Ontology Based Req. Analysis: Lightweight Semantic Processing Approach

Haruhiko Kaiya
Shinshu University, Japan
Motoshi Saeki
Tokyo Institute of Technology, Japan

Contents

- Background and Purpose
- Semantic Issues in Req. Analysis
- Ontology in THIS Research
- Detecting Problems in Requirements
- Evaluating Req. by Metrics
- Predicting Req. Changes
- Conclusions and Future Works

Background

- Support Requirements Analysis Systematically and Automatically
- Semantic Processing is required for such Support
- Formal or Semi-formal notations for requirements are costly
  - e.g., cost for training software engineers

Research Goal

- Lightweight Semantic Processing in Requirements Analysis
  - Lightweight ⇨ without rigorous Natural or Formal Language Processing
  - Semantic ⇨ focusing on meaning of each requirements statement.
  - Processing ⇨ using Inference Tools like Prolog

Important Semantic Processing in RE

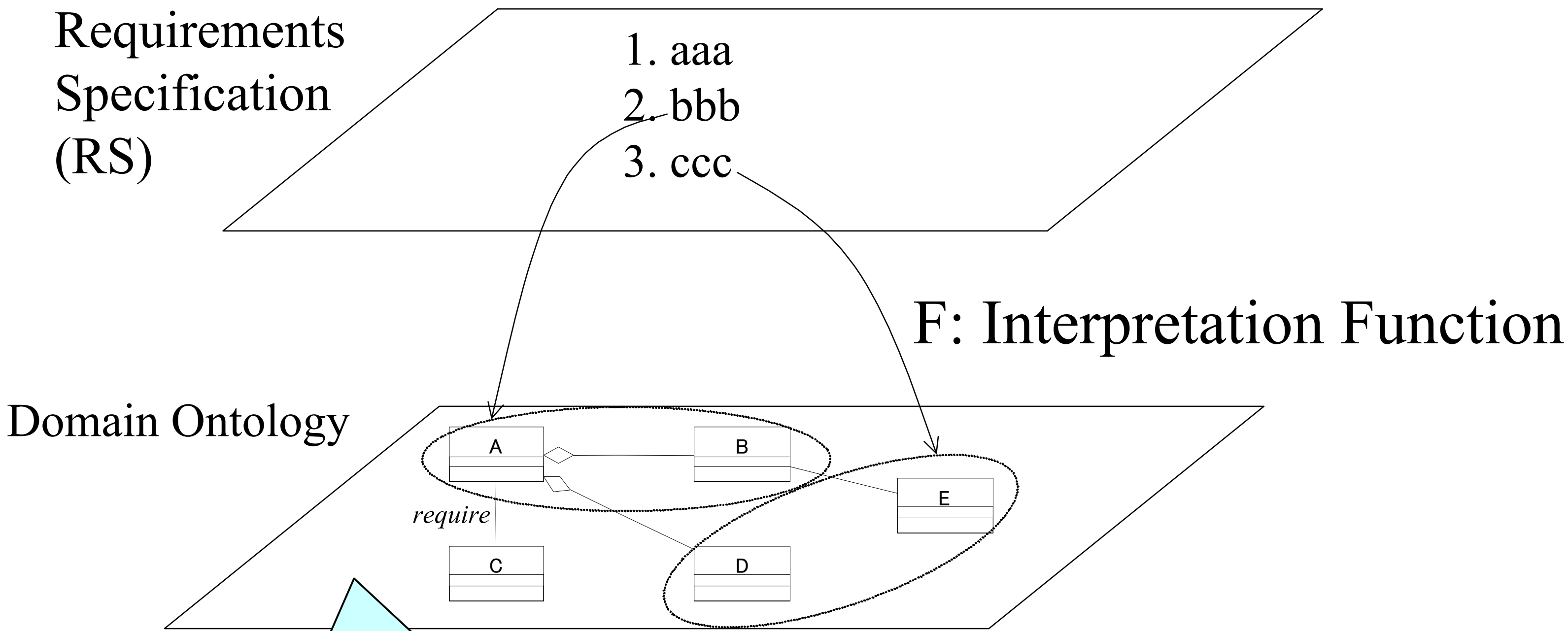
- Consistency
  - Mutual inconsistent requirements confuse issues in design and implementation.
- Completeness
  - Missing requirements cause additional cost.
- Correctness
  - Customers never accept a product based on incorrect requirements.
- Unambiguousness
  - Ambiguousness also confuses issues in design and implementation.

Ontology in THIS Research

- Ontology ⇨ Domain Ontology
  - Ontology for a specific area or a field

    e.g., Chemical Plant, Web Commerce....
- Ontology = Thesaurus + Inference Rules
- Usages of such Ontology
  - Interpreting sentences in Requirements
    - Relate each sentence with concepts in thesaurus.
  - Confirming properties of Requirements
    - Inferring propositions on the thesaurus.

Relationship: Req’s and Ontology



Symbolic Calculation (Inferences) instead of Direct Processing in RS

Requirements Specification (RS)

- A Document written in Natural Language, e.g., English.
- Especially, a List of Requirements Statements.
- Currently, we do not handle Figures and Tables.
- In the Future, we want to use standard formats recommended in IEEE830 std.

Example of RS in This Research

1. Play a music, pause. Go to next or previous music.
2. Forward and rewind.
3. Adjust volume and mute.
4. Repeat play list.
5. Random play list.

