Policy Based Control of Cognitive Radios: Methods and Tools

Mitch Kokar

mkokar@ece.neu.edu

www.ece.neu.edu/groups/scs/kokar

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Outline

- What is a Cognitive Radio (CR)?
- A scenario
- Semantic Web, ontology, ontologies
- Semantics, Semantic Webs
- Cognition and Cognitive Radio
- Classification of languages – procedural vs. declarative
- Interoperability scenario/problem
- Solving the interoperability with ontologies/OWL
- Adding Rules – representation of structure
- Adding functions – representation of functionality
- Behaviors
- Computational complexity
- Summaries
- Language standardization efforts

Proceedings of the SDR '08 Technical Conference and Product Exposition.

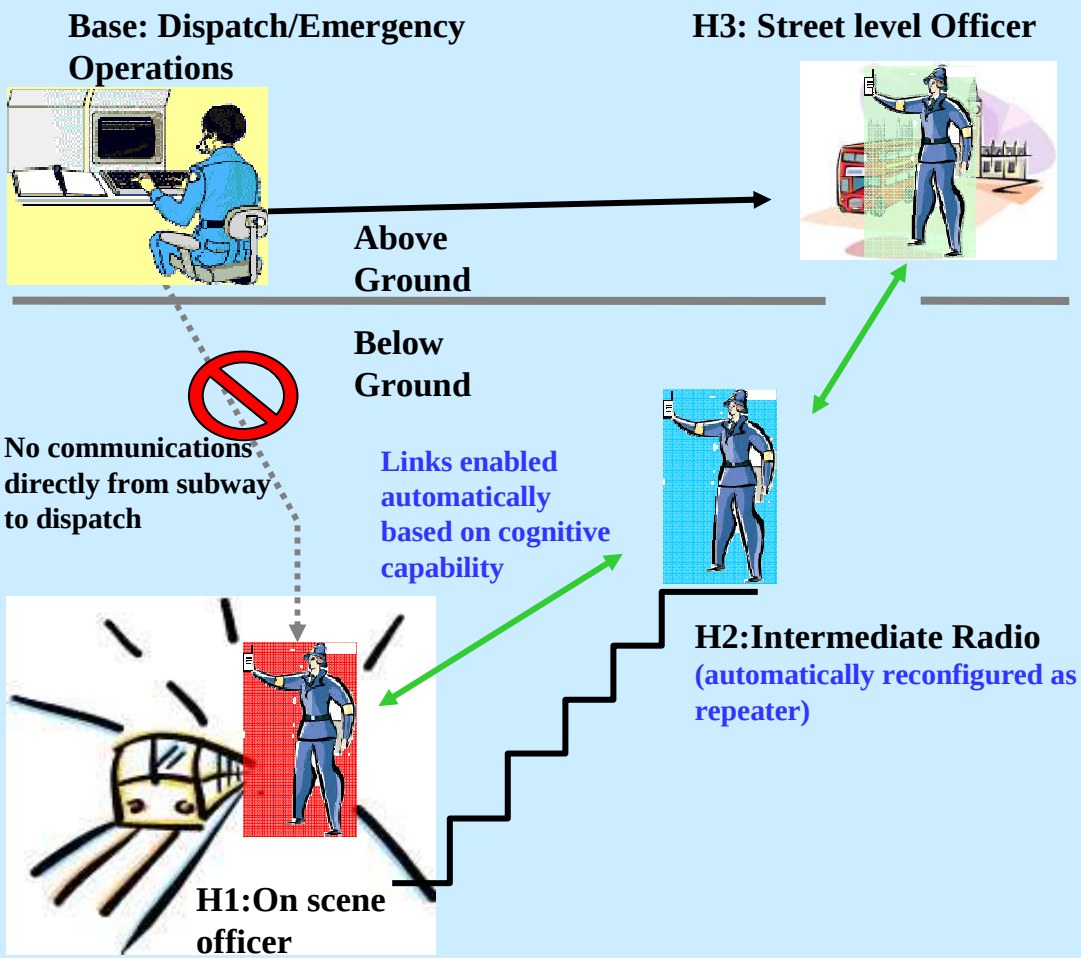
Cognitive Radio Requirements

- Be aware of its own state and the state of the environment
- Tell other radios and network of what it knows and what it wants
- Reflect, i.e., be able to draw conclusions from the facts that it is aware of
- React to surprise, i.e., react to the circumstances it has not seen before.

Note: there are many other requirements …

Proceedings of the SDR '08 Technical Conference and Product Exposition.

