INCOSE International Symposium
Philadelphia, PA
June 24-27, 2013

Model as You Write, or Write as You Model – A Grammar-Based Approach to Modeling and Documentation

Takahiro Yamada (JAXA/ISAS)

June 24, 2013

Model-Based Systems Engineering

- According to the INCOSE document "Systems Engineering Vision 2020"
  - "Model-Based Systems Engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities."
- To support this approach, standard languages (such as UML and SysML) that specify methods for developing models have been developed.
- Models developed with UML or SysML have less ambiguity than documents written in a natural language like English. Therefore, models can facilitate sharing of engineering information among engineers.
- This is certainly a great benefit of MBSE.

A Problem with MBSE

- The INCOSE Vision document also says,
  - "MBSE is expected to replace the document-centric approach that has been practiced by systems engineers in the past."
- However, many engineers still write documents (e.g., design specifications) in addition to developing models. Why?
- UML and SysML define rules on how to construct models from elements. The syntax of models is determined by these languages.
- The semantics of models is not always specified by the languages.

  YES "A is a subclass of B"

  YES "X consists of Y and Z"

  NO "A sends message M to B every *n* minutes with protocol X"
- The domain-specific semantics of models is usually specified in documents.

Why Not Use Profiles?

- You can define a profile of UML/SysML with stereotypes such as
  <<send>>, <<message>>, <<protocol>>
  for specifying the domain-specific semantics of models.
- However, there is no standard framework for defining such domain-specific profiles.
- Therefore, there is no guarantee that models using a profile can be integrated with other models using other profiles even in the same problem domain.
- To enable integration of models developed independently, a general framework for specifying the semantics of models is necessary.

Proposed Approach

- This paper proposes an approach for developing documents and models at the same time.
- In this approach, the semantics of any technical information, which may be presented as models or documents, is specified with the same artificial language.
- Semantically, there is no distinction between models and documents.

Grammar-based Approach

- This approach is called GraMod (Grammar-based Modeling and Documentation) because the key concept of this approach is grammar.
- This method can be applied to
  - Any technical information (except for analog information such as drawings, pictures, movies, etc.).
  - Requirement specifications, design specifications, interface specifications, development plans, test plans, test procedures, operations procedures, etc.
- Since the study of this method is still in an early stage, this paper only presents conceptual aspects. Nonetheless, we hope that this paper will stimulate discussion on how to develop models that can meet the true goal of MBSE.

