuring that the surrogate models are modular and self-contained.

Moreover, the role of the database in a simulation is to store results from detailed simulations, as well as the state of the surrogate models. Therefore, the database can be described as a passive participant in the simulation, since it is not involved in passing data between the models. This detail is particularly important in applications that may need to evaluate a surrogate model thousands of times, e.g. CFD.

Surrogate Function

The surrogate function contains the functional representation of the surrogate model. As illustratef in Figure 4 it defines the inputs, outputs and parameters, as well as the C-code that is compiled into a executable.

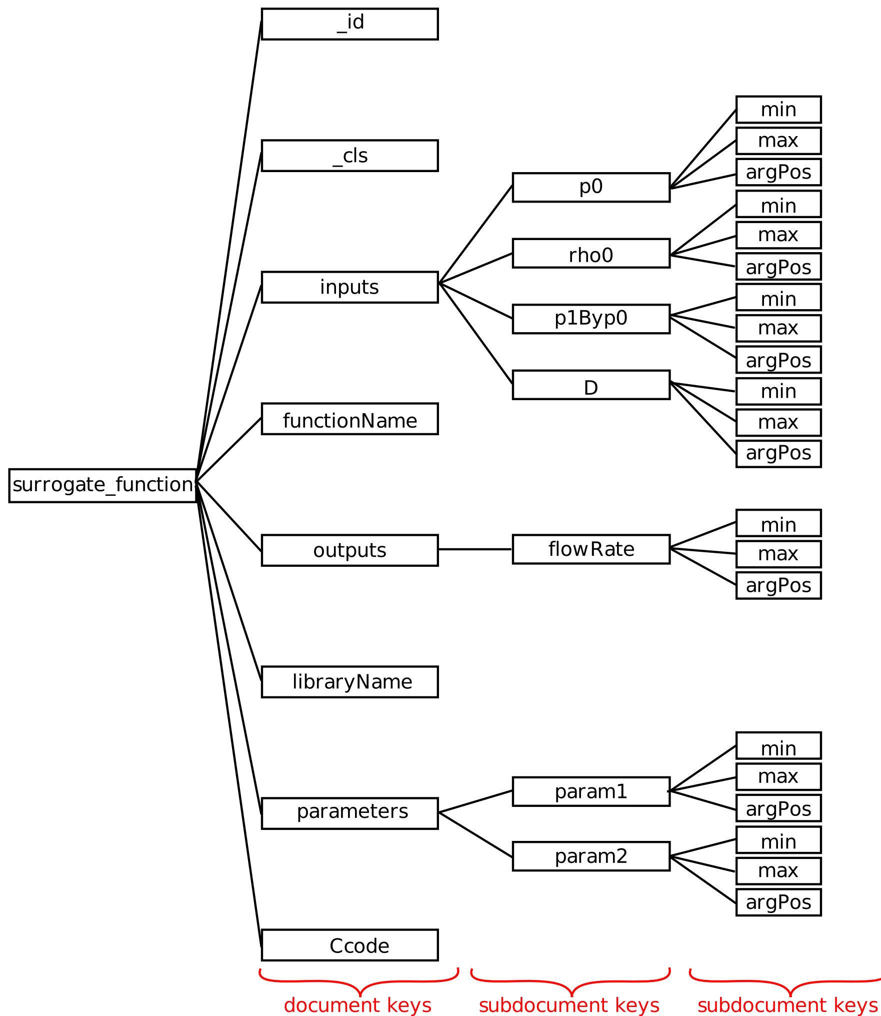

Figure 4: Data schema of surrogateFunction collection.

A member in the surrogate function collection is never modified during simulation, it contains input, output and parameter keys, as well as the global boundaries. Together, the global boundaries makes up the definition space, a continuous range in which the surrogate model can be evaluated. However, the surrogate model is not necessarily ready to be used in the entire definition space, i.e. the parameters does not have to be validated across the entire definition space. The MoDeNa framework allows the validated domain of the surrogate model to be expanded run-time during a simulation. The validated domain is defined in the surrogate model that references the surrogate function.

Surrogate Model

The surrogate model contains the strategies that gives the surrogate function its dynamic update capabilities. Specifically, in addition to a reference to the appropriate surrogate function it also references a detailed model whose input-output behaviour is being approximated by the surrogate model. Moreover, it contains strategies which the MoDeNa framework uses in order to fix non-initialised, inacurate or out-of-bounds parameters.

As shown in Figure 5 the surrogate model is much more comprehensive than the surrogate function from Figure 4. One important feature of the surrogate model entries is the `_id` field, which contains a unique string that the user of the MoDeNa software framework uses in order to include the surrogate model into their application.