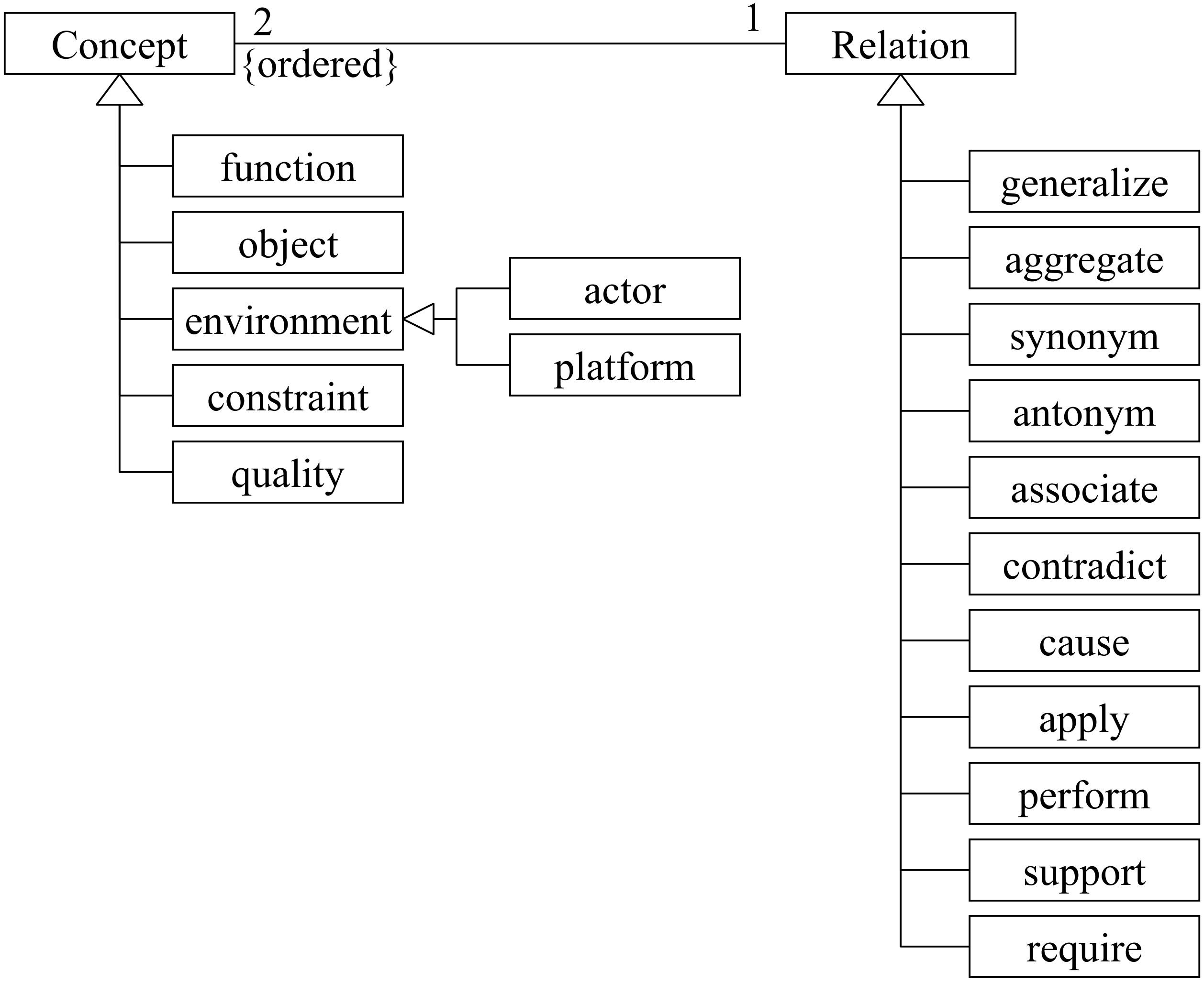
   ......