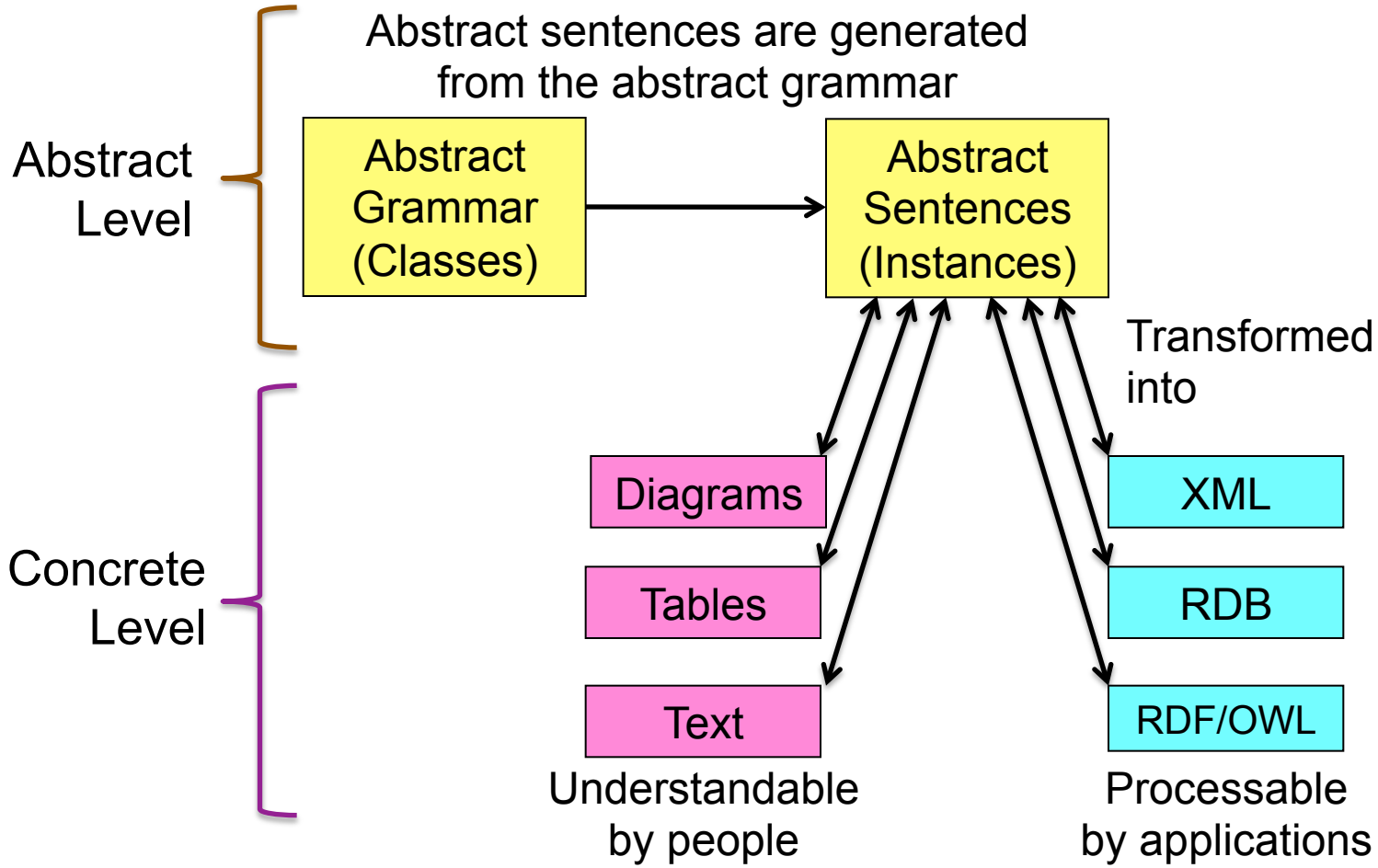
Similar Methods

- Object-Process Methodology (OPM)
  - Developed by Dov Dori.
  - Diagrams called Object-Process Diagrams (OPDs) are automatically translated into sentences of a language called Object-Process Language (OPL).
  - Each sentence of OPL consists of a subject, a verb and an object.
- Object-Role Modeling (ORM)
  - Developed by Terry Halpin.
  - Sentences and diagrams are used to define conceptual schemas that represent the structure of information to be stored in a database.
  - Each sentence consists of a predicate and multiple objects.

Basic Concept 1

- GraMod is a method for generating models and documents at the same time.
- Any technical information is encoded into sentences of an artificial language.
- The information encoded with the artificial language can be transformed into:
  - Visible forms such as diagrams, tables, and text in a natural language.
  - Electronic forms such as XML, RDB, Linked Data (RDF/OWL).
- The electronic data can be used by any application program for validation, simulation and automation of the model described with the data.

Basic Concept 2



Abstract and Concrete Levels

- Abstract sentences at the abstract level and concrete forms at the concrete level are equivalent because the same semantic information is maintained at both levels.
- The semantic information contained in any format is specified by the abstract grammar.
- Abstract sentences can be transformed into concrete forms using mapping rules.
- Concrete forms generated at the concrete level can also be converted to abstract sentences.

Abstract Grammar - Concept

- The abstract grammar specifies a method for describing the semantics of technical information in a formal way.
- The abstract grammar was developed based on the grammar of natural languages.
- In this version of the method, we assume that models are developed to present situations/facts and rules.
- The abstract grammar was designed by selecting grammatical features necessary for presenting situations and rules.
- The abstract grammar specifies how to generate abstract sentences that present particular situations or rules.
- These sentences are abstract in the sense that they only express the semantic structure of the situations or rules.

Dialects

- In order to limit the complexity of the language and to increase the understandability of generated sentences, domain-specific dialects will be defined for different problem domains (for example, information systems, factory automation, spacecraft development, etc.).
- The dialect for a problem domain specifies the set of situations and rules that should be used in that domain.
- The abstract grammar actually consists of a set of domain specific dialects, each of which is optimized for a particular domain.