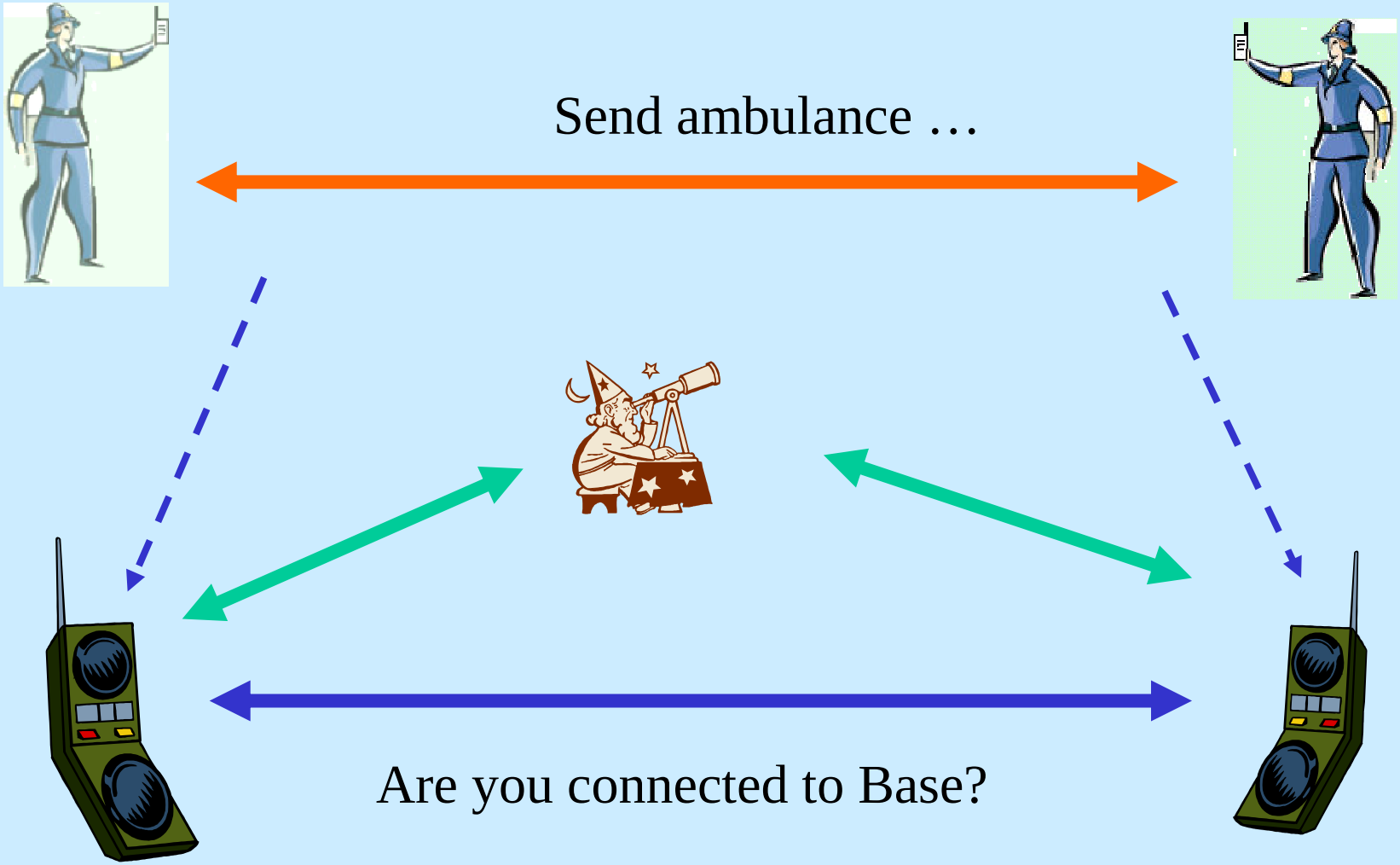
A Scenario for CR Application



Source: Use Cases for MLM Language in Modern Wireless Networks. MLM Work Group, SDRF.

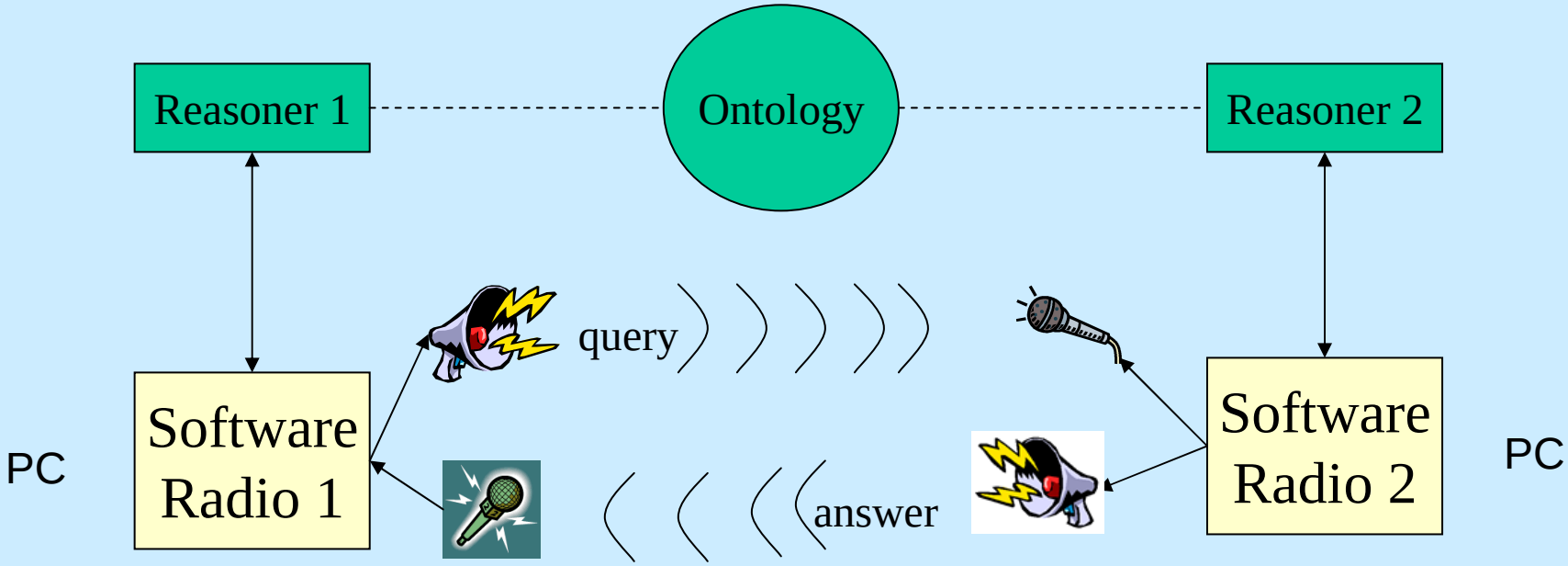
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Two Conversations



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Another Example



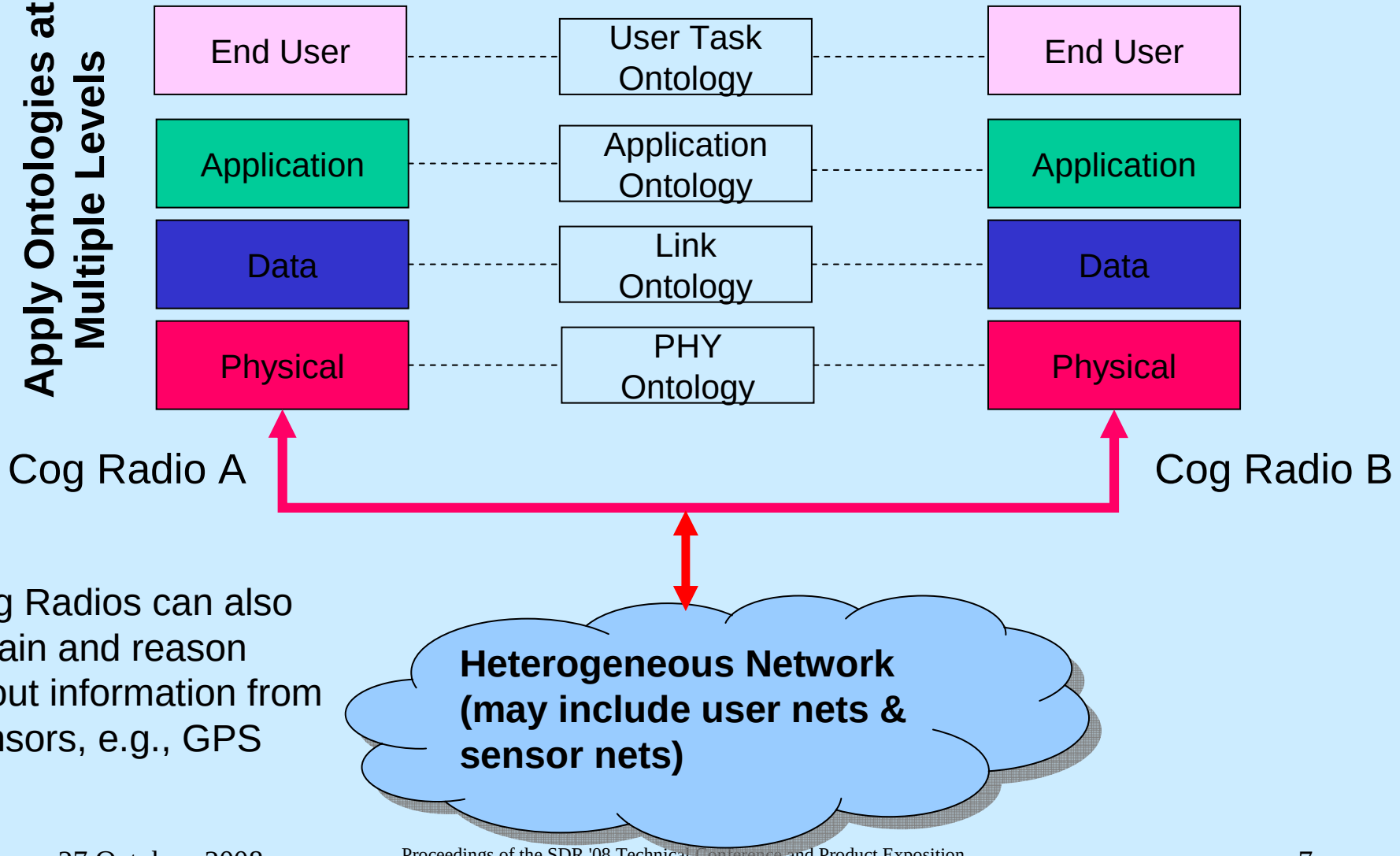
query: What is the rmsDelay, excessDelay and equalizerError in your last Buffer?

