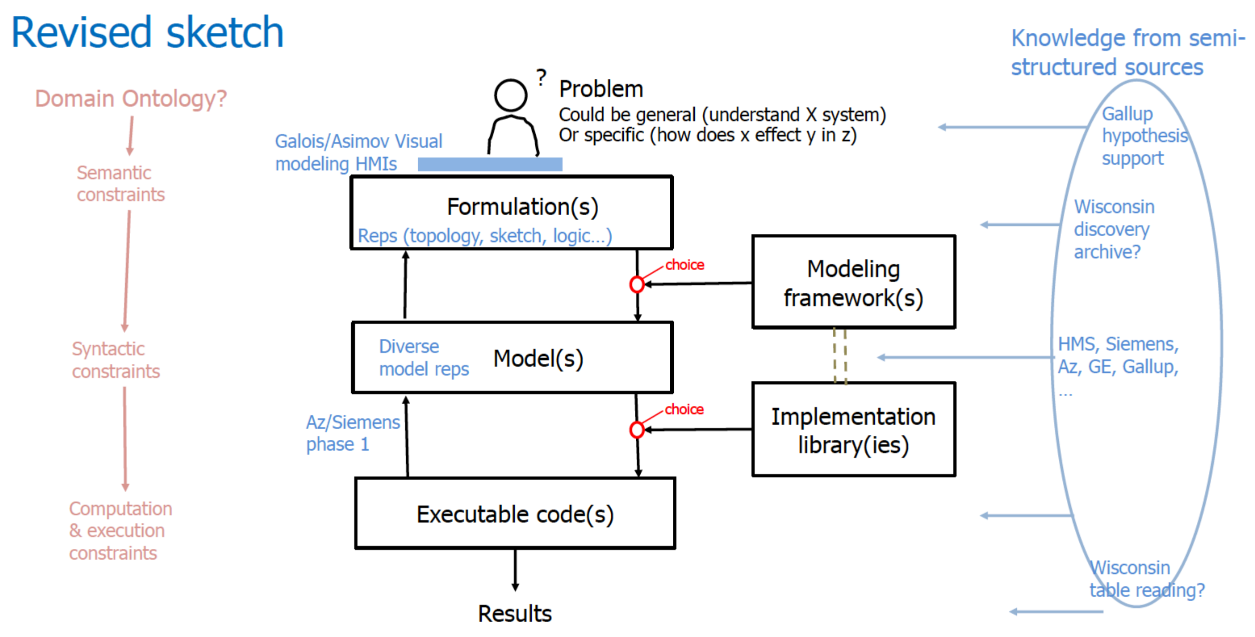
ASKE Kapeesh Description

Alfredo Gabaldon, Natarajan Kumar, Andrew Crapo, Vijay Kumar

This is a summary description of the project with respect to the new representation diagram that came out of the Phase 2 kick-off meeting.

We will first summarize our understanding of the three layers in the revised representation diagram.


Revised sketch
Domain Ontology?
Semantic constraints
Syntactic constraints
Computation & execution constraints
Galois/Asimov Visual modeling HMIs
?
Problem
Could be general (understand X system)
Or specific (how does x effect y in z)
Formulation(s)
Reps (topology, sketch, logic...)
choice
Modeling framework(s)
Diverse model reps
Model(s)
Az/Siemens phase 1
choice
Implementation library(ies)
Executable code(s)
Results
Knowledge from semi-structured sources
Gallup hypothesis support
Wisconsin discovery archive?
HMS, Siemens, Az, GE, Gallup, ...
Wisconsin table reading?

Formulation Layer: this layer contains knowledge about scientific concepts represented in a way that is independent of a particular modeling framework. It is thus missing information required to transform abstract models, for example a set of equations/functions, into a model based on a particular framework such as DBNs, ABMs or some other type of framework. This layer does contain semantic constraints which restrict the set of abstract models (e.g., equations) that can be combined together into a larger model. For example, two equations can be combined if their shared variables are of the same semantic type and use the same units or provisions are made for suitable conversion. Another semantic constraint could involve assumptions or preconditions required by the submodels. For example, if an equation that involves air temperature applies under the assumption that the object of interest is located within the troposphere, then other equations intended to be part of the model should be applicable under that condition as well.

Models Layer: models at this level are expressed or serialized for a particular framework. This involves adopting a particular syntax and syntactic constraints. It also involves providing additional information required by the framework to execute the model.

Executables Layer: Models at this level are in executable form. The simplest example is code. But we would also include here problem specifications in the required input format of an inference engine, such as a DBN specification for a DBN execution framework or an encoding of a DiffEq in the format required by a DiffEqSolver. In the case of problem specifications, the difference between the structured and execution layer representations is mostly format.