RS for software music player like Windows Media Player

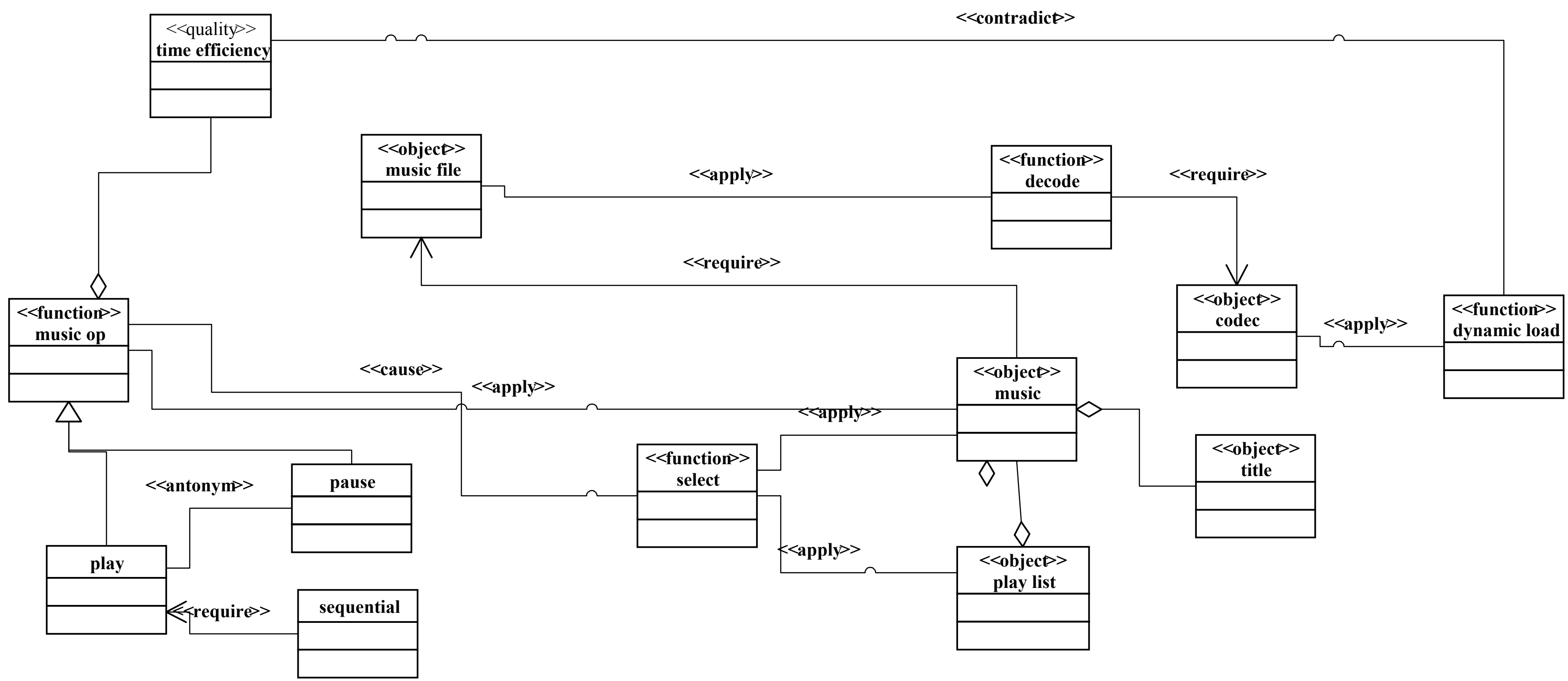
Thesaurus in THIS research

- Simple Directed Graph
  - Node = Concept = Word
  - Arc = Relations between two concepts
- Nodes and Arcs are typed.
  - Such types are used to infer properties in RS.
- How to create such thesaurus...
  - Out of scope of this paper, but...
  - Results in last presentation could be used for.

Types in our Thesaurus



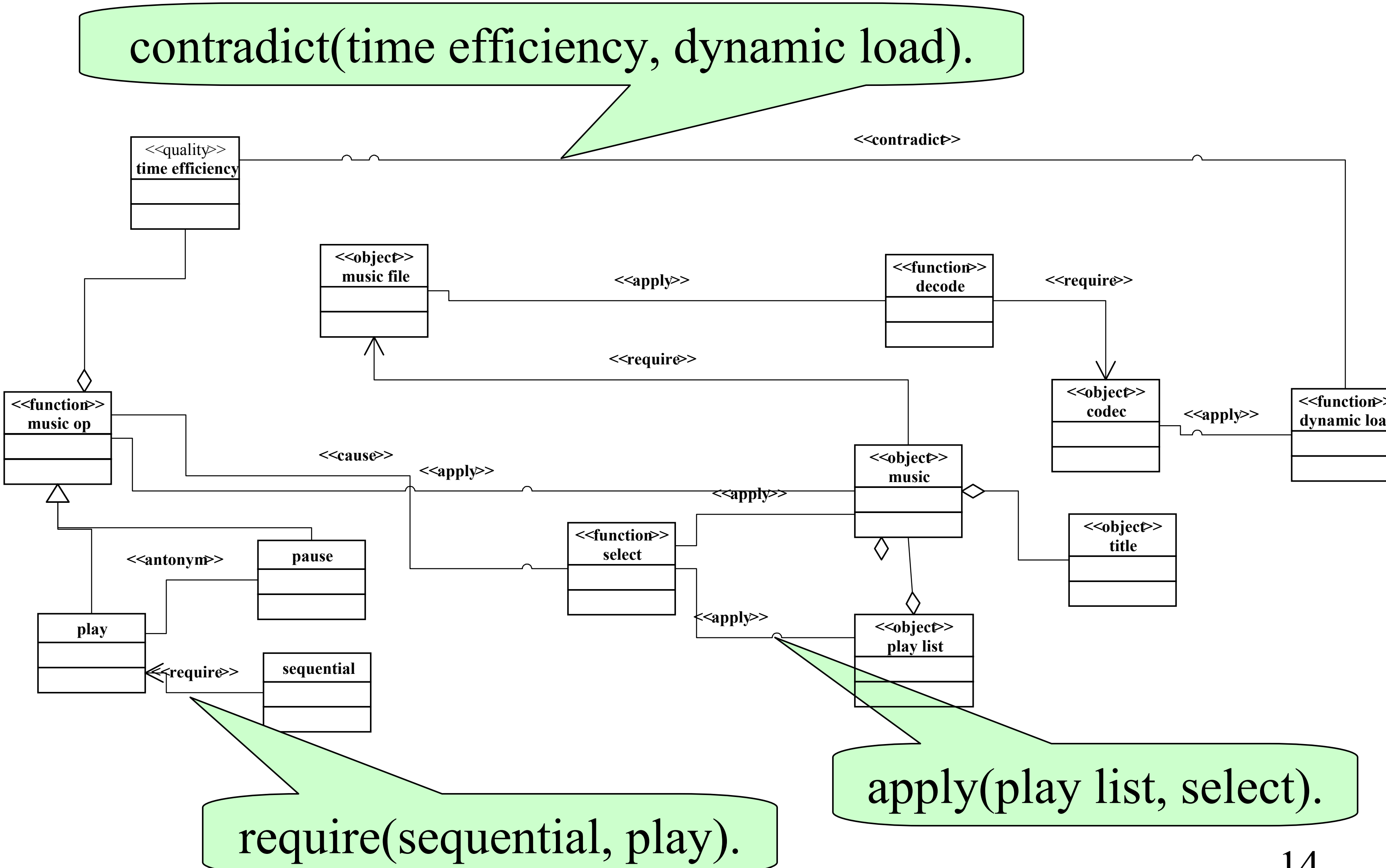
Example of Thesaurus (partially)



Inference Mechanism

- Simple First Order Logic
  - Predicate ⇨ types in thesaurus
  - Variables and constants ⇨ instances of concepts and relationships.
  - Using existing tools and/or languages, e.g., prolog.
- Logical Formulas
  - Logical facts corresponding to a part of thesaurus
  - Rules based on the types.
  - Formulas corresponding to a property to be proved.