answer: rmsDelay = 1.007837037250556; excessDelay = 1.062759005498691; equalizerError = 851.5498431539809

Depending on these values, the transmitter can: select the size of the alphabet, e.g., `QAM2` or `QAM4` or `QAM16`, increase equalizer length, increase power …

Proceedings of the SDR '08 Technical Conference and Product Exposition.

A Framework for Use of Ontologies in Cognitive Radios



Cog Radios can also obtain and reason about information from sensors, e.g., GPS

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Solution Ingredients

- If the “connected to Base” question was the only question, then we could hard-code it into the process.
- But then the capabilities would be limited to the encoded question!
- How can we add more flexibility for questions/answers?

Ingredients exist (Semantic Web Technology):

- Ontologies to define a description-and-query language
- Ontology description language (OWL) for computer processing
- SPARQL – Query Language
- Formal reasoners (theorem provers) to derive answers to queries, e.g., Racer, Pellet, BaseVISor, Jena reasoners, KAON2, Fact++, Prolog, …

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Semantic Web

- Semantics + Web
- Semantics: meaning plus inference
- Web
  - WWW
  - One web page
  - Distributed databases
  - One database with simplified schema (triples)
  - Any source of information with semantics

Proceedings of the SDR '08 Technical Conference and Product Exposition.

What is Ontology (in philosophy)?

1. A science or study of being: specifically, a branch of metaphysics relating to the nature and relations of being; a particular system according to which problems of the nature of being are investigated; first philosophy.

2. A theory concerning the kinds of entities and specifically the kinds of abstract entities that are to be admitted to a language system.

   - *Webster's Third New International Dictionary*

   - *www.formalontology.it/*

- Study of fundamental categories of *object, state of affairs, part, whole*, the relations between parts and the whole and their laws of dependence
- Study of logical features of predication and of the various theories of universals

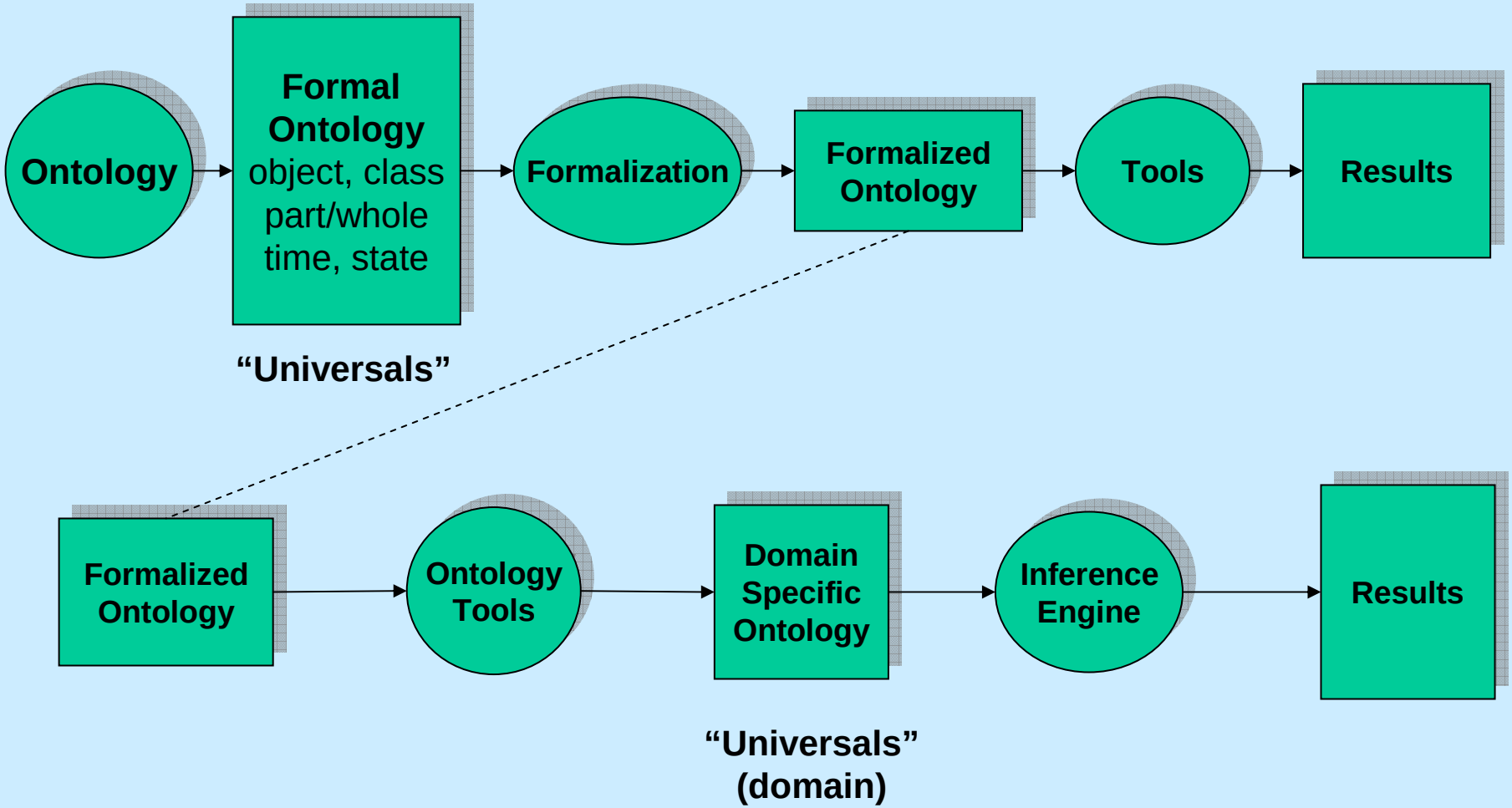
Proceedings of the SDR '08 Technical Conference and Product Exposition.