Abstract Sentence Types

- The abstract grammar specifies abstract sentence types.
- Each abstract sentence type is used for expressing a particular type of situations or rules, and specifies how the semantics of the situations or rules should be expressed.
- From an abstract sentence type, abstract sentences of that type are generated.
- Each abstract sentence generated from an abstract sentence type describes a particular situation or rule.
- Each domain-specific dialect specifies a set of abstract sentence types.

Abstract Sentence Types for Situations

- The situations represented by abstract sentences are classified into situation types.
- For example, situations in which something measures some attribute belong to a situation type called “measure”.
- A set of situation types are selected for each domain-specific dialect.
- A particular situation is encoded with an abstract sentence having a predicate and a set of arguments.
- The predicate represents the type of situation. For example, the verb “measure” is the predicate for the situation type “measure”.
- Predicates are usually represented by verbs or verbal phrases.
- If necessary, the required level of obligation (e.g., “must measure”, “may measure”, etc.) can be specified. In such cases, the modality of the predicate (i.e., “must” or “may”) is added to the predicate.

Arguments and Argument Roles

- The arguments in an abstract sentence represent participants involved in the situation.
- Arguments are usually represented by nouns or noun phrases.
- Arguments are specified based on the roles they play in situations.
- For each situation type, a set of argument roles is specified.
- For example, for the situation type “measure”, there are two argument roles: Agent and Theme.
  - The Agent is the thing that initiates the situation, and, in the situation type “measure”, it is the component that measures the value.
  - The Theme is the thing that is treated in the situation, and, in the situation type “measure”, it is the attribute that is measured.