Facts from parts of Thesaurus



Rules for Types

- Rule for Generalization type

  forall x gen(x, x)

  - Reflective rule

  gen(x, y) & gen(y,z) → gen(x,z)

  - Transitive rule

- Rule for Antonym type

  antonym(x, y) → antonym(y,x)

  - symmetrical rule

- Mixed

  gen(a,d) and require(a,c) → require(d,c)

Steps to prove RS properties

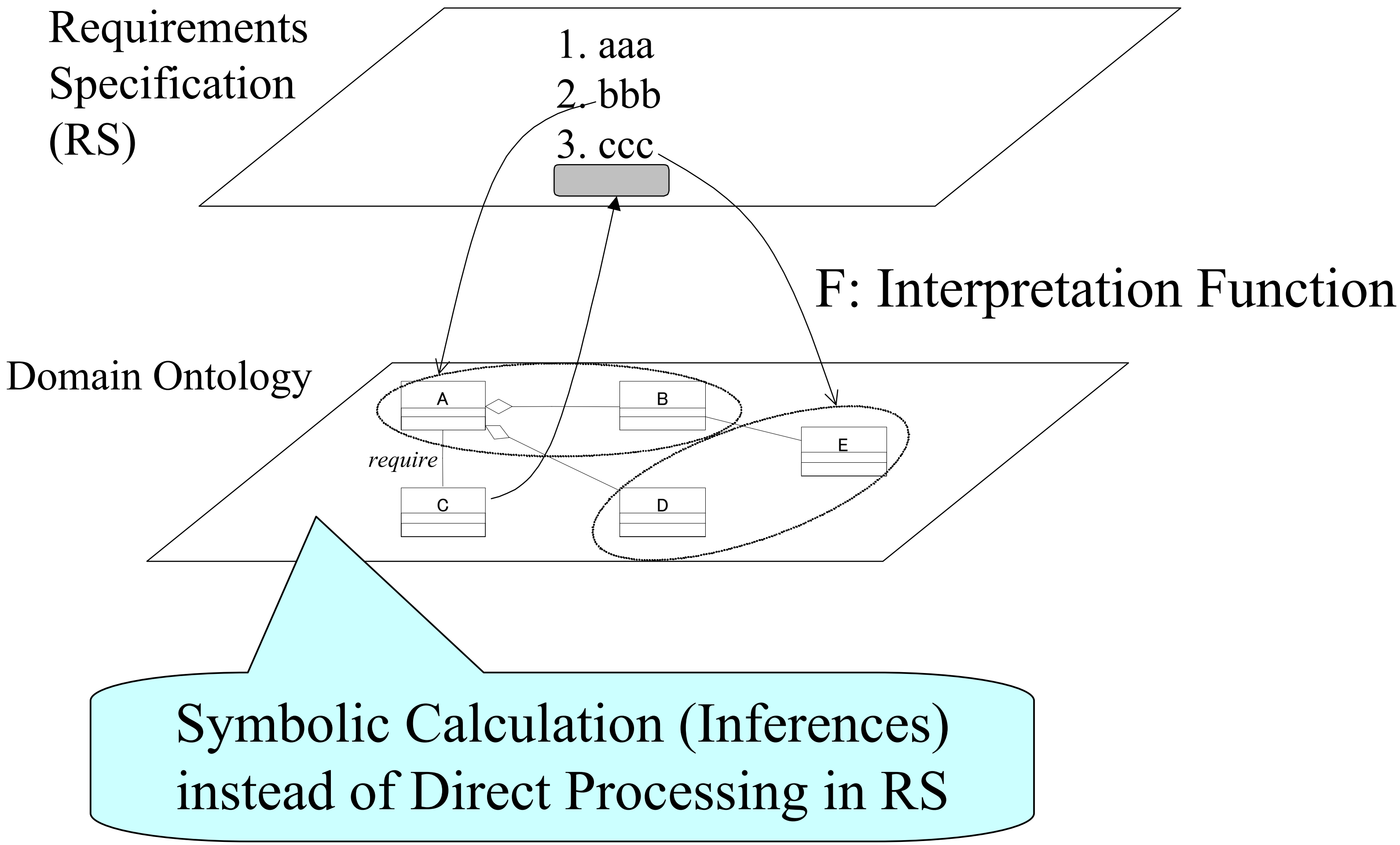
1. Prepare an Ontology (Thesaurus + Rules) for the problem domain.
2. For each statement in RS,

   map the statement to concepts and relations on the Thesaurus.
3. Identify logical facts from the mapped concepts and relations.
4. On the mapped concepts and relations,

   prove specific formulas to detect properties the statement

   using rules and facts.

Relationship: Req's and Ontology



Properties of RS to be detected

- Completeness of Requirements Spec.
  - All significant requirements are described.
- Inconsistency in Requirements Spec.
  - No requirements conflict with each other.

