DARPA ASKE GE Team Final Report

Andrew Crapo, Alfredo Gabaldon, Narendra Joshi, Vijay Kumar, Varish Mulwad, Nurali Virani
GE Research
June, 2020

This work is supported by the Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR00111990006 and under Agreement No. HR00111990007. This report combines the Final Report for GE TA1 and GE TA2. The latest version of this report is available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/GE_ASKE_Final_Report.pdf

and at

https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/tree/master/Reports/GE_ASKE_Final_Report.pdf

Table of Contents

Introduction

The purpose of this final report is to complete reporting of the GE ASKE System applied to the target sources of code and text, the NASA Glenn Hypersonics Web Site, and to report the results of applying the GE ASKE System to another domain, specifically that of wind turbine design. This report extends but does not repeat what was covered in the M13 Report, available at the locations below, and other prior reports accessible on the project Wiki (https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/wiki).

M13 Report Links

- https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/M13/GE_ASKE_M13_Report.pdf
- https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/tree/master/Reports/GE_ASKE_M13_Report.pdf

Accomplishments of GE ASKE – Novel Capabilities and Key Outcomes

The key concepts of our approach are shown in Figure 1. Here, the Semantic Model of Knowledge contains the extracted knowledge from scientific code, associated text, and inputs or corrections from human experts. An end-user's query causes the just-in-time composition of the model, which is then used to answer the user's query and to provide additional insights.

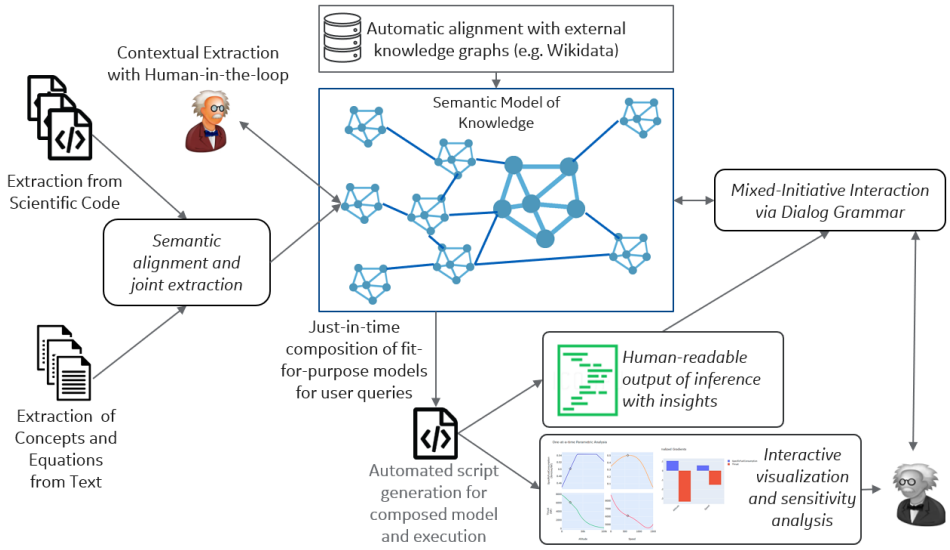

Figure 1: GE-ASKE Overview of Key Novel Capabilities

1. Mixed-Initiative Dialog Using Controlled English

We have found controlled English to be a useful mechanism for making formal semantic models more transparent and understandable to subject matter experts[1]. In the GE ASKE System, we have extended the controlled-English grammar to support the kinds of questions, answers, and

[1] See "SADL github Wiki," [Online]. Available: https://github.com/crapo/sadlos2/wiki. See also Semantic Application Design Language (SADL) at http://sadl.sourceforge.net/.

assertions necessary to the capabilities of ASKE TA1 and TA2. More importantly, we have granted full access to the Dialog Editor Window to the Curation Manager Agent so that either the human user or the artificially intelligent backend agent can ask for or provide information to the other at any time. The conversation in the Dialog Window is organized around anchor statements or questions. For example, if the user asks a question which the System cannot answer because of missing information, the dialog that occurs around the original question and the missing information stays clustered together in the window. While the conversational interface is only a proof-of-concept and worthy of much greater development investment[2], it illustrates the essential capacity to carry on a collaborative, mixed-initiative interaction between man and machine in which neither is the slave of the other but both contribute to arriving at the desired result.

2. Semantic Context and Constraints in the Knowledge Graph

Every scientific theory, reduced to a computational model whether deterministic and physics-based or statistical and data-driven, carries with it a context and a set of constraints or assumptions. Without an understanding of this context and the constraints/assumptions of the model, no agent, human or artificially intelligent, can ensure proper use of the computational model. Consider two simple examples from the hypersonics domain. One is the computation of Mach number, which is the ratio of the speed of an object to the speed of sound in air. What is not usually explicit in the model is that the speed of sound must be for the medium through which the object is passing and that the speeds must be in the same units. A second example is the model to approximate the temperature of the earth's atmosphere as a function of altitude. The NASA Web Site's earth atmosphere model consists of three equations, each applicable to a different range of altitudes. The altitude range of each serves as a constraint on when the equation can appropriately be used. The GE ASKE System makes these contexts, constraints, and assumptions, as well as others, explicit and actionable by the artificial intelligence. For more details, see Capturing Assumptions and Constraints in the M9 Reports.[3]

3. Integrated Extraction from Code and Text Using Locality Search

The key novelties in Knowledge Extraction from Text include the ability to extract equations from text documents, converting the extracted equations into executable Python code and extracting context associated (e.g. the semantic types) with the equation variables. Extracted equations are captured into a "local" semantic graph associated with a given (or a set of) document(s). We have demonstrated that equations extracted from text can be chained with the scientific models extracted from code to answer complex questions. Furthermore,

---

[2] See requirements document for Dialog framework at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/SupportingFiles/A%20Mixed%20Initiative%20Dialog%20Framework.pdf

[3] The M9 reports are available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/M9_ANSWER_Report.pdf and https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/blob/master/Reports/ASKE_TA2_Project_KApEESH_M9.pdf .
See also the original documentation of the approach at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/Ontology/UsingGraphPatternToAugmentIntegrationOfModelsIntoSemanticFrameworkV2.pdf.

knowledge extraction from text is also leveraged to enhance the understanding of the scientific models extracted from code. Equation variables in code have little or no context to interpret their semantic meaning. We demonstrated that the semantic types associated with such variables can be extracted from headers and comments surrounding the variables.