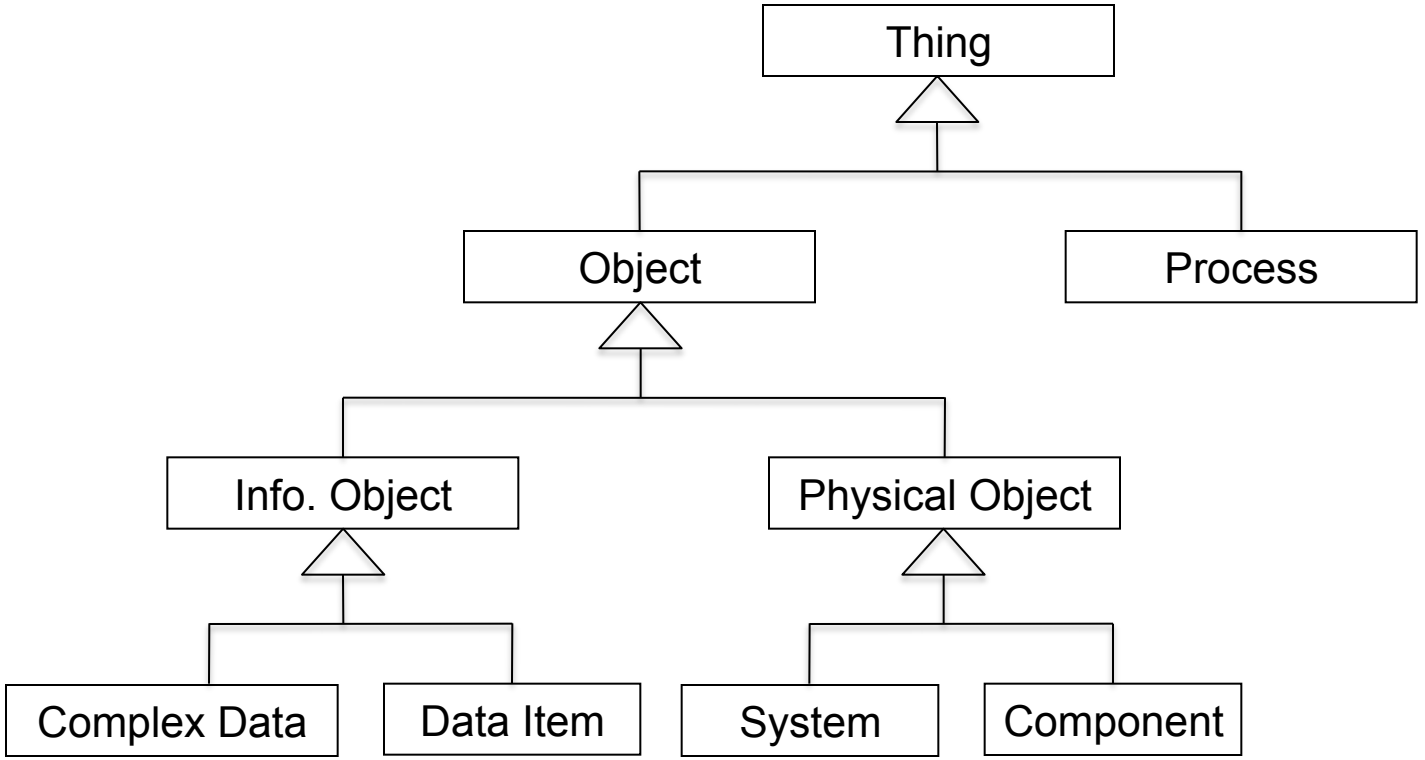
Semantic Roles

- Many (if not all) argument roles of abstract sentences can be determined based on the semantic roles (sometimes also called thematic roles) discussed in the literature of semantics.
- Typical semantic roles discussed in the literature of semantics include:
  - Agent, Patient, Theme, Experiencer, Beneficiary, Instrument, Location, Goal, Source, …
- To define a standard set of argument roles for each dialect (i.e., each problem domain) is a theme for future work.

Argument Classes

- For each argument used in a situation type, the class of things that the argument belong to must be specified.
- For the situation type “measure”,
  - the Agent is a component, which is a class of things.
  - the Theme is an attribute, which is another class of things.
- To specify the class of things that each argument belongs to, we need a way of classifying things.
- A standard method for classification is required in each domain, and a set of standard classes should be used in each domain-specific dialect to specify argument classes.
- The classification of things that should be used in a domain can be defined based on an ontology developed in that domain.

An Example of Classification of Things



Argument Modification and Cardinality

- It is sometimes necessary to modify the noun used as an argument with another word or phrase (e.g., “the temperature of X” and “the temperature measured at 8 o’clock”).
  - The resulting argument is a noun phrase, and it is a theme of future work to determine the structure of allowable noun phrases.
- If it is necessary, the cardinality of an argument can also be specified in the specification of a situation type to indicate how many instances of the argument there should or can exist in individual situations (i.e., in individual abstract sentences).

Situation Type Template

- A situation type is defined with the template shown below (argument cardinalities are omitted here for brevity).

  <predicate (argument role 1: class 1,
  argument role 2: class 2,
  ....
  argument role n: class n) >

- The situation type "measure" can be defined as follows:

  <measure (Agent: component,
  Theme: attribute) >

- Although abstract sentence types and abstract sentences are expressed in English here, they do not depend on any natural language.

Abstract Sentences for Situations 1

- From the situation type “measure”

  <measure (Agent: component,
  Theme: attribute) >

- The following abstract sentence is generated:

  measure (SensorA, temperature)

- The above abstract sentence corresponds to the following English sentence.

  SensorA measures temperature.

- Or, to the following Japanese sentence.

  SensorA wa ondo wo hakaru.

  (SensorA-sbj temperature-obj measure)

Abstract Sentences for Situations 2

- From the situation type “send”

  <send (Agent: component,
  Theme: information object,
  Recipient: component) >

- The following abstract sentence is generated:

  send (SensorA, temperatureData, ProcessorA)

- The above abstract sentence corresponds to the following English sentence.

  SensorA sends temperature data to ProcessorA .

- Or, to the following Japanese sentence.

  SensorA wa ondo-deta wo ProcessorA ni okuru.
  (SensorA-sbj temperature-obj ProcessorA-rcp measure)

Abstract Sentence Types for Rules

- A rule is a description of an action to be performed or a fact that should exist when a condition is met.
- A rule is encoded with an abstract sentence composed of two clauses (a main clause and a conditional clause) concatenated with a conjunction such as “if”, “when”, “while”, “until”, or “after”.
- The rules represented by abstract sentences are classified into rule types, each of which is represented with the conjunction used in the sentence.
- The structure of the main clause is the same as that of abstract sentences for situations.
- The conditional clause is expressed by a group of abstract sentences for situations concatenated with “and”, “or”, and “not”.

Rule Type Template

- A rule type is defined with the template shown below.

  <conjunction (conditional clause),
  main clause) >

- The rule type “if” can be defined as follows:

  <if (conditional clause),
  main clause) >

Abstract Sentences for Rules

- From the rule type “if”

  <if (conditional clause),
  main clause) >

- The following abstract sentence is generated:

  if (is-more-than (temperatureA, 40 degrees)),
  (must turn-off (ProcessorA, HeaterA))

- The above abstract sentence corresponds to the following English sentence.