Completeness 1

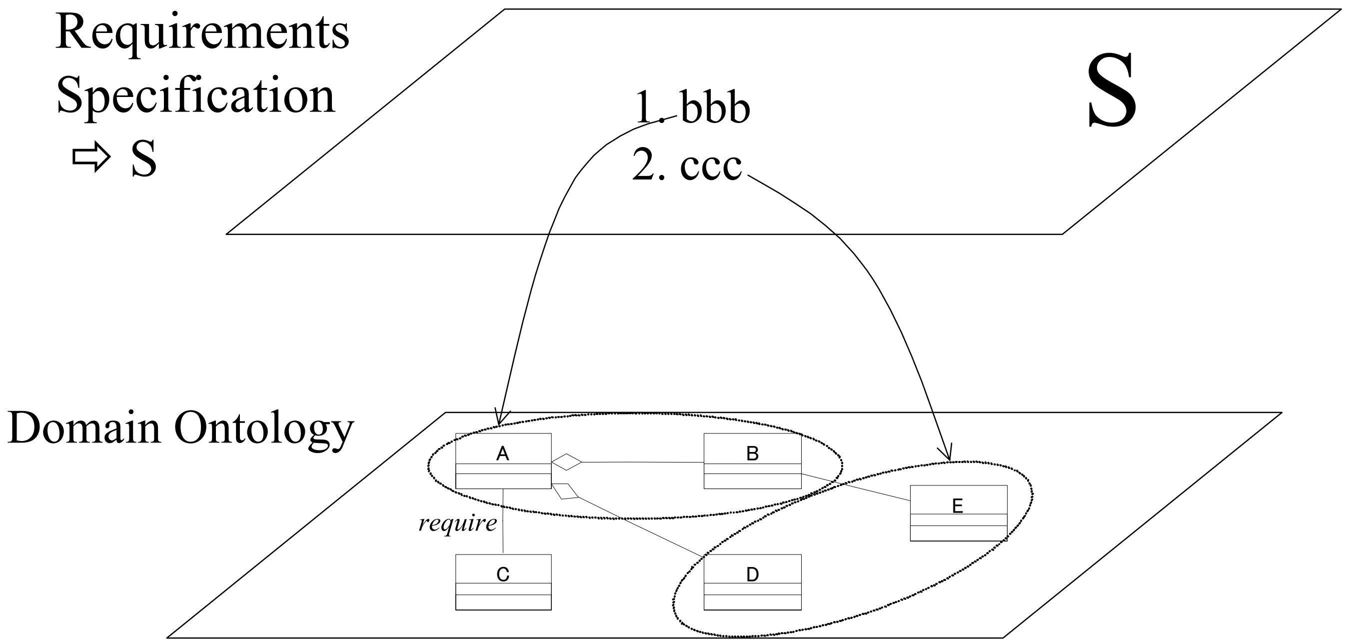
- When an object is mentioned in a RS and there are functions that can be applied to the object, such functions should be examined.

  forall s, x, exists y (object(x) & inSpec(x, s))

  →(function(y) & apply(y, x) & inSpec(y,s)))

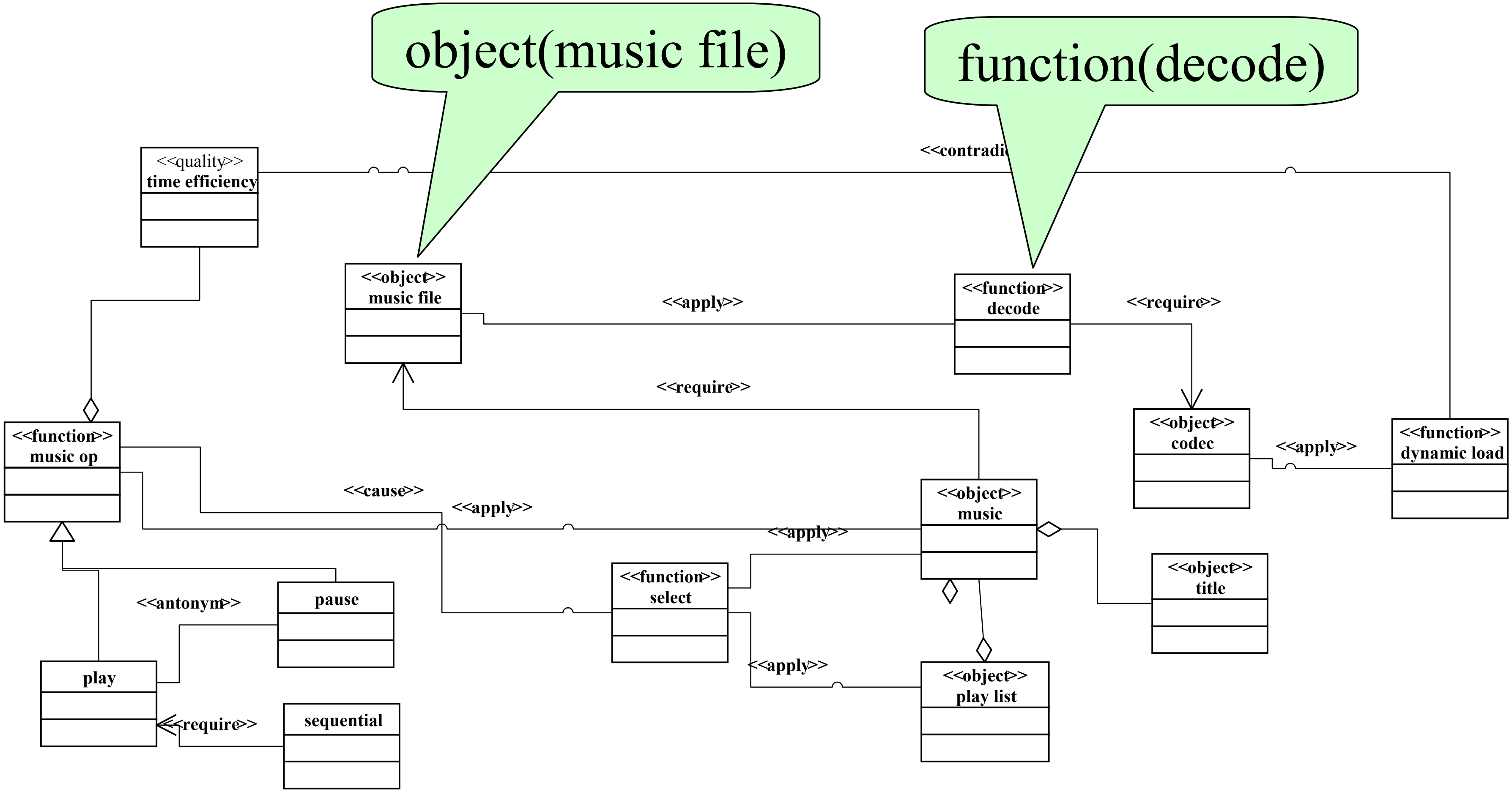
- Prove the formula above and find corresponding instances of function(y).

inSpec(x,s)



inSpec(A,S), inSpec(B,S), inSpec(E,S), inSpec(D,S) are all true.

object(x), function(y)...