Description of ASKE Kapeesh project

Scientific Knowledge
Scientific knowledge is represented in a knowledge graph in a form independent of any particular execution framework (formulation layer). The knowledge graph contains domain specific knowledge defining domain concepts, e.g., Temperature, and equations (models) encoding the dependencies among the concepts, e.g., the relationship between the TotalTemperature of the air around an object and the Altitude and Speed of the object. The knowledge graph also includes domain independent knowledge such as the concepts of UnittedQuantity and Equation, and relations such as 'argument' (or input) between equations and unitted quantities.

Computational Graphs
The knowledge graph also contains the concept of ComputationalGraph and a DBN subconcept. These are used to capture models at the models layer level, where abstract models are extended by linking them to DBN (or other framework) specific information such as Node, NodeType, Distribution, Range, etc. New subconcepts of ComputationalGraph can be added to incorporate other frameworks (K-Chain, ABMs, etc).

Model Assembly (or sub-selection, or problem solving)
Each equation or function in the knowledge graph can be seen as a single model. Given a goal to model a Y from a set of X's (a query), the system looks for a set of equations that together form a complex model of Y from the X's, provided semantic constraints are satisfied by the set of equations (some ASKE teams use “model assembly” to refer to the process of assimilating a newly extracted piece of knowledge into the domain knowledge base. We should probably use a different term to avoid confusion). In addition to satisfying the semantic constraints, making the model self-consistent, the assembled model may be valid only under the context or assumptions stated in the query, for example, that the phenomenon is occurring in the troposphere. The submodels of the selected model may be connected to other submodels forming a potentially larger model, but for a particular modeling task or query, they may have been excluded (Figure 1). This process occurs at the formulation layer.

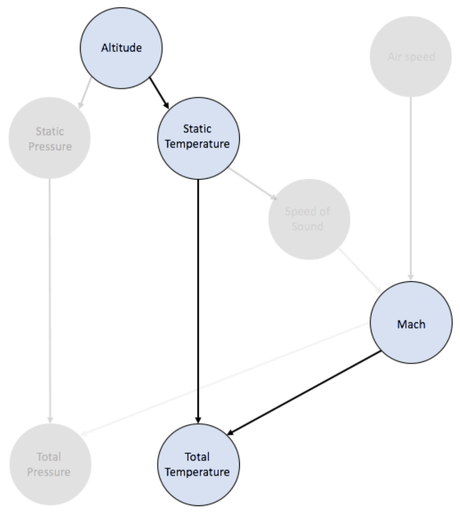


Figure 1 Model assembled for a Total Temperature query. Shadowed nodes correspond to submodels not used.

Human-Machine Interaction

We use the SADL Dialog interface for user interaction with the system. This interaction occurs mostly at the formulations level, but we have also implemented query forms that let the user access information at the models level, for example to inspect the model the system assembled to answer a query. After the models are executed, they are displayed graphically to the user as networks of variables with their computed values.

Dynamic Bayesian Network Execution Framework

As described in previous reports, we have been using a DBN framework for executing models. We use the DBN framework as an external inference engine. The engine accepts DBN modeling tasks in Json format, so we generate such specs instead of source code. The spec is generated from the output of the model assembly task described earlier by adding DBN execution parameters, some retrieved from structured knowledge, some determined from the query and selected model (e.g., prognostics vs calibration), and some default parameters.

Modeling Knowledge, Meta-data/reasoning

We also have the beginnings of another layer (or vertical?) where we capture knowledge about the modeling process itself. This knowledge "talks" about how the scientific knowledge has been used, so it includes a concept for Query, for Model, and relations to capture the input values given in a query, the models that were used to answer the query, and the results of the model executions, etc. This will get more interesting when we extend the system to allow more user intervention. For example, if the user modifies the model constructed by the system before it is executed, this operation would also be captured. The intention is that this knowledge will capture the workflow or "paper trail" from initial query to final model and results. This will enable the system to assist a user going through similar modeling steps. Also important: this knowledge can be queried and reasoned over.

Hypothesizing Models of Datasets

Another way the user can pose a problem to the system is by providing a dataset in CSV format and a semantic description of the dataset using scientific concepts from the knowledge graph to define the type of each column. When the user asks for models that may explain the data, the system takes the set of column types and runs model assembly attempts using different inputs/output combinations. After running any models found, the system stores meta-data about the dataset, the constructed models and their accuracy (what layer does this go?).

Representation Examples

The representation examples are given in SADL for the formulations and models layers and in Json for the executables layer.