What is Ontology (in AI)?

- An ontology is an explicit specification of a conceptualization. (Tom Gruber)
- What exists is that which can be represented.
- Knowledge of a domain represented in declarative formalism, objects of a domain, universe of discourse.
- Definitions that associate the names of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms.
- The statement of a logical theory.
- Note: Microsoft recognizes only the philosophy version of Ontology!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Philosophy + AI + Engineering



Semantics

- Meaning, but what does it mean to have a “meaning”?
- Answer: It’s mainly in classifications and relations
- Example (next page)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

20 Questions

- Is it a thing? Yes.
- Is it an institution? Yes.
- Is it government owned? No.
- Is it a school? Yes.
- Is it a university? Yes.
- Is it located in Boston? Yes.
- Is it near a river? No.
- Is it near MFA? Yes.
- Bingo: Northeastern!

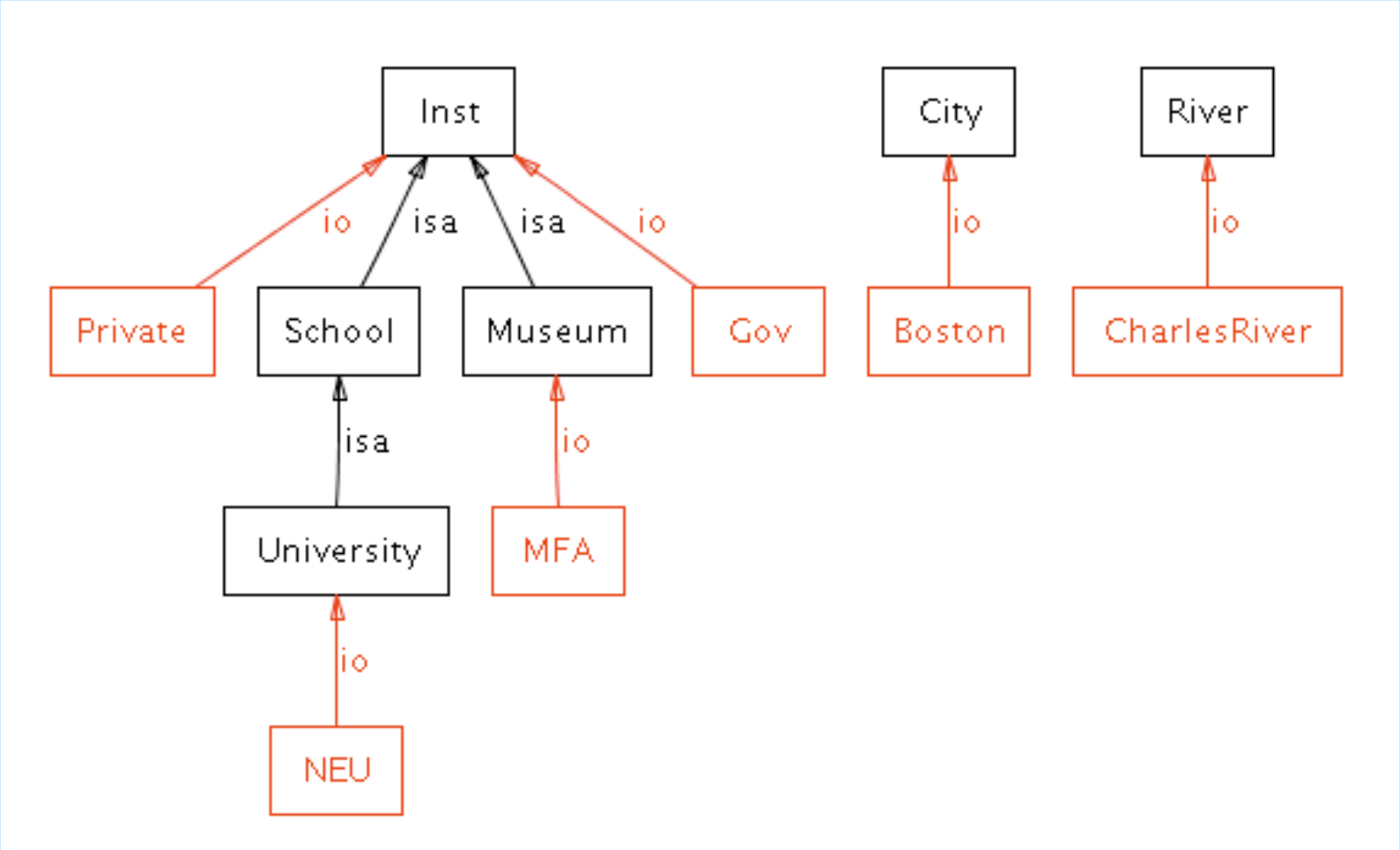
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Terminology → Ontology

- Is it a **thing**? Yes.
- Is it an **institution**? Yes.
- Is it **government owned**? No.
- Is it a **school**? Yes.
- Is it a **university**? Yes.
- Is it **located in Boston**? Yes.
- Is it **near** a **river**? No.
- Is it near **MFA**? Yes.
- Bingo: **Northeastern**!