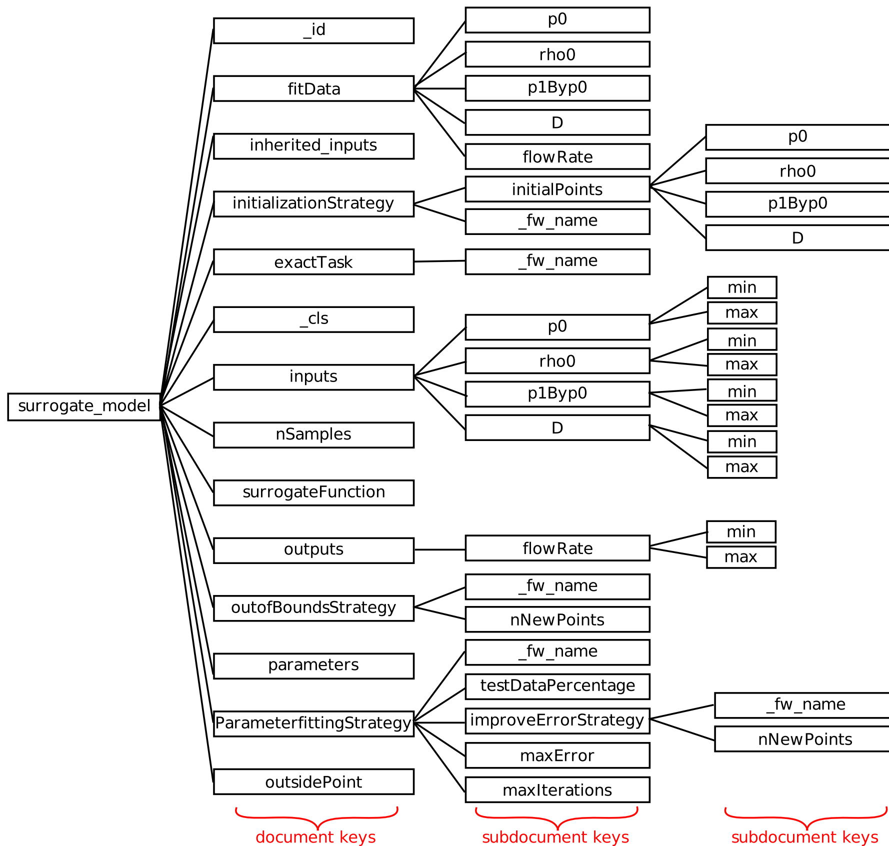

Figure 5: Data schema of surrogateModel collection.

The surrogate model receives simulation results from the `ModenaFireTask` and saves it in the `fit data` collection, as described in Rusche and Preisig (2 31). Moreover, the range of the inputs to the simulations, i.e. the `min` and `max` entries of the inpurt collection, is a subset of the global definition space. Consequently, the surrogate model defines the local domain which the parameters of the surrogate model has been validated, and is ready to be used.

The advantage of this structure is that the surrogate model collections are self-contained, and a surrogate model can be instantiated based on the information from the database. The surrogate model can also be re-initialised because the surrogate function contains the reference functional form.

Index set

The index sets were introduced in an earlier version of the MoDeNa software framework in order to allow a surrogate model to be functionally dependent on chemical species. A index set is a list of species mapping to all allowable inputs to a surrogate model, and the database schema is shown in Figure 6.

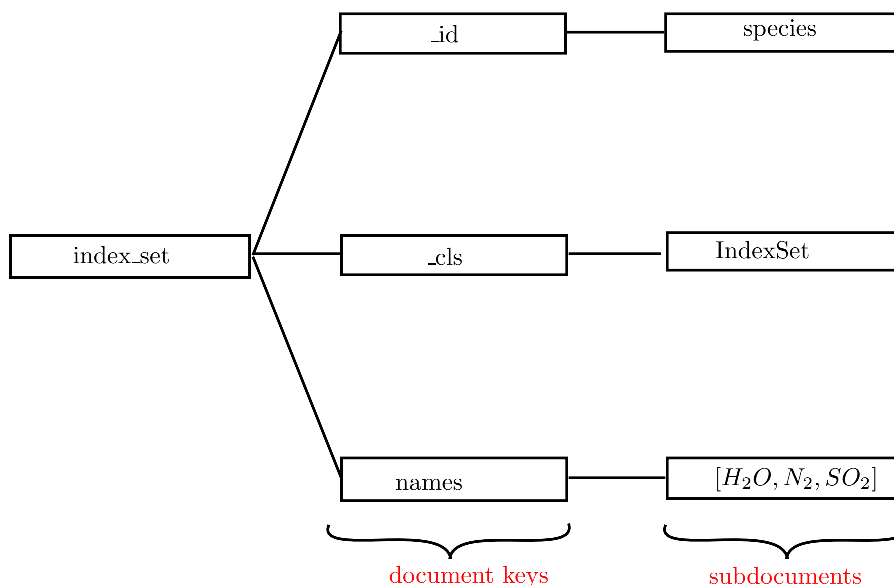

Figure 6: Data schema of the index set collection.

The concept of the index set is reflected in the surrogate model in two ways.

1. The surrogate model `_id`, e.g. `_id=myModel[A=H2O,B=CH4]`
2. The "species" sub-collection of the surrogate model, which defines the index set for the parameters `A, B`.

This implies that the index set document is referenced as a "specie" in the surrogate model, where it is associated with a parameter that is used in the name of the surrogate model. Hence, the name index set is justified in the sense that it maps all possible values that parameters can have.