4. Just-in-time Scientific Model Composition

As a result of the extraction process, a collection of semantic descriptions of scientific models will be stored in the knowledge graph. These models will be available to answer user queries on demand. The GE ASKE system uses this knowledge to perform just-in-time assembly of new models to answer user queries without the need to modify model code directly. Figure 2 shows the diagram of a composed model for the total thrust of an aircraft. These composed model diagrams are also part of the query answers presented to the user.

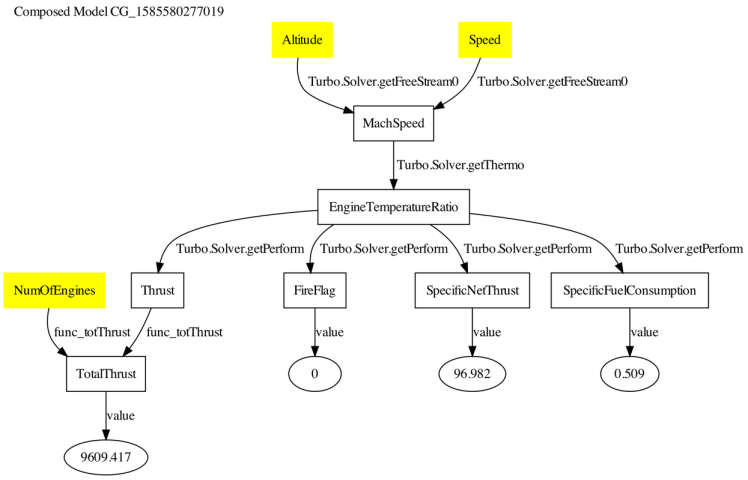

Figure 2: Diagram of an assembled model for computing the total thrust of an aircraft

More details on just-in-time model composition can be found in the M3 report and a publication.[4]

5. Automated computational script generation and execution

The result of just-in-time scientific model composition is a subset of assertions drawn from the knowledge graph, including parent-child connections between *nodes* (e.g., MachSpeed in Figure 2) in the assembled model, knowledge of constituent *models* (e.g., getThermo in Figure 2) and their respective inputs/outputs, dependencies, and *expressions*, i.e., the computational logic associated with the constituent model. Here, an expression could be an inline equation, code extracted from a scientific code base, or code snippets generated from extracted or human-

[4] The M3 report is available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/blob/master/Reports/ASKE_TA2_KApEESH_M3_Report.pdf . See also: Gabaldon, A; Kumar, Natarajan. "Knowledge-driven Model Assembly and Execution" Modeling the World's Systems Conference, Washington DC, 2019 available from https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/blob/master/Reports/MWS19final.pdf .

provided equations. A Model Execution Request Service then takes these assertions and automatically generates a computational graph script to be carried out by a model execution framework such as K-CHAIN. To do so, this service performs a *topological sort* of the nodes in the assembled model to determine the correct order of invocation of methods within the computational script. Additionally, this service establishes appropriate scoping of implicit variables within the code, ensures that global variables are declared or initialized as required, reconciles any inconsistencies in the names and labels used to reference the same variable across different methods, and ensures that any initializer or dependency methods are invoked at the right time within the script. When the feature is enabled, this service also implicitly handles unit conversion in the assembled model by automatically injecting unit-conversion code into model scripts. Details on unit handling can be found on the M11 report.[5] This created script is then executed in Python with suitable inputs from user query to derive outputs and conduct sensitivity analysis.

6. Automated Interactive Visualization of Model Sensitivity Analysis

Along with responding to user query, our system automatically performs a few sensitivity analyses that are display as interactive plots to the user to derive insights and validate or understand model behavior. Three salient features are as follows:

1. The interactive visualizations are generated procedurally based on user query to determine number of subplots, their arrangement, reference values, and the axis labels (with units).
2. The values for plots are obtained by interacting with the K-CHAIN model execution service by automatically constructing suitable REST API calls.
3. The Python NumPy[6] version of model code allows us to use *autograd*[7] to compute local gradients or Jacobians of the code outputs with respect any number of inputs to provide model sensitivity insights.

Examples of interactive visualizations are shown below in Figure 3 (left – one-at-a-time parameter sweep from min to max value of that parameter, right – one-at-a-time parameter sweep from -10% to +10% of the reference value based on user query).

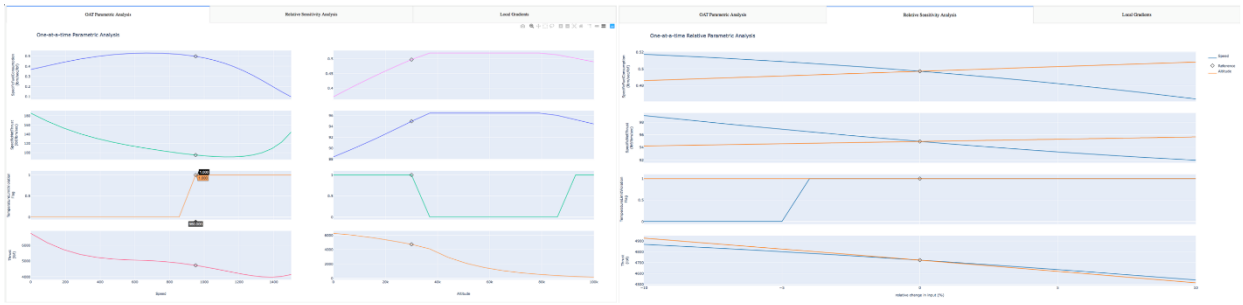

Figure 3: Insights through interactive visualizations

[5] The M11 report is available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/blob/master/Reports/ASKE_TA2_Project_KApEESH_M11.pdf .

[6] See NumPy at https://numpy.org/.

[7] A third-party open source package available at https://github.com/HIPS/autograd.

7. Human-readable Insights Generation

The GE ASKE system takes the composed models and executes them not only to compute a direct answer to a user query, but also to perform analyses that lead to a deeper understanding of the model and the results for the user. The system performs a number of sensitivity analyses that are displayed as interactive plots and also generates readable insights in natural language. The GE ASKE system currently supports these types of insights:

- Increasing/decreasing-increases/decreases: describe how a change of value in a model input affects an output.
- Sensitivity and slope: indicate when an output is particularly sensitive to changes in an input with positive/negative slope, i.e., the direction of change.
- Local extrema: point out local minima/maxima of an output around the region of the query point.
- Operating conditions transition points: indicate whether a model output is near a transition into or out of a safety range. This type of insight is currently domain specific for the hypersonics domain.