Completeness 2

- When a concept in a RS and the concept requires other concepts, such concepts should be examined.

  forall s, x, y (InSpec(x,s) & require(x,y)

  → InSpec(y,s))

- Prove the formula above and find corresponding instances of y.

Consistency

- Find relationships typed by `inconsistent' in relationships corresponding to a RS.

  forall s, x (InSpec(x,s) → exists y (InSpec(y,s) & contradict(x,y))
- If the formula above is true, the RS can be inconsistent.
- We explicitly have inconsistent relationships in our domain ontology.

Semantic Metrics for RS

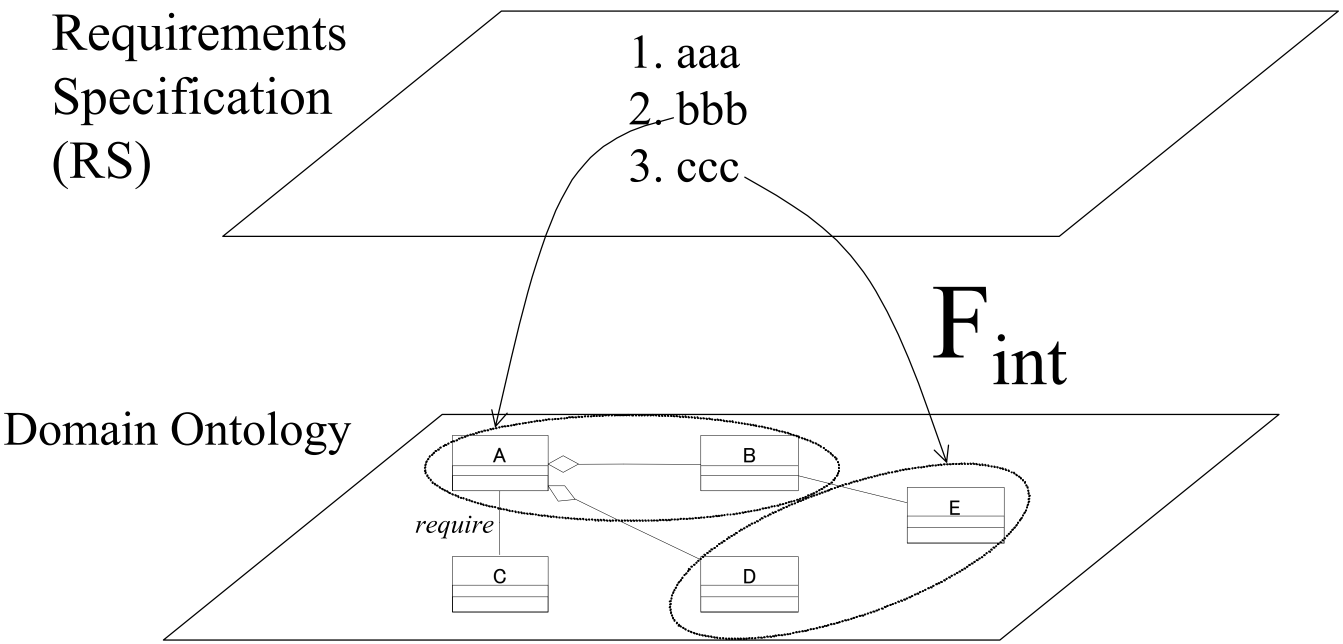
- Indexes to know how far a RS is good.
- Good: correctness, completeness, consistency, unambiguity.
  - mentioned in IEEE 830 standard.
- To measure (count) the numbers of
  - concepts, relationships with certain types in ontology.
  - requirements statements.

Correctness Metric

- Because a domain ontology can be a guideline to decide req's naturally required in the domain, all statements in RS should correspond to elements in the ontology.
- Formal definition

$$\text{Correctness} = \frac{|\{x \mid x \in \text{ReqItem} \wedge F_{\text{int}}(x) \neq \phi\}|}{|\text{ReqItem}|}$$