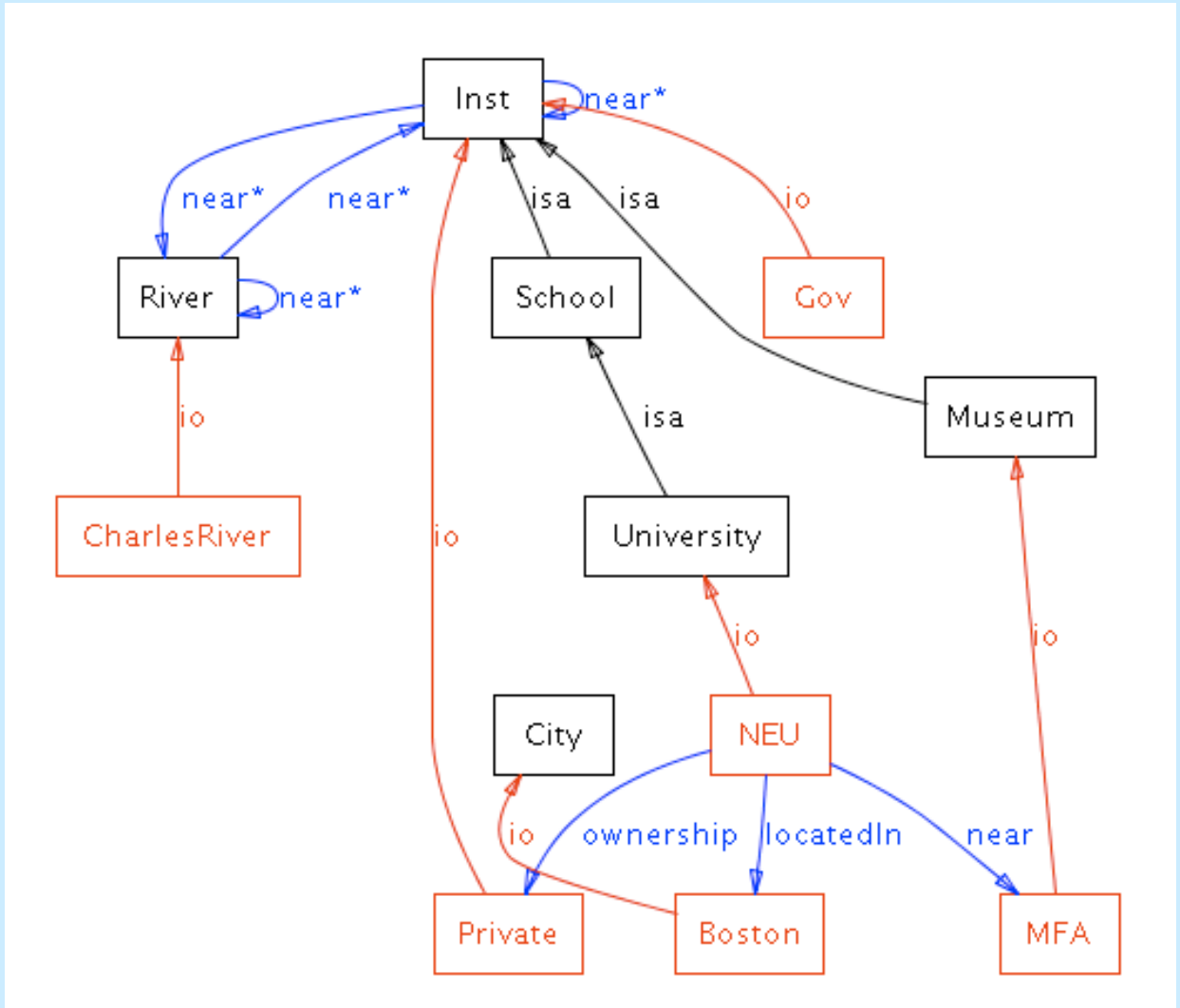
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Ontology



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Annotation



Proceedings of the SDR '08 Technical Conference and Product Exposition.

In Web Ontology Language (OWL)

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.owl-ontologies.com/Ontology1203277937.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.owl-ontologies.com/Ontology1203277937.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="School">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Inst"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="University">
    <rdfs:subClassOf rdf:resource="#School"/>
  </owl:Class>
  <owl:Class rdf:ID="City"/>
  <owl:Class rdf:ID="River"/>
  <owl:Class rdf:ID="Museum">
    <rdfs:subClassOf rdf:resource="#Inst"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="locatedIn">
    <rdfs:domain rdf:resource="#Inst"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="near">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Inst"/>
          <owl:Class rdf:about="#River"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#River"/>
          <owl:Class rdf:about="#Inst"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="ownership">
    <rdfs:domain rdf:resource="#Inst"/>
  </owl:ObjectProperty>
  <River rdf:ID="CharlesRiver"/>
  <Inst rdf:ID="Private"/>
  <City rdf:ID="Boston"/>
  <Museum rdf:ID="MFA"/>
  <Inst rdf:ID="Gov"/>
  <University rdf:ID="NEU">
    <near rdf:resource="#MFA"/>
    <locatedIn rdf:resource="#Boston"/>
    <ownership rdf:resource="#Private"/>
  </University>
</rdf:RDF>
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Querying and Inference

- “Semantics” means we can infer facts that are not explicit in the representation
- Does NEU have students?
  - Yes, since every school has students
- Is NEU in Massachusetts?
  - Yes, since Boston is in Massachusetts
- Is NEU near Symphony Hall?
  - Yes, since MFA is near Symphony Hall
- Give me all Colleges of NEU
  - Technological Entrepreneurship, Engineering, CIS, Arts & Sciences, …

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Semantic Webs

- Web of pages (from www to SW)
- Web of data (from databases to RDF stores)
- Web of services (from SOA to Semantic Web Services)
- Web of actors (humans, agents and all of the above)
- Web of Cognitive Radios

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Towards Cognitive Radio

- Premise: Humans possess cognitive skills
- If Cog Radios are to possess some cognitive skills, it might be worthwhile to mimic some of the humans' cognitive skills in the radios
  - Philosophical views of human cognition
  - Architectural aspects of cognition
  - Procedural vs. declarative languages

Proceedings of the SDR '08 Technical Conference and Product Exposition.