Figure 4 shows an example query and answer with insights.

```
what is the thrust of a CF6 when the altitude is 30000 ft and the speed is 900 mph?
CM:  a Thrust with ^value 4804.7 lbf .
       (See 'model diagram: "file:///Graphs/CF6_CG_1588177140447"', 'sensitivity plot
    Increasing Altitude increases SpecificFuelConsumption.
    Increasing Speed decreases SpecificFuelConsumption.
    Increasing Altitude increases SpecificNetThrust.
    Increasing Speed decreases SpecificNetThrust.
    Decreasing Altitude takes the CF6 outside operating conditions.
    Increasing Speed takes the CF6 outside operating conditions.
    Increasing Altitude decreases Thrust.
    Increasing Speed decreases Thrust.
```

Figure 4: A query answer example with human-readable insights

8. Rapid generalization and demonstration in multiple domains

The GE ASKE system was initially developed and demonstrated using NASA Engine simulation codes and NASA Hypersonics index webpages. These models were primarily in the domain of thermodynamics for jet engines. In a period of 3 weeks, we implemented and demonstrated our system on wind turbine modeling problem in the domain of aerodynamics of wind turbine blades. More details on this generalization use case is shown later in this report.

Final Prototype Architecture

This section will describe the overall GE ASKE System architecture and then go into some detail on the extraction components.

Figure 5 shows the final GE ASKE System architecture.

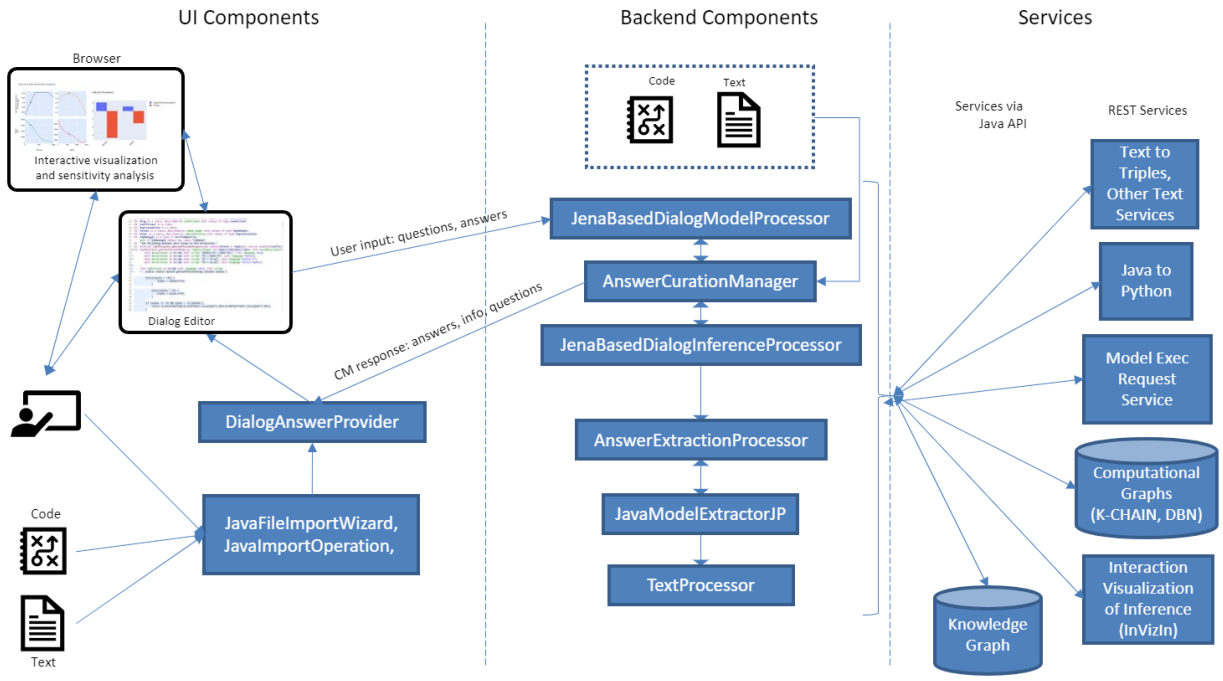

Figure 5: GE ASKE System Architecture

The most visible UI components include the Eclipse-based Dialog Editor and a Web browser. The Dialog Editor is Xtext-based[8], extends the SADL IDE, and has a Dialog grammar which is an extension of the SADL grammar. The Web browser displays sensitivity plots and other graphs. In addition, the JavaFileImportWizard and associated operation are modal dialogs that allow selection of text and/or code files for ingestion for knowledge extraction.

An essential UI component which is not directly visible to the user is the DialogAnswerProvider. This class handles all insertions of text coming from the backend into the Dialog Editor window. It is also responsible for displaying model graphs, although they are displayed in a separate Eclipse window or in a browser.

There are two backend components with which the UI interfaces. The JenaBasedDialogModelProcessor is the Xtext model processor component and is used to convert the parsed Dialog Editor content into an OWL model and a set of conversation elements specified in the Dialog content. It is an extension of the JenaBasedSadlModelProcessor and much of its functionality is inherited from this Open Source class in the SADL project. The conversation elements found in the Dialog content are passed to the AnswerCurationManager (ACM) for processing. The ACM calls on other classes to process conversation elements. Those that require inferencing over the knowledge graph (KG), possibly composition and execution of computational graphs (CG), and querying of the KG are passed to the JenaBasedDialogInferenceProcessor. This class is an extension of the JenaBasedSadlInferenceProcessor, from which it inherits generic inferencing and querying

[8] Xtext is a framework for development of programming languages and domain-specific languages. See https://www.eclipse.org/Xtext/.

capabilities. Conversation elements that initiate extraction from text and/or code are passed to the AnswerExtractionProcessor, which will in turn make calls to the JavaModelExtractorJP or the TextProcessor.

The ACM communicates back to the DialogAnswerProvider in two important ways. First, it returns answers to the user's questions. These answers are inserted into the Dialog Editor window at the appropriate location and can be expressed in the Dialog grammar, which ensures that domain concepts referenced in the answer's contents are defined in the semantic model and are hyperlinked to their KG definitions and references, or they may include quoted text for the reader's consumption as entirely natural language not constrained by the Dialog grammar. The second type of communication is questions for the user. For example, if an equation is extracted but the semantic type of an argument or of the returned value is not known, the user will be asked for clarification either by direct input or through additional extraction from code and/or text. Sometimes the communication is not a question but just a statement such as notification that a query could not be processed because no model was found to do some part of the computation. Such a statement serves as an invitation to the user to help identify the missing model either by extraction from some source or by direct input.

The backend of the GE ASKE System depends on several services as shown in Figure 1. These include the following.