The index set collection is, just as the surrogate function, constant. The idea is that the index set that belongs to a parameter is referenced in the definition of the surrogate model. Consequently, the surrogate model can have different index sets corresponding to different parameters. This situation is very common, i.e. a surrogate model can be thought of representing a number of acidic gases dissolved in alkanes. In this situation parameter `A` can correspond to one alkane in a index set containing all possible alkanes, and parameter `B` to a acidic gas in a different index set.

A natural extension to the index set is to link the concept to a standardised chemical data lookup system, e.g. simplified molecular-input line-entry system (SMILES). However, this feature has not been requested to be implemented in the current version of the MoDeNa software framework.

Data Management

The data management is handled entirely by the MoDeNa software framework, i.e. when the developer of an application defines surrogate function, models and index sets using Python templates, and embeds surrogate models into an application through the API, it is not necessary to manually update the database.

The detailed single-scale codes are wrapped into a class, which saves the results in the database. Moreover, the parameter fitting and error handling is triggered automatically when the recipes of the surrogate models are implemented in a detailed model. The details of the data structures, and the relationship between the high-level Python module, and the low-level C library has been documented in Deliverable Rusche and Karolius (5 13).

In short, the database communication is handled through Python, the role of the C-library is to evaluate the surrogate models and provide callbacks to Python when an error occurs. Therefore, when a surrogate model is instantiated in the detailed model, the C-API uses the Python library to get the surrogate model from the database, but afterwards there is no further communication with the database until an error is detected.

The communication with the database is done using the python package `mongoengine`, which maps Python objects into MongoDB schemas. Technically, the MoDeNa framework consists of objects that are inheriting from the mongoengine base-classes. In this way, the low-level API can communicate with the database through the Python objects.

However, the MoDeNa framework has been designed such that it pulls all the necessary information from the database when a model is instantiated, and as long as the parameter estimation framework is not invoked there is no further communication with the database. In the event that one-or-more surrogate models must be fixed by the parameter estimation framework the database will be updated sequentially with input-output data from detailed simulations, as well as surrogate model parameters. However, this is an automated procedure that the user of a model is only capable of influencing by choosing different strategies from the parameter estimation framework as described in Deliverable Preisig et al. (2 31).

The point that this section is making is that the data-management structure of the MoDeNa software framework can be considered as divided into two parts:

- Passing information between Python and C
- Storing and retrieving information in the database

This divide is used in the MoDeNa framework to avoid querying the database needlessly.

The mongoengine object is used to fetch all surrogate models and instantiate Python objects that can share information through the MoDeNa API.

This may seem like an unnecessarily complicated structure, but when considering that the MoDeNa framework may be employed in a massively parallel manner a direct database connection is simply unfeasible. Moreover, as stated in the beginning, the purpose of the MoDeNa framework is speed, and relying on waiting for database queries for data passing would be a unnecessary time-delay.

This part of the framework is simultaneously the most important and the most abstract idea to convey. Even though a multiscale modeling framework can be organized in many ways it can not rely on direct database communication for information passing.

Discussion

Even though the database has a central role in the MoDeNa software framework the most important concept to note from this report is when the database is **not** used. The reason for this is that developers would be hesitant to use a software that relies on querying a database. Modern database systems are fast and capable of handling large traffic, but in a software whose purpose is to be fast it would be counterintuitive and unnecessary to spend time waiting for the database. This is particularly important in the case that the database is hosted remotely.

In order to avoid needlessly querying the database, the MoDeNa framework initialises python objects based on the data that is stored in the database. Therefore, in the case that information is needed from the database the local Python object already contains all the information and it is not necessary to query the database.

The data schema of MoDeNa is designed to be modular and flexible. The functional form, i.e. code, inputs, outputs and boundaries, is defined as surrogate functions, and the surrogate function is referenced in the surrogate model. The flexibility of the data schema is facilitated by the python module mongoengine, and the schema is used to instantiate a Python object, whose member functions can be called through the MoDeNa API.

The most important aspects from this report is the role of the database in a simulation that employs the MoDeNa framework.

References

Jain, A., Ong, S. P., Chen, W., Medasani, B., Qu, X., Kocher, M., Brafman, M., Petretto, G., Rignanese, G.-M., Hautier, G., Gunter, D., and Persson, K. A. (2015). Fireworks: a dynamic workflow system designed for high-throughput applications. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a.

Karolius, S., Birgen, C., Preisig, H. A., and Rusche, H. (2015-08-31). Modena - deliverable d4.2: Model and data containers. Technical report, FP7 MoDeNa.

Preisig, H. A., Thombre, M., and Karolius, S. (2014-12-31). Modena - deliverable d5.3: Parameter estimation framework. Technical report, FP7 MoDeNa.

Rusche, H. and Karolius, S. (2015-05-13). Modena - deliverable d5.8: Production release - recipes and adapters. Technical report, FP7 MoDeNa.

Rusche, H. and Preisig, H. A. (2014-12-31). Modena - deliverable d5.2: Orchestrator v0: First release of orchestrator, database structure and interface library for the modena software. Technical report, FP7 MoDeNa.