  If temperatureA is more than 40 degrees,
  ProcessorA must turn off HeaterA.

Concrete Level

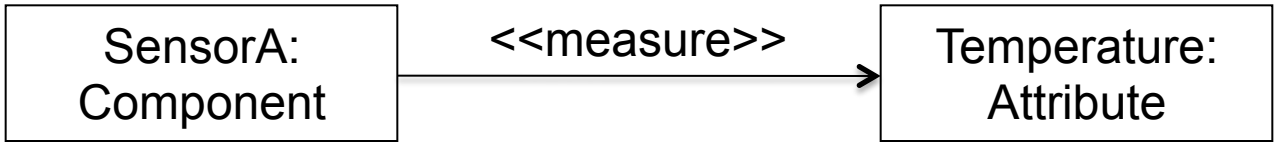
- The concrete level specifies how to present abstract sentences in well-established formats such as natural languages, diagrams, tables, XML, Linked Data, and so on, which are understandable by people or processable by applications.
- To enable this, rules for transforming abstract sentences into concrete formats should be defined (definition of such rules is a theme for future study).
- The same rules can also be applied in the reverse direction to transform concrete formats into abstract sentences.
- An example of concrete format is natural languages, and some examples of transforming abstract sentences into English and Japanese sentences have already been shown.

Transformation into Diagrams

- The following abstract sentence

  measure (SensorA, temperature)

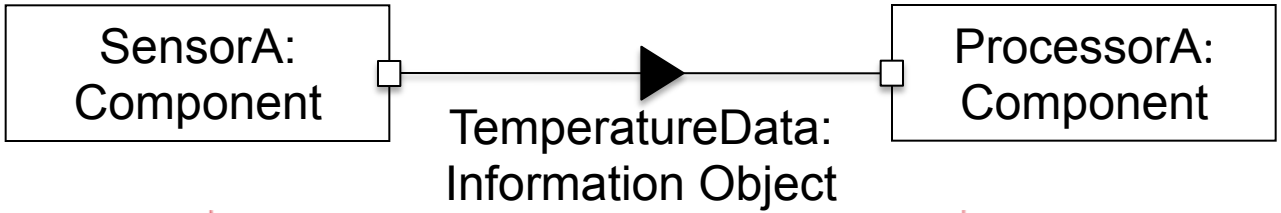
  can be transformed into the following UML diagram.



- The following abstract sentence

  send (SensorA, temperatureData, ProcessorA)

  can be transformed into the following SysML diagram.



Transformation into Tables

- The following abstract sentences

  measure (SensorA, temperature)
  send (SensorA, temperatureData, ProcessorA)

  can be transformed into the following table.

Agent	Predicate	Theme	Recipient
SensorA	measure	temperature	
	send	temperatureData	ProcessorA

Benefits of This Approach

1. Semantic information can be embedded in models, and models can also be used as documents.
2. Models and documents can be generated easily. They can be generated with the most appropriate format for the problem domain (e.g., diagrams, tables, text, etc.).
3. Models and documents can be shown in various ways (i.e., can be transformed into various formats).
4. Generated models and documents can be processed by application programs electronically.
5. Knowledge-base can be generated with this approach.

Future Work

1. Selection of a standard set of situation types and rule types for each problem domain.
2. Definition of standard argument roles.
3. Definition of classes of things to be used as argument classes.
4. Definition of the structure of allowable noun phases.
5. Definition of mapping rules between abstract sentences and concrete formats.
6. Extension of the abstract grammar to cover more complex sentence structures.
7. Comparison of expressive power between this method and other methods including OPM, ORM, UML, SysML, RDB, RDF/OWL.

Conclusion

- This paper has presented a novel approach for generating models and documents at the same time, which is called Grammar-based Modeling and Documentation (GraMod).
- This paper only presented conceptual aspects. To validate our claim, this method must be applied to real problems.
- Our next steps are:
  - to define a dialect for spacecraft testing, and to apply it for specifying spacecraft test procedures.
  - to define a dialect for spacecraft data systems, and to apply it for specifying requirements and designs of spacecraft data systems.
- Although the study of this method is still in a very early stage, we hope that this paper will stimulate discussion on developing models that can be used for real problems.