- TextToTriples and other text services are invoked to extract knowledge from documents and from code comments. Details of this service are described in the next section.
- JavaToPython converts Java methods extracted from Java code and thought to be of interest to Python. This is necessary because the CG expects Python code.
- The Model Execution Request Service receives a set of knowledge graph facts about a composed model and generates an input payload in the format required by the model execution framework (DBN/K-CHAIN). When the feature is enabled, this service also handles unit conversion by injecting conversion code into model scripts.
- The Computational Graph (CG) provides services to add knowledge to the graph and to execute that knowledge to do computations for query answering. The initial CG implementation was in a GE-proprietary Dynamic Bayesian Network (DBN). The final CG implementation is in K-CHAIN, which is implemented using NumPy and TensorFlow[9]. For additional details on K-CHAIN capabilities see the "Computational Modeling in GE ASKE System: Comparison and Characteristics" section in the M13 report.[10]

[9] TensorFlow is an end-to-end open source platform for machine learning. See https://www.tensorflow.org/.

[10] The M13 report is available at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase2/M13/GE_ASKE_M13_Report.pdf.

Knowledge Extraction from Text Pipeline

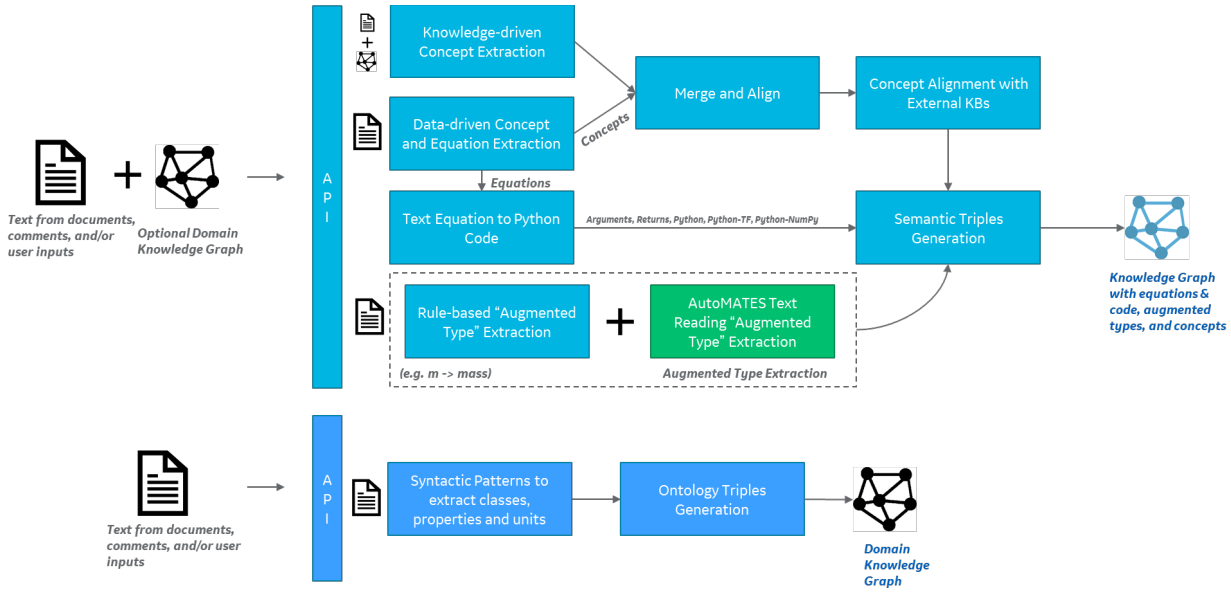

Figure 6: GE ASKE Knowledge Extraction from Text Pipeline

Knowledge Extraction from Text consists of two main APIs along with additional helper APIs.

The **text2triples API** extracts scientific concepts and equations that appear in text (sentence/paragraph/document). Additionally, it converts the extracted equations into executable Python code and extracts the semantic types (also referred to as augmented types) associated with the input and return variables wherever possible. We developed two parallel independent approaches to extract scientific concepts from text – a) Knowledge-driven Concept Extraction approach and b) Data-driven concept and equation extraction approach. Details of the data-driven concept and equation extraction approach can be found in our M3 report[11]. This current report covers the details for the remaining components of the Knowledge Extraction from Text pipeline.

Knowledge-driven Concept Extraction: The Knowledge-driven Concept Extraction component leverages a domain ontology provided by the user. The classes and properties in the domain ontology are used to dynamically create a dictionary to be used with Apache UIMA

[11] https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/StatusReports/Phase1/DARPA-ASKE-ANSWER-M3-Report.pdf

ConceptMapper.[12] ConceptMapper uses this dictionary to identify references to scientific concepts appearing in the text.

Merge and Align: Leveraging both knowledge-driven and data-driven concept extraction methods often leads to duplication. Duplicate concepts are merged together by the comparing the names extracted by the data-driven and knowledge-driven approaches.

Concept alignment with external KBs: Extracted concepts are first aligned with existing entities from Wikidata, thus providing additional context to the extracted knowledge. The alignment process involves two steps: a) generating a list of Wikidata candidates for a given scientific concept and b) re-ranking the candidates. The scientific concept's name is used as part of Elastic Search's match query to generate an initial list of Wikidata candidates. Details of the candidate generation process can be found in our M5 report[13]. The candidates are further re-ranked by computing Sørensen–Dice coefficient score between the concept name and the Wikidata entity name. The candidate list is sorted based on this similarity score and the top-ranked candidate (if the score is greater than 0.9) is chosen as the concept's Wikidata entity alignment.

Text to Python: This module converts a text equation into a piece of Python code which represents a function that can be run by a regular Python interpreter. The module is essentially a parser. It scans the input text string from left to right and creates the equivalent code as it does so.

The module assumes that the input text is in the form of a string that represents a mathematical equation of the generic format "expression1 = expression2" where expression1 and expression2 can both have variables, constants, and operators, e.g., "a + b = c * (2.5 ^ (1 – alpha))". The operators are the regular mathematical operators such as +, -, /, *, ^, etc. Additionally, we also consider e^ as the exponent operation. We assume that the input text string is a valid text string. If, for instance, the input text string is missing a bracket, the module will behave incorrectly or fail since it does not attempt to rectify such errors.

The parser works by processing the input text string one character at a time. Based on whether the character is an alphabetic character (a-z,A-z) or a numeric one (0-9) or an operator (+,-,/,*,^) or a period (.), it determines what to do next. It also considers the next character and the previous character in making these determinations. For example, if there is a '2' followed by a '.', this could mean it is the beginning of a floating-point number "2.". If it is a '2' followed by a ')', this could mean it is the end of an expression and in that case, it has to backtrack to find the matching '(' that marks the start of the expression. For more than one bracket like characters, e.g., "2 – b) }", the parser has to backtrack till it fills all matching closing brackets. All bracket like characters are converted to ')' and '(' since those are the legitimate characters in Python.