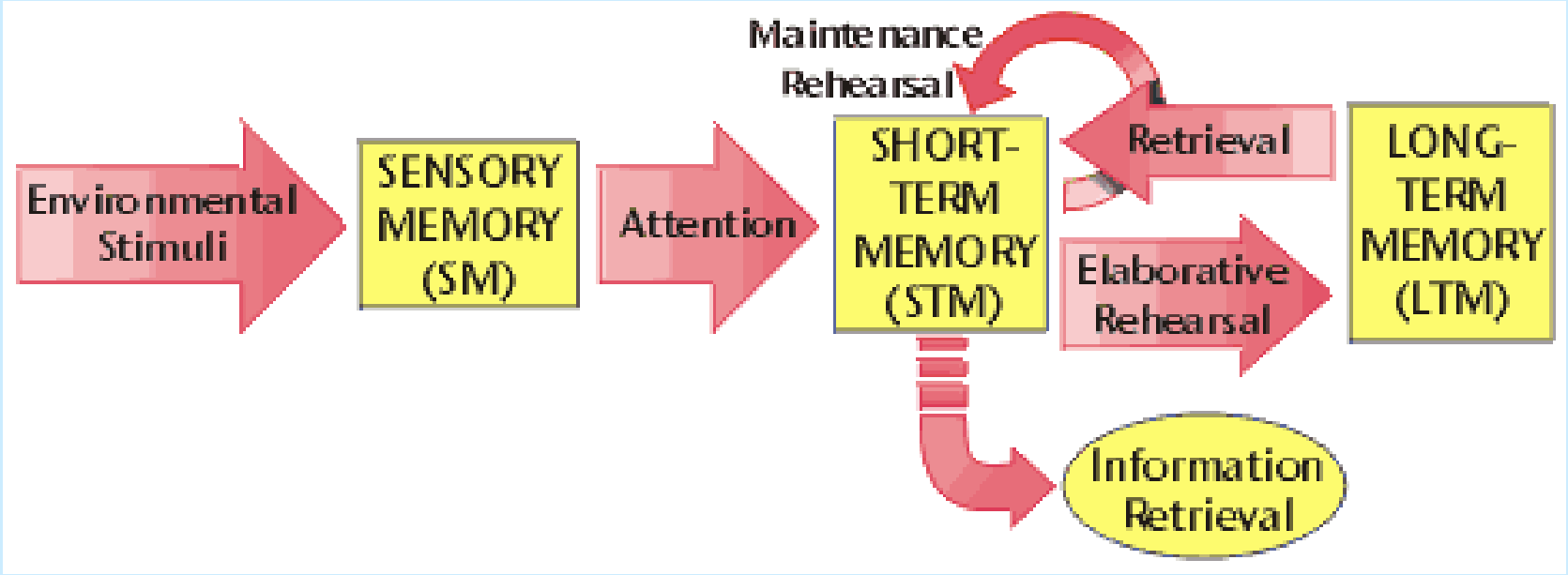
Philosophy of Cognition

- Behaviorism (mainly 50's, 60's and earlier)
  - Observable behaviors, response to environment
  - Input-output reflexive behaviors
  - Focus on "low-level" learning experiments (Pavlov's dogs salivated after hearing the bell)
  - No explicit reference to mental processes/reasoning
- Cognitivism
  - Mental processes "inside the head" subject of study
  - Knowledge as symbolic, mental constructions
  - Explicit storage of (absolute) knowledge in the minds
- Constructivism
  - Knowledge as constructed entity, needs to be constructed by the learner, cannot be just transmitted (unlike in The Matrix!)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Cognitive Architecture

http://evolution.massey.ac.nz/assign2/HBB/modmem1.html

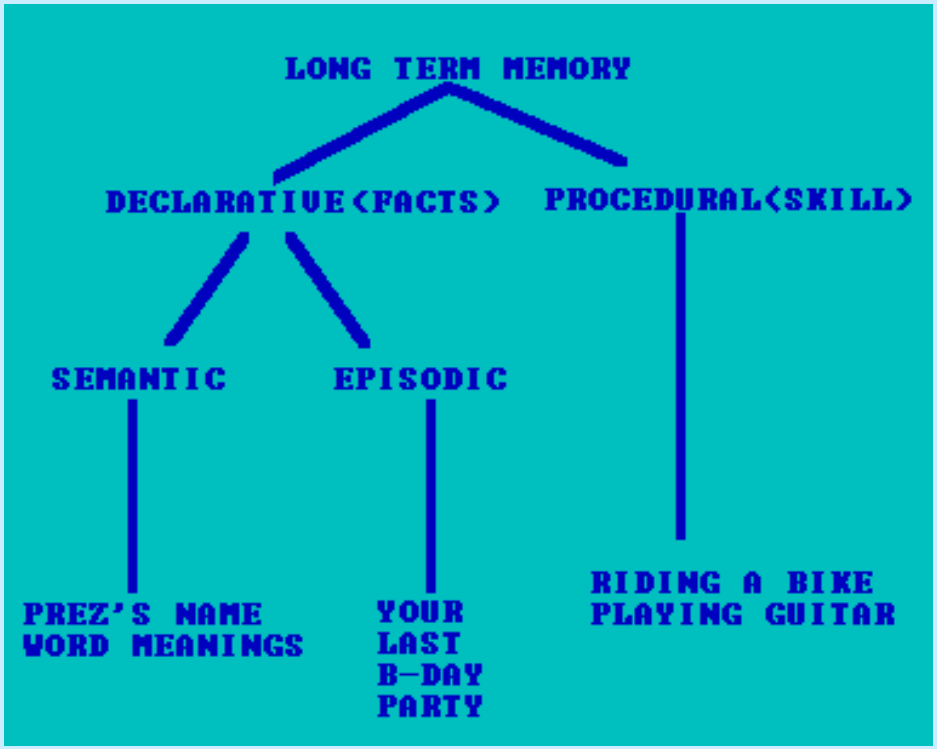


- SM: More than enough, very short – typically less than 1sec, up to 2.5
- STM: 7 +/- 2 “chunks” (digits), up to 18sec, “true cognition” or conscious thought
- LTM: 1 billion bits for 50 years

Proceedings of the SDR '08 Technical Conference and Product Exposition.