Example



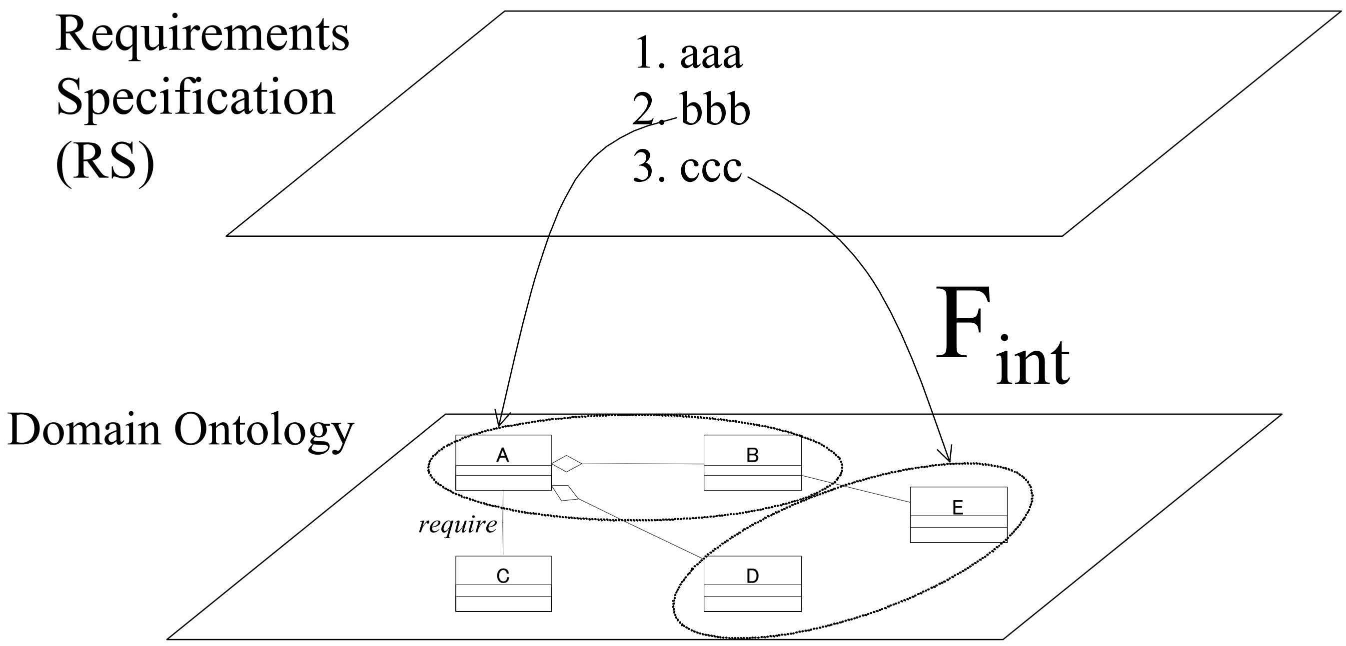
Correctness = 2/3 = 66 %

Completeness Metric

- All in ontology should be mentioned in RS ideally.
- Formal definition:

$$\text{Completeness} = \frac{\left|\begin{array}{l}\{x \mid x \in \text{Con} \cup \text{Rel} \wedge \\ \quad \exists y : \text{ReqItem} \cdot x \in F_{\text{int}}(y)\}\end{array}\right|}{|Con \cup Rel|}$$

Example



Completeness = (4+1) / (5+4) = 55 %

Consistency Metric

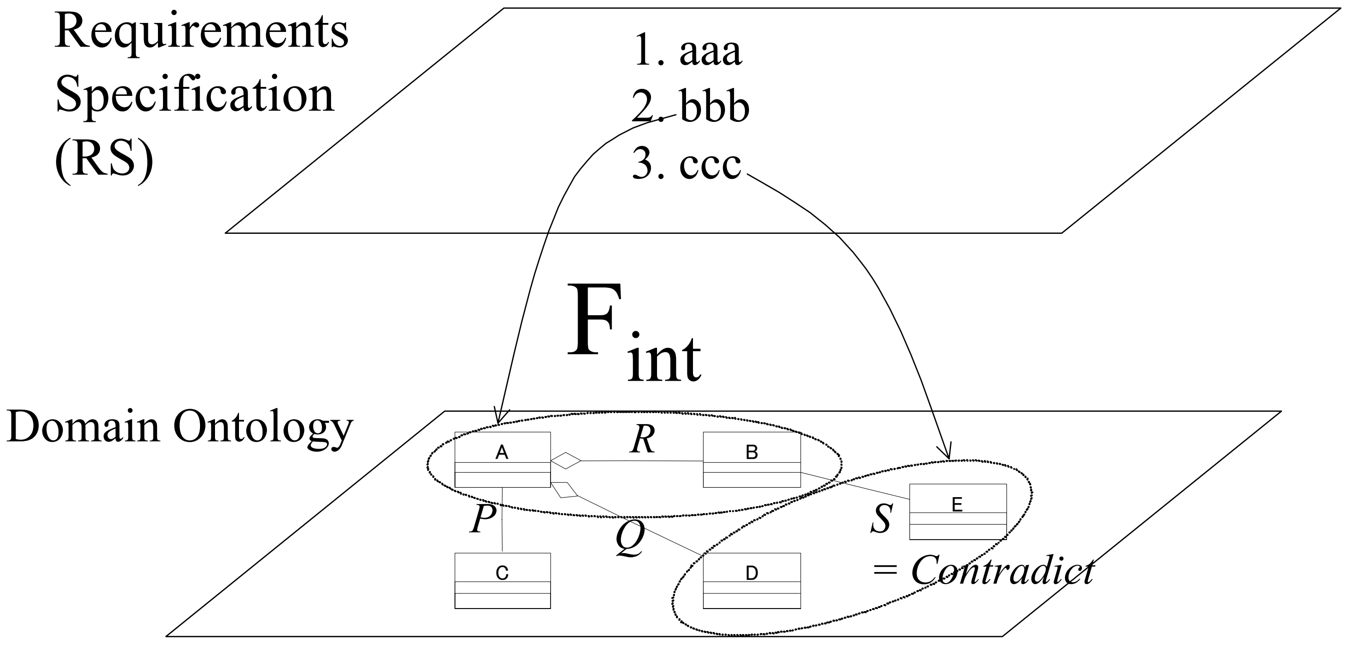
- The relative number of `contradict relationships' out of all relationships can indicate the degree of inconsistency.
- Formal Definition:

$$\text{Consistency} = \frac{|\{\mathrm{x} \mid \mathrm{x} \in \mathrm{RCC} \wedge \neg \mathrm{contradict}(\mathrm{x})\}|}{|\mathrm{RCC}|}$$

- Where RCC is a set of relationships that can connect concepts corresponding to RS.