[12] Tanenblatt, M. A., Coden, A. and Sominsky,I. L. 2010. The ConceptMapper Approach to Named Entity Recognition. In Language Resources and Evaluation Conference (LREC).

[13] https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/wiki/Extraction-of-Scientific-Knowledge-from-Text

Sets of characters such as “exp” and “log” are not treated as simply characters but as mathematical operators and converted appropriately.

The easiest to convert are simple equations where there is a single variable on the left hand side and simple operations such as +, - , *, / on the right hand side. Slightly more complex equations such as having an operator on the left-hand side are also handled without too many issues. The complicated ones are usually the ones that involve the exponent operations, e.g., ^ or e^. The tricky part is to backtrack from the ^ to figure out what text segment is the ^ operator being applied to. Especially complicated cases include having to deal with the unary negation operator in addition, e.g., e^-(1 + gamma) or e^-T.

Currently, it assumes everything is literal text. Therefore, expressions such as “exp” or “degrees” occurring in the text string will be translated as is without interpretation. The converter module is essentially a parser that works with whatever it knows of currently. Therefore, whenever it encounters a new condition that it has not seen before, it will fail. However, new functionality to handle new cases can be built in.

The input is a single string that represents an equation in text format. The output has several pieces of information. In addition to the converted Python code function, it also returns the extracted left-hand-side and right-hand-side expressions, the original input text string, and where applicable, multiple code interpretations of the text string. For instance, if the left hand side has a mathematical operator, e.g., “a * b = c + d”, the code will return two equations – one corresponding to “a = (c + d) / b” and the other corresponding to “b = (c + d) / a”.

The module is made available as a REST API service and it can also be invoked directly as a function call with minimal modifications.

Augmented Types Extraction: The text to python component returns a list of arguments and return variables for every equation converted into executable Python code. Text2triples extracts augmented types or the semantic meaning of each variable by leveraging the context provided by the text surrounding the equation. We developed a rule-based approach to extract the augmented types for equation variables. The augmented types for variables appear in the text in the form “mass m”, “specific heat gam” etc. We identify such patterns by extracting noun phrases for *+/- k* lines (we set *k* = 2) surrounding the equation and then filtering the ones that reference an equation variable. We further expand the coverage of syntactic patterns for extracting augmented types by leveraging the Text Reading pipeline[14] from University of Arizona’s AutoMATES system. As with scientific concepts, duplicate augmented types are merged together. The text for the extracted augmented types is aligned with the list of extracted scientific concepts.

The **text2ontology API** takes text as input returning a domain ontology as output. The extracted domain ontology consists of top-level classes along with properties and their appropriate

[14] https://github.com/ml4ai/automates/tree/master/src/text_reading

domain and ranges. Wherever possible, this API also extracts scientific units associated with certain properties. This module leverages syntactic patterns to extract classes, properties, domains, ranges and units. It specifically leverages the pattern "*X* of (the) *Y* in *Z*" where part-of-speech tags for *X* and *Y* is a noun. *Y* represents a class and domain of the property *X* and *Z* the units of the property *X*. This is preliminary work and was used to generate domain ontology from comments in the Wind Turbine Design Code is from the Wind-Turbine-Sim project[15].

Knowledge Extraction from Code Pipeline

Extraction of scientific knowledge from code is accomplished by the backend class JavaModelExtractorJP. The JP on the end refers to the github.com Open Source project JavaParser[16]. The JavaParser is used to parse the Java code and create an Abstract Syntax Tree (AST). This tree contains all of the syntactic elements of the code along with their type and the code structure into which they fit. Comments are also identified and associated in as much as is possible with code elements. This AST is then traversed to populate a semantic model of the code, the meta-model for which may be found in an appendix of the M13 Report (see reference at beginning of this report). This semantic model of the code is further analyzed by a reasoner informed by rules in the code metamodel to identify such things as implicit inputs and outputs.

Comments are given special attention. They are associated with code model elements and stored during processing of the AST. When the code model is further analyzed by the ACM, the methods and their inputs and outputs, implicit or explicit, of interest are identified and relevant comments are retrieved. These comments are sent to the backend TextProcessor to identify semantic information in the comments that can be used to identify the semantic meaning of code variables. Currently this extraction is focused on method inputs and outputs, but in the future extraction can be done at a finer level of granularity to find interesting computations and the semantic significance within methods.

Finally, the extracted code model methods and inputs and outputs, with semantic information extracted from comments, is serialized as equations with augmented type information and sent to the DialogAnswerProvider to be displayed to the user as extracted equations. These equations are processed by the JenaBasedDialogModelProcessor and added to the KG and CG for use in computation for query answering.

GE ASKE Performance on Other Sources

As identified in the GE proposals, the GE ASKE System was developed and tested on code and text sources from the NASA Glenn Hypersonics Web Site. To demonstrate how amenable the system is to generalization, we applied the extraction from code capability, supplemented by extraction from the text of code comments, to a different domain, that of wind turbine design. We did extraction both without a pre-existing domain ontology and with a basic pre-existing domain ontology. While the results clearly show the necessity of having human-in-the-loop collaboration and curation, they also demonstrate the usefulness of a domain ontology and the

[15]See https://github.com/Cvarier/Wind-Turbine-Sim

[16] See "Tools for your Java Code," [Online]. Available at https://javaparser.org/.

feasibility of extraction with human assistance as a viable approach to extracting scientific models from high-quality existing code.

The Wind Turbine Design Code is from the Wind-Turbine-Sim project on github.com[17]. The project page describes the project as "Java project simulating the physics of HAWT wind turbines using Blade Element Momentum theory." (HAWT is an acronym for horizontal-axis wind turbine.) The code of greatest interest is located in two files, WindTurbine.java and Coefficients.java. A third file, TurbineSimulation.java, orchestrates the running of a simulation and display of the results, tasks which are done by the GE ASKE System itself, so this code is not needed and does not contain scientific information.

Code Modifications

Minor changes were made to the Java code to alleviate difficulties in translating from Java to Python using the Open Source Java2Python project[18] as well as to avoid elaborate effort to update NumPy-based autograd package for automatic differentiation. Note that we also made several adjustments to the Java2Python code as well over the course of the project[19], but for some deficiencies it was more efficient to modify the input Java code than to try to fix the translator to be more capable, since translation from Java to Python was not a major objective of the research.

Coefficients.java

- Recursive calls are not well supported by autograd package, so recursive calls to getCoefficientLift (lines 40 and 44) were replaced with code blocks of equivalent functionality without using recursive calls.

WindTurbine.java

- Class field variables were given dummy initial values to cause them to be declared in the Python translation, as expected by NumPy.
- Our code converted the constructor to a regular method which is called before using methods from a class. However, references to class variable in the Java constructor were prefixed with "this." and were simplified by dropping the "this." prefix. Since there were no local method variables with the same name, the change did not affect the outcome of a constructor call and allowed the Java to Python translator to produce correct results.
- Due to incorrect and multiple declarations and initializations showing up in Java2Python translation, declaration of local variables in getPowerCoefficient were moved outside of the for loop, which fixed the issue without changing the intended code behavior. The same was done to the getPowerCoefficentNoTip method.

Extraction with No Pre-existing Domain Ontology

Extraction from code was done on both Java files together so that the ontological concepts extracted from code comments in the first, Coefficients.java, would be available to the extraction from code comments in the second, WindTurbine .java. The following were extracted from the Coefficients.java code file.

- 15 top-level classes were extracted from code comments

[17] See https://github.com/Cvarier/Wind-Turbine-Sim.
[18] https://github.com/natural/java2python
[19] The modifications to the java2python code may be found in a fork of that repository at https://github.com/GEGlobalResearch/java2python.

- 6 properties with domain and range were extracted from code comments
- 2 subclasses of UnittedQuantity, including restrictions on allowed unit of measure, were extracted from code comments
- 2 methods were extracted from code, along with their scripts. Note that the class does not have a constructor.
- For each method, augmented type information to tie the arguments and returned values to domain concepts were extracted but all were ambiguous unions of classes and required human curation to identify the correct class.

Extraction from WindTurbine followed with these extractions.

- 17 top-level classes were extracted from code comments
- 7 properties with domain and range were extracted from code comments
- 2 subclasses of UnittedQuantity, including restrictions on allowed unit of measure, were extracted from code comments
- 5 methods were extracted from code, along with their scripts, including the class constructor
- For four of the methods, augmented type information to tie the arguments and returned values to domain concepts were extracted but all were ambiguous unions of classes and required human curation to identify the correct class. No augmented type information was extracted for the fifth method, which had no JavaDoc method information.
- 2 dependencies between methods were extracted from code

Extraction with a Small Pre-existing Domain Ontology

Extraction from Coefficients.java and WindTurbine.java were also performed sequentially with a small pre-existing domain ontology being provided to the text extraction service prior to extraction. This domain ontology is shown in Appendix A. Providing it to the text extraction service allowed better locality search when processing comments in the code.

Extraction with and without an existing domain ontology both yielded ambiguous augmented type information in the form of disjunctions of classes. However, with the exception of getPowerCoefficientNoTip in WindTurbine.java, which had not comments, extraction with the pre-existing domain ontology resulted in the correct augmented type domain class being among those listed in the disjunction. This reduced the human-in-the-loop's task to that of recognizing the correct class and removing the others. More precise comments and/or a better pre-existing domain ontology, along with better filtering of possible semantic information will improve performance in the future and reduce the number of ambiguities in the extracted equations.

GE ASKE Source Code, Installable Components, and Documentation

The source code, installable components, and documentation for the GE ASKE System are found on github.com in two github.com repositories.

1. https://github.com/GEGlobalResearch/DARPA-ASKE-TA1: contains the source code for the GE ASKE System UI Components and Backend in Figure 1, as well as some services components
2. https://github.com/GEGlobalResearch/DARPA-ASKE-TA2: contains the semantic models required for storing queries, composed models, and results; the model execution request service; the service wrapper for the DBN execution engine. Most of the code implementing TA2 function has been merged into the TA1 repo and code.

The location of artifacts relating to each part of the system are described in the sections below.

Source Code and Other Artifacts

The source code for the various capabilities developed during the ASKE projects is found in the following locations.

1. The Dialog ANSWER Eclipse-based extensions to SADL consist of a set of Eclipse Plugin Development projects that are grouped under a parent project with the following project structure.
    a. com.ge.research.sadl.darpa.aske.dialog.parent: parent to all other plug-in projects, contains the pom.xm. file for Maven builds
        i. com.ge.research.sadl.darpa.aske.dialog: Dialog backend containing the Xtext grammar definition (extension to SADL grammar), the model processor, AnswerCuration Manager, inference processor, and all other components shown in the middle (Backend) column of Figure 1
        ii. com.ge.research.sadl.darpa.aske.dependencies: contains third-party dependency JavaParser
        iii. com.ge.research.sadl.darpa.aske.feature: defines the Eclipse feature implemented by these plug-ins
        iv. com.ge.research.sadl.darpa.aske.ide: contains the Xtext module definition and setup and the content proposal provider
        v. com.ge.research.sadl.darpa.aske.tests: backend JUnit test cases
        vi. com.ge.research.sadl.darpa.aske.ui: contains the UI components shown in Figure 1, especially the DialogAnswerProvider, and other Xtext UI components to implement the Dialog editor, syntax coloring, preferences, etc.
        vii. com.ge.research.sadl.darpa.aske.ui.tests: JUnit UI plug-in tests
        viii. com.ge.research.sadl.darpa.aske.update: the Eclipse update project; the ZIP for release is created by a Maven build in the “target” folder.
2. Knowledge Extraction from Text Pipeline sources: Source code can be found in the DARPA-ASKE-TA1 repository under ModelsFromText: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/ModelsFromText
3. JavaToPython sources: The JavaToPython service is a Flask-based web service that wraps around an existing Java to Python code translation package and exposes the package’s underlying translation capabilities via high-level APIs specified using Connexion/OpenAPI. Currently, we support three APIs: `translateExpression`, `translateMethod` and `translateFile` that respectively translate syntactically correct Java-based input expression statements, methods and entire files (containing

class definitions and their contained methods) into some flavor of Python. The service and supporting API definitions are made available as part of the TA1 github repo (https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/java2python).

For the ASKE program, we leveraged an open-source publicly-available Python-based tool called java2python[20] to translate Java-based source code to Python. We made a number of low-level modifications to the java2python package to meet the requirements of the GE ASKE system, chief among which are updates to use it within the same Python v3.7 environment used to support our other services such as Text2triples and K-CHAIN (the out-of-the-box version of java2python only works with Python v2.x). Other changes include altering the translation logic for static methods, variable declarations, print statements, comments and comment blocks. This modified version of the java2python package is made available as part of the following github repo: (https://github.com/GEGlobalResearch/java2python).

4. Model Execution Request service: The Model Execution Request service is designed as a Java Spring Boot-based web service with two REST endpoints – one that can translate the just-in-time assembled model from the Dialog backend into a JSON-ified table form, and another that uses this intermediate tabular representation to automatically generate the input payloads for a model execution (and visualization) framework such as DBN or K-CHAIN. The source code for this service is made available as part of the TA2 github repo (https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/tree/master/ModelExecutionManager/dbnSpecGenerator). When used in conjunction with DBN, this service can be deployed within a Docker container – supporting scripts to do so are available in the TA2 github repo at https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/tree/master/ASKE-Service.
5. K-CHAIN sources: Codes are available here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/kchain
6. InvizIn sources: Codes are available here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/tree/master/Invizin
7. DBN: The DBN execution engine source code is not publicly available, but the executable is available as a Docker container. The source code implementing a service wrapper for the DBN is available from the TA2 github repo (https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/tree/master/DBN). Earlier versions of the ASKE system use DBN. However, as intended, an open source execution engine (K-CHAIN) is now available and the final release of the ASKE system uses K-CHAIN instead of DBN to execute models.

Binaries for Standing Up the GE ASKE System

To stand up a working GE ASKE System, one must stand up the Services in the right-hand side of Figure 1 and install the Eclipse plugins implementing the UI and Backend components. This section documents how to do this.

Services

The following REST services must be running to support a fully functional GE ASKE System.

[20] https://github.com/natural/java2python

1. Knowledge Extraction from Text Pipeline services: Instruction for starting the services under knowledge extraction from text can be found in this file: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/ModelsFromText/README.md
2. JavaToPython: Instructions to install the dependencies for this service and to start it up can be found in this file: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/java2python/README.md
3. K-CHAIN: Instructions for setting up K-CHAIN service is given here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/kchain/setupInstructions.md
4. InVizIn: Instructions for setting up InVizIn (interactive visualization) service is given here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/blob/master/Invizin/setupInstructions.md
5. Model Execution Request service: Instructions to install and bring up this service can be found at: https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/wiki/Model-Execution-Request-Service-for-K-CHAIN
6. Model Execution Request service and/or DBN: Instructions for installing both the DBN and the Model Execution Request service for DBN are available at: https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/wiki/Model-Execution-Request-Service-for-DBN .

Eclipse-based UI and Backend Components

The UI and Backend Components of Figure 1, along with the Knowledge Graph, which is exposed via a Java API, are all implemented as Eclipse plug-ins. Instructions for installing these plug-ins may be found at https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/wiki/Installation-Instructions-for-GE-ASKE-System-Eclipse-Plug-ins. Note that these components are extensions to the Open Source SADL project (see https://github.com/crapo/sadlos2/wiki/Installation-Instructions-for-SADL-IDE,-Version-3.3) and require that a compatible version of SADL be installed in a supported version of Eclipse before their installation.

Documentation

User Documentation

From a user's perspective, most of the interaction with the system happens via the Dialog interface. Thus, most of the user documentation is related to using this interface. This documentation is found here: https://github.com/GEGlobalResearch/DARPA-ASKE-TA1/wiki/Using-the-GE-ASKE-System-Dialog-User-Interface

Other Documentation

Documentation for the various other parts of the GE ASKE System may be found in prior reports, especially the M13 Report.

An example Eclipse project that can be used to experiment with the GE ASKE system is available from https://github.com/GEGlobalResearch/DARPA-ASKE-TA2/blob/master/Sample-Projects/SampleASKEProject.zip . To add as a project an Eclipse instance, download the zip file.

Then create a new Eclipse general project. Then right click the new empty project and select Import. On the import wizard window, select General->Archive File, then Next, then Browse to the downloaded zip file and Open. Answer yes to override project. The project will then have a number of files with extensions .sadl and .dialog. Try opening a .sadl file. If a pop up asks if Xtext nature should be added to the project, answer yes. This will result in some new directories being added to the project. Finally, select the project main directory then under the Eclipse Project menu select Clean. After this it should be possible to open one of the dialog files and try a query.

Key Learnings and Unresolved Challenges from GE ASKE

- **Mixed-initiative interaction allows collaborative knowledge extraction and curation:** While the approach taken in the GE ASKE System's Dialog editor has many limitations and was chosen because it was much less expensive than more capable alternatives, it did demonstrate the power of a mixed-initiative human-computer interface. A key challenge for this kind of interface is the organization of the conversation—chronological or based on conversation threads. Perhaps some combination will prove most effective, or perhaps different users will be most comfortable with different organizations and user-choice is important.
- **Capturing model semantics allows effective model composition:** The case for capturing, in domain terms, the semantics of the inputs and outputs of each model is irrefutable. The GE ASKE System only implemented a portion of the capability inherent in our approach, but the approach proved sound and supported on-demand model composition. While progress was made in extracting model semantics from text, there is much left to be done to improve accuracy and capability. This work will continue.
- **Dialogue v/s visual exploration of models:** Dialogue is suitable for open-ended mixed initiative question-answering and providing human-readable insights. Sensitivity analysis and local behavior of model is difficult to convey with text/tabular information and visualization is more helpful in that scenario.
- **Alignment of similar equations from text and their code implementations is non-trivial:** The code implementation of certain equations might be accomplished over multiple lines, using methods, external APIs, or using approximations, thus aligning the code with similar equations in text needs more improved code understanding than current capability.

Interesting challenges that were not covered or completely addressed in our current effort and would need further DARPA or DoD investments.

- In our design we created the ability to represent knowledge about constraints and model assumptions. However, we did not achieve the ability to automatically extract model assumptions/constraints from text and code. This is an important capability that must be achieved for intelligent model composition by an AI.
- In this effort, the model execution was centralized via a monolithic code script generated to respond to user query. However, more complex problems will require

decentralized model execution based on requirements of computational resources and third-party solvers, where complete code might not be available to be ingested.

- Although some fit-for-purpose notion was addressed in our current work, automated validity analysis of curated knowledge and composed models will require more work.
- Ambiguous information in extracted knowledge needed human intervention to be resolved in our current effort, but capability to handle ambiguous, uncertain, and even conflicting information more autonomously during reasoning and inference will be needed for a more scalable system.
- We created the capability for the AI to realize a gap in knowledge, where it asks for human help to either respond or provide text/code to ingest and fill that gap. However, creating a capability for the AI to search for and obtain responses without human intervention was envisioned but not implemented.

Potential ASKE Applications in Industry and Defense

The aerospace industry produces highly engineered products that meet exacting standards (extreme reliability for aircraft engines) to provide services to the economy. The development of these very complex systems includes setting up a business case that provides an impetus to the end user customer to acquire the next generation of product. In the commercial space this could mean substantial reduction in cost of service provided (think about lower cost per passenger mile flown for commercial aircraft). For military systems, objectives include increased capability of systems, time on target, and speed to get to destination. Once a robust set of requirements is obtained, the industry uses a number of internal programs to develop a 'revision 1' product. This is analyzed and tested in all domains of operations and modified where it does not meet requirements. This is long, iterative, and expensive process.

The ASKE program is the first step in integrating a number of programs that 'design' ie., select dimensions etc., for 'revision 1' conceptual design. To take an example, the modern gas turbine engine has over 10,000 components. There are a number of computer programs that set the dimensions of each stage of the compressor, the combustor, the static structure, the turbines etc. While the thermodynamic cycle deck can provide a good first step in design performance and initial conditions, it is the conceptual design that provides the ability to set the revision 1 design. A fully implemented ASKE program will integrate a large number of engineering design programs enabling designers to semantically explore of the complex design space and test out new design hypothesis without source code changes, which will lead to faster and potentially more creative designs, so that the first-time yield as well as performance is substantially improved. If successful, this can reduce the EMD (Engineering and Manufacturing Development) cost and time of new capability introductions for the US DOD.

Appendix A: Small Wind Turbine Ontology

The Wind Turbine Domain Model

```
/***********************************************************************
* Note: This license has also been called the "New BSD License" or
* "Modified BSD License". See also the 2-clause BSD License.
*
* Copyright Â© 2018-2019 - General Electric Company, All Rights Reserved
*
* Project: KApEESH, developed with the support of the Defense Advanced
* Research Projects Agency (DARPA) under Agreement  No.  HR00111990007.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
* 1. Redistributions of source code must retain the above copyright notice,
*    this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
*    this list of conditions and the following disclaimer in the documentation
*    and/or other materials provided with the distribution.
*
* 3. Neither the name of the copyright holder nor the names of its
*    contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
***********************************************************************/
 uri "http://aske.ge.com/WindTurbine" alias WindTurbine.

import "http://aske.ge.com/sciknow".

WindTurbine is a type of PhysicalThing
      described by radius with values of type Radius
      described by numOfBlades with values of type NumOfBlades
//    described by blade with values of type Blade
      described by tipSpeedRatio with values of type TipSpeedRatio
      described by powerCoefficient with values of type CoefficientOfPower
      described by liftCoefficient with values of type LiftCoefficient
      described by dragCoefficient with values of type DragCoefficient
      described by angleOfAttack with values of type AngleOfAttack
//
//    described by rotationalSpeed with values of type RotationalSpeed
//
      described by radialPosition with values of type RadialPosition
```

```
      described by twist  with values of type Twist
      described by chordLength with values of type ChordLength
      described by radiusIncrement  with values of type RadiusIncrement
      .

Radius is a type of UnittedQuantity.
NumOfBlades (alias "number of blades") is a type of UnittedQuantity.
TipSpeedRatio (alias "lambda") is a type of UnittedQuantity.
CoefficientOfPower (alias "efficiency", "power coefficient") is a type of
UnittedQuantity.

//Blade is a type of PhysicalThing
//     described by bladeSegment with values of type BladeSegment
//     described by rotationalSpeed with values of type RotationalSpeed
//     .

RotationalSpeed is a type of UnittedQuantity.

//BladeSegment (alias "blade element") is a type of PhysicalThing
//     described by radialPosition with values of type RadialPosition
//     described by twist  with values of type Twist
//     described by chordLength with values of type ChordLength
//     described by radiusIncrement  with values of type RadiusIncrement
//     .

RadialPosition is a type of UnittedQuantity.
Twist (alias "local pitch angle") is a type of UnittedQuantity.
ChordLength (alias "chord length") is a type of UnittedQuantity.
RadiusIncrement is a type of UnittedQuantity.

AngleOfAttack is a type of UnittedQuantity.
LiftCoefficient (alias "coefficient of lift") is a type of UnittedQuantity.
DragCoefficient (alias "coefficient of drag") is a type of UnittedQuantity.

//PhysicalThing is a class,
//     described by mass with values of type Mass
//     described by volume with values of type Volume
//     described by density with values of type Density
//     described by temperature with values of type Temperature
//     described by altitude with values of type Altitude
//     described by speed with values of type Speed
//   .
//
//Temperature is a type of UnittedQuantity.
//Pressure is a type of UnittedQuantity.
//
//Speed is a type of UnittedQuantity.
//Mass is a type of UnittedQuantity.
//Volume is a type of UnittedQuantity.
//Density is a type of UnittedQuantity.
//Length is a type of UnittedQuantity.
//Time is a type of UnittedQuantity.
//
//Altitude is a type of UnittedQuantity.
```

The ScienceKnowledge Domain Model (Imported by WindTurbine)

```
/*******************************************************************
* Note: This license has also been called the "New BSD License" or
* "Modified BSD License". See also the 2-clause BSD License.
*
* Copyright Â© 2018-2019 - General Electric Company, All Rights Reserved
*
* Project: KApEESH, developed with the support of the Defense Advanced
* Research Projects Agency (DARPA) under Agreement  No.  HR00111990007.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
* 1. Redistributions of source code must retain the above copyright notice,
*    this list of conditions and the following disclaimer.
*
* 2. Redistributions in binary form must reproduce the above copyright notice,
*    this list of conditions and the following disclaimer in the documentation
*    and/or other materials provided with the distribution.
*
* 3. Neither the name of the copyright holder nor the names of its
*    contributors may be used to endorse or promote products derived
*    from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
* INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
* CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
* THE POSSIBILITY OF SUCH DAMAGE.
*
*******************************************************************/
uri "http://aske.ge.com/sciknow" alias sciknow .

Temperature is a type of UnittedQuantity.
Pressure is a type of UnittedQuantity.

Speed is a type of UnittedQuantity.
Mass is a type of UnittedQuantity.
Volume is a type of UnittedQuantity.
Density is a type of UnittedQuantity.
Length is a type of UnittedQuantity.
Time is a type of UnittedQuantity.

Altitude is a type of UnittedQuantity.

PhysicalThing is a class,
      described by mass with values of type Mass
      described by volume with values of type Volume
      described by density with values of type Density
```

```
        described by temperature with values of type Temperature
        described by altitude with values of type Altitude
        described by speed with values of type Speed
    .

//Gas is a type of PhysicalThing
//     described by pressure with values of type Pressure.
//Air is a type of Gas.
```