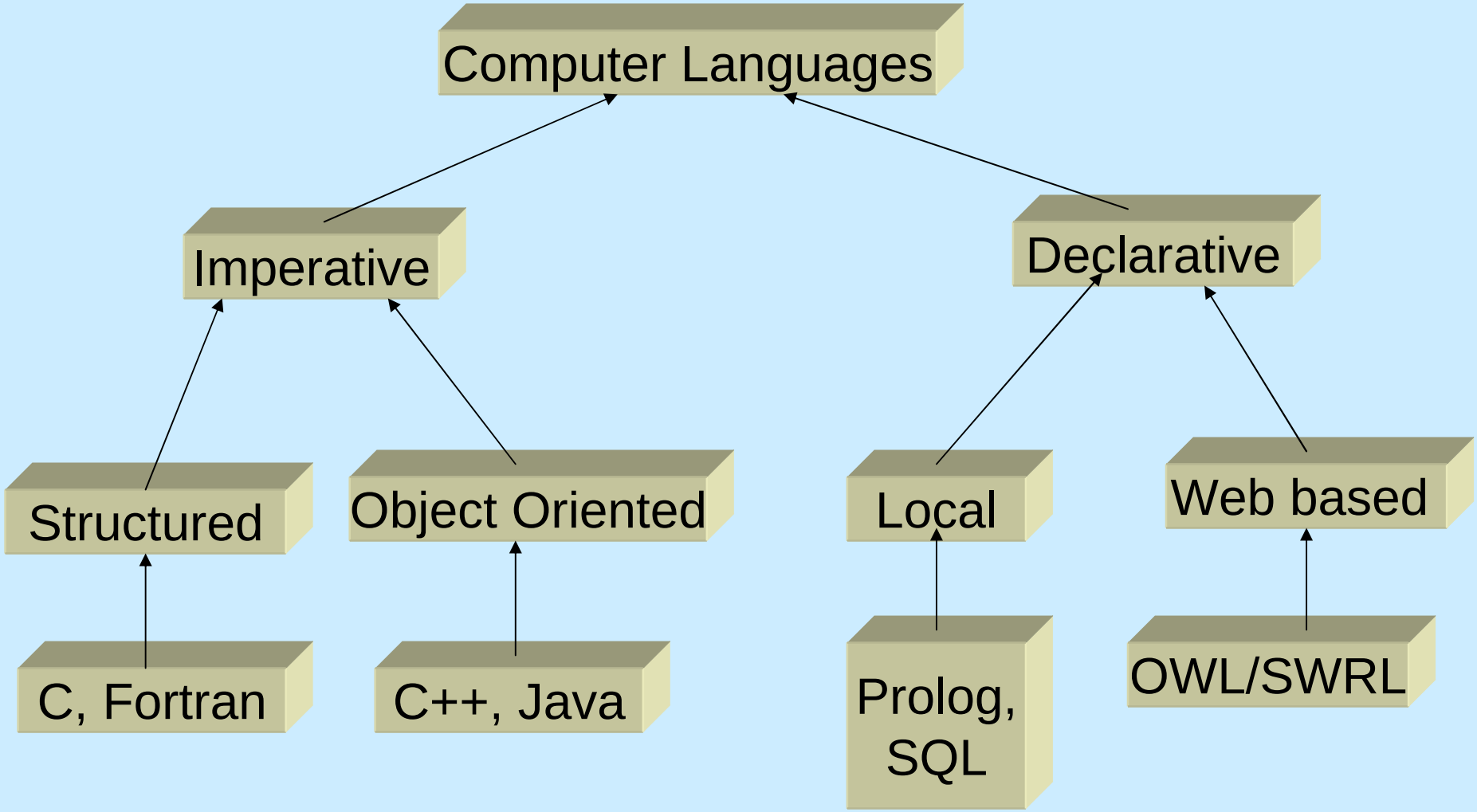
LTM

http://evolution.massey.ac.nz/assign2/HBB/modmem1.html



Procedural: “unconscious effects of learning such as skills and behavioural responses.”
Episodic: What you “remember”
Semantic: What you “know”

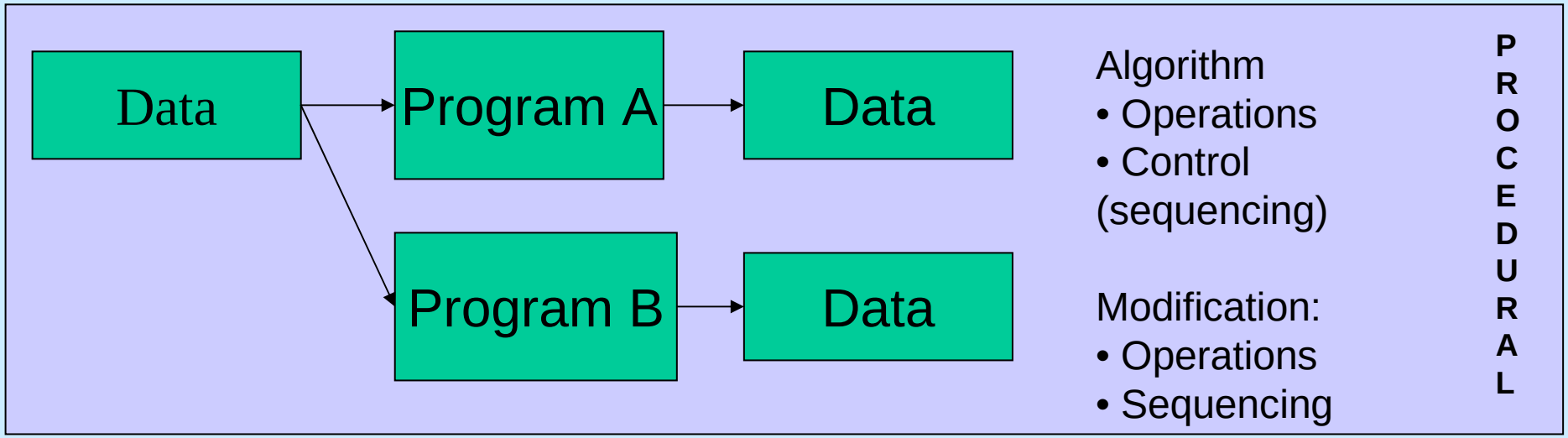
Proceedings of the SDR '08 Technical Conference and Product Exposition.

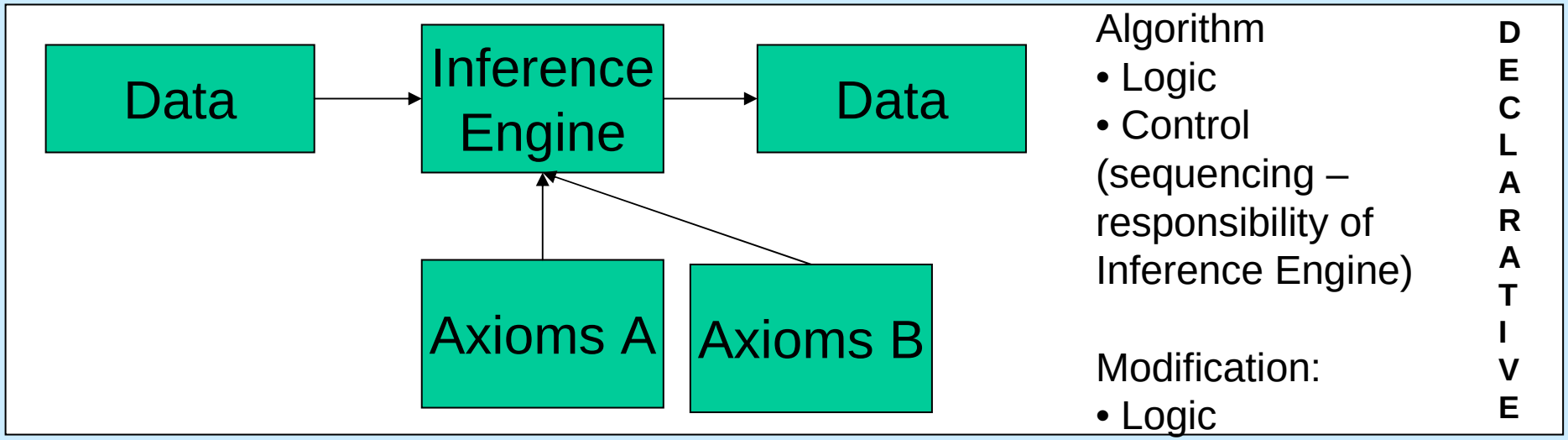


Note: This classification is not complete; not disjoint!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Procedural vs. Declarative Processing





Proceedings of the SDR '08 Technical Conference and Product Exposition.

Program = What + How

- Logical Program = Logical Theory + Deduction
- What = Logical Theory (domain specific)
- How = Deduction (generic)
- Programmer needs to provide only logic (what)
- Logical Program has formal semantics
  - Therefore, it is harnessed for formal verification
  - Important for accreditation of software controlled radios (safety, liveness properties)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example

- Lists

  append(First, Second, Result)

  - append(L1,L2,?L3)
    - Result: new list L3 built out of L1, L2
  - append(L1,L2,L3)
    - Result: “true” or “false” depending on whether L3 is a combination of L1 and L2 (verification)
  - append(?L1,L2,L3)
    - Result: L1 that is a sub-list of L3
- Flexibility due to the ability to use the append operation in many different ways – one definition instead of four

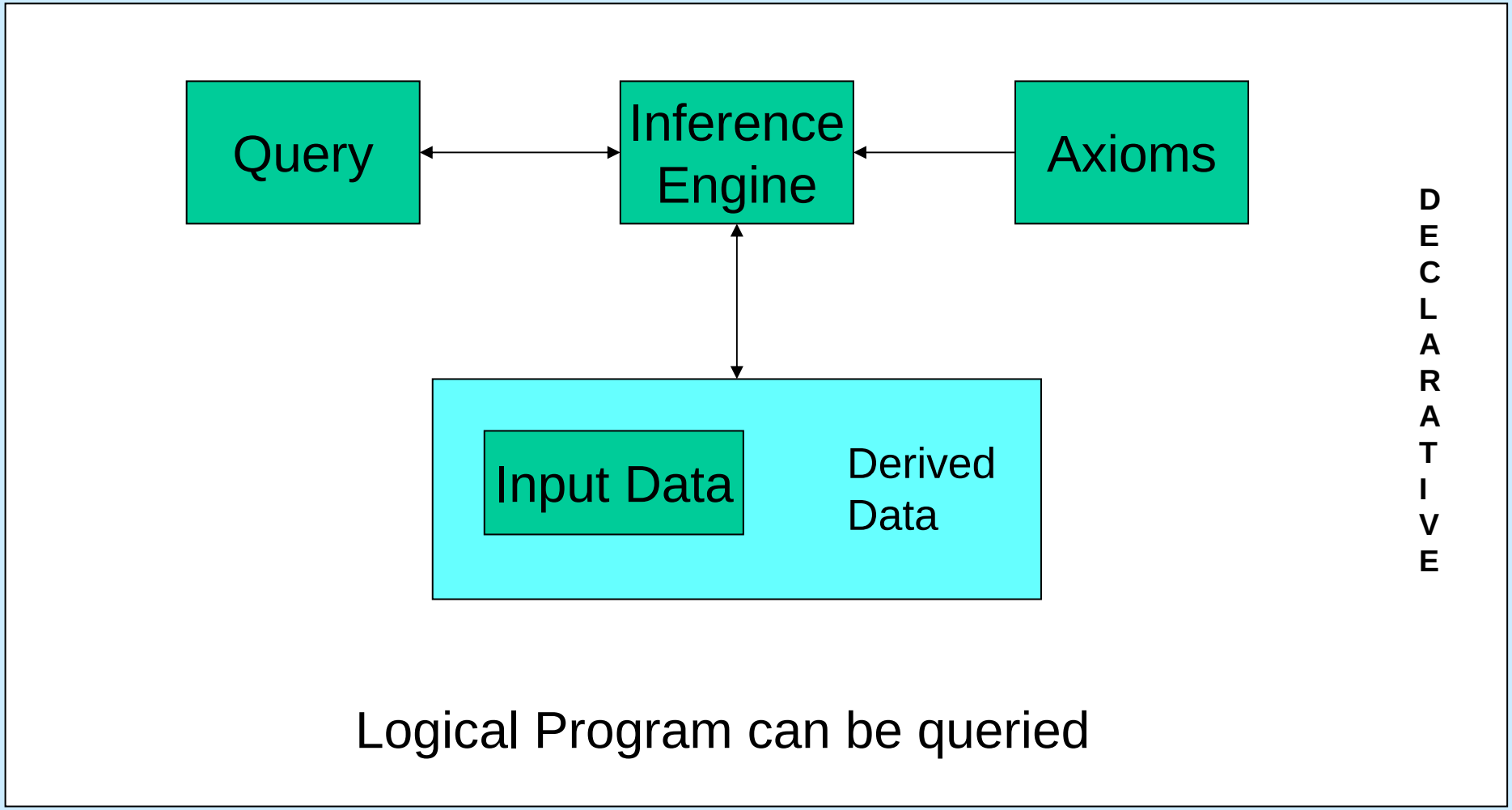
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example

- Numerical constraints: X < Y
- Procedural: if a+b < c*d then … else …
  - Can read in the variables a, b, c, d, but cannot input the expression
- Declarative:
  - Develop domain ontology, add rules
  - Collect input data (annotated w/Ontology)
  - Input any expression for query
    - Numerical
    - **Logical**
  - (a < b) & ~(b>c) OR c*a = d …
  - Get answers to your query

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Logical Program "*knows*" Things



Logical Program can be queried

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Declarative vs. Imperative

- Declarative programming advantages
  - Clear formal semantics (formal verification)
  - Programmer productivity (only “what” not “how”)
  - Querying/flexibility (without knowing “how”)
  - Metaprogramming (programs “understand” their own structure, can manipulate themselves)
- The price for this is performance
  - Being addressed by language developers as we speak
  - More on “complexity” later

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Declarative vs. Procedural

- Declarative languages seem to be a better fit for the CR Requirements:
  - Awareness: not only facts, but also consequences
    - Currently operating within 700MHz band – what's the implication?
    - Reflection – knowing its own variables (most declarative and some procedural support)
  - React to surprise
    - Knowledge not available at design time

Proceedings of the SDR '08 Technical Conference and Product Exposition.

CR, Interoperability and Constructivism

- The theories of cognitivism and constructivism are well suited to the problem of transfer of structural knowledge between cognitive radios.
- One radio (the teacher) can convey the knowledge of its capabilities to another cognitive radio (the learner) by expressing it in terms of *base ontology* that is shared among CRs.
- The important aspect is that the learner's architecture might differ from that of the teacher. Consequently the learner has to interpret the teacher's knowledge and then reconstruct it using its own components and architecture

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Interoperability Scenario

- Assumptions
  - CR nodes share the same *base ontology*
  - CR nodes have a way of communicating with each other; have fall back procedure in case communication between them fails
  - CR nodes can query for and respond to queries for specific facts in their knowledge/databases
  - CR nodes can reason about facts in their knowledge base and facts learned from other nodes through queries
  - A CR node can query for an arbitrary ontology concept. The queried node responds by transferring a fragment of its ontology base related to that concept. The first node can then incorporate that knowledge into its own ontology and can generate software components based on that knowledge.
  - If an unknown ontology concept is expressed in terms of other unknown concepts, the querying can continue recursively until the concept can be expressed in terms of concepts known to the querying node or in terms of *base ontology.*

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example Interaction Sequence (1/2)

Node A		Node B
<query> error bit rate, channel equalizer coefficients</query>	→	
	←	<response>error bit rate = …, channel equalizer coefficients = {…}</response>
[Node A uses the data received from Node B and its own parameters and decides to switch to QAM16]		
<request>change modulation to QAM16</request>	→	
	←	<query>QAM16 modulator</query>
<response>QAM16 modulator is a composite component which consists of quadrature modulator, …, etc.</response>	→	

Example Interaction Sequence (2/2)

Node A		Node B
	←	<query>quadrature modulator </query>
<response>quadrature modulator is a composite component which consists of 2 multipliers, 1 adder, 1 phase shifter, connected in the following way: …</response>	→	
		[Node B builds the model of QAM16 modulator using collected facts and facts in its knowledge base. Uses its resoner to prove