$$\mathrm{RCC} = \begin{array}{l}\{\mathrm{r} \mid \exists \mathrm{a} \exists \mathrm{b} : \mathrm{Con} \cdot \exists \mathrm{x} \exists \mathrm{y} : \mathrm{ReqItem} \cdot \\ \quad a \in F_{int}(x) \wedge b \in F_{int}(y) \wedge r(a, b)\}\end{array}$$

Example



RCC = { R, Q, S}
If type of S is `contradict',
Consistency = 2/3 = 66 %

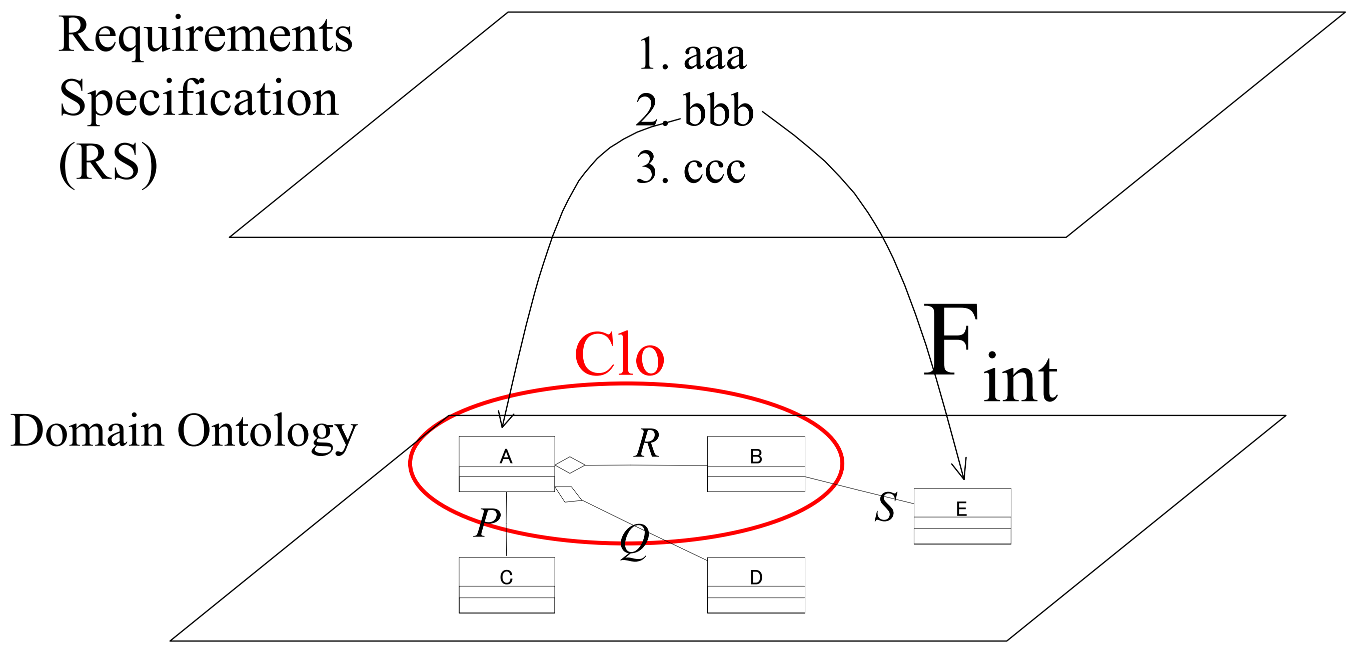
Unambiguity Metric

- When a req. statement is mapped onto several elements that are not semantically related, the statement is regarded as an ambiguous one.
- Formal Definition:

$$\text{Unambiguity} = \frac{|\{x \mid x \in \text{ReqItem} \wedge F_{\text{int}}(x) \subseteq \text{Clo}\}|}{|\text{ReqItem}|}$$

- `Clo' is a transitive closure of relationships except contradict and antonym.

Example



If statement ‘bbb’ is mapped to A and E,
but closure Clo including A cannot include E,
we regard ‘bbb’ is ambiguous.

Small Case Study

- Goal: observe the values of metrics and compare the values with our intuition.
- Ontology for software music player.
  - developed by using the manuals of existing systems.
- A requirements specification (Fig. 6)
  - written by one of authors.
  - Several contradictions were embedded intentionally.

Results

- Correctness = 87 %
  - This result meets our intuition.
  - Excluded items in RS seemed to unfit for this kind of applications.
- Completeness = 44 %
  - Too small!
  - The metric should be reconsidered.
- Consistency = 97 %
  - Embedded errors could be handled, but too large!
- Unambiguity = 87 %
  - Actually, detect statements seems to be ambiguous.

Predict Requirements Changes

- Hypothesis:
  - In a specific application domain, similar kinds of requirements changes may frequently occur.
  - Such changes will help requirements analysts to analyze/elicit requirements.
  - Such changes can be represented by using types in our Ontology.
- Advantages:
  - Identify stable/unstable parts in RS in advance.
  - Pay attention to expected requirements in advance.

Small Case Study

- Goal: Explore such patterns, and identify their advantages.
- Resources: changes logs of several software products.
- Results:
  - Several patterns of changes could be identified.
  - Such patterns can be represented by using types in ontology.
    - e.g., After a function is added, its quality is often improved.

Conclusions

- Propose a method to analyze/elicit requirements by using domain ontology.
  - Mapping requirements statements onto elements in ontology.
  - Symbolic calculation and simple proof on such elements instead of NLP for the statements.
- Show small case studies.

Future Works

- Propose a method to create domain ontology.
  - We are going to progress another project for this issue.
- Procedural support to requirements analysis.
  - And we will provide supporting tool.
- Compatibility with standard notation of ontology.
  - RDF/OWL

Thank you for your attention.