==Formulations Layer==

```
ScientificConcept is a class.
UnittedQuantity is a type of ScientificConcept,
      described by ^value with values of type decimal,
      described by stddev with values of type decimal,
      described by unit with values of type string.
^Equation is a class
   described by expression with values of type Script
   described by arguments with a single value of type DataDescriptor List
   described by returnTypes with a single value of type DataDescriptor List.

AtmosphericTemperature is a type of UnittedQuantity.
TotalTemperature is a type of AtmosphericTemperature.

TotalTemperatureEquation is a type of ^Equation.

TotalTemperatureEq1 is a TotalTemperatureEquation
sciknow:input (a Argument argType StaticTemperature argName "ts0")
sciknow:input (a Argument argType MachSpeed argName "mach")
sciknow:output (a TotalTemperature)
expression (a Script with script "ts0*(1.0 + 0.5*(1.4-1.0)*mach*mach)", with
language Text).
```

==Models Layer==

```
ComputationalGraph is a class.
argType describes Argument with a single value of type ScientificConcept.
argName describes Argument with a single value of type string.
sciknow:input describes ^Equation with values of type Argument.
sciknow:output describes ^Equation with values of type ScientificConcept.

DBN is a type of ComputationalGraph
      described by node with a single value of type Argument
      described by hasEquation with a single value of type ^Equation
      described by hasModel with a single value of type ^ExternalEquation
      described by ^type with values of type NodeType
      described by distribution with a single value of type Distribution
      described by range with values of type Range.

TotalTemperatureDBN is a type of DBN.
hasEquation of TotalTemperatureDBN only has values of type
TotalTemperatureEquation.
range of TotalTemperatureDBN always has value (a Range with lower 359 with
upper 2000).
distribution of TotalTemperatureDBN always has value uniform.
```

==Executables Layer==

This is just a snippet of the DBN specification for a modeling query for TotalTemperature, in the format required by the DBN execution framework.

```
{"outputs": [],
 "techniqueName": "dbn",
 "mapping": {
  "ts0": "http://aske.ge.com/hypersonics#StaticTemperature",
  "altitude": "http://aske.ge.com/hypersonics#Altitude",
  "tt26": "http://aske.ge.com/hypersonics#TotalTemperature",
  "mach": "http://aske.ge.com/hypersonics#MachSpeed",
  "u0": "http://aske.ge.com/hypersonics#AirSpeed",
  "a0": "http://aske.ge.com/hypersonics#SpeedOfSound"
 },
  "DBNSetup": {
   "PlotFlag": false,
   "TaskName": "Prognosis",
   "TrackingTimeSteps": 1,
   "ParticleFilterOptions": {
    "BackendFname": "/tmp/Results",
    "Backend": "RAM",
    "ResampleThreshold": 0.4,
    "BackendKeepVectors": true,
    "BackendKeepScalars": true,
    "Parallel": true,
    "NodeNamesNotRecorded": [],
    "ParallelProcesses": "5"
   },
   "WorkDir": "/tmp",
   "NumberOfSamples": 500
  },
  "Nodes": {
   "tt26": {
    "Type": "Deterministic",
    "DistributionParameters": {},
    "ModelName": "tt26",
    "Distribution": "",
    "Parents": [ "mach", "ts0"  ],
    "Tag": [],
    "Children": []
   },
   "ts0": {
    "Type": "Deterministic",
    "DistributionParameters": {},
    "ModelName": "ts0",
    "Distribution": "",
    "Parents": ["altitude"  ],
    "Tag": [],
    "Children": ["a0", "tt26" ]
   },

  "Models": {
   "tt26": {
    "FunctionName": "tt26",
    "Input": [ "mach", "ts0" ],
    "Type": "Equation",
    "Output": [],
    "ModelForm": "ts0*(1.0 + 0.5*(1.4-1.0)*mach*mach)"
   },
   "ts0": {
    "FunctionName": "ts0",
    "Input": [ "altitude" ],
    "Type": "Equation",
    "Output": [ "a0", "tt26" ],
    "ModelForm": "518.6-3.56 * altitude /1000.0"
   },
 "taskName": "build",
 "dataSourceIds": {},
 "workDir": "/tmp",
 "dataSources": {}
}
```

==Process ==
This is content of what could conceptually be another layer that captures modeling process information as described earlier in the Modeling Knowledge paragraph.

```
SubGraph is a class
      described by cgraph with values of type ComputationalGraph
      described by mm:output with values of type UnittedQuantity.

QueryType is a class must be one of {prognostic,calibration,explanation}.

CGQuery is a class
      described by queryType with a single value of type QueryType
      described by mm:input with values of type UnittedQuantity
      described by mm:output with values of type UnittedQuantity
      described by machineGenerated with a single value of type boolean
      described by execution with values of type CGExecution.

CGExecution is a class
      described by startTime with a single value of type dateTime
      described by endTime with a single value of type dateTime
      described by compGraph with values of type ComputationalGraph
      described by accuracy with values of type decimal.
```

```
CSVDataset is a type of Dataset
      described by column with values of type CSVColumn.

CSVColumn is a class
      described by header with values of type string
      described by colType with values of type string
      described by variable with values of type UnittedQuantity
      described by percentMissingValues with values of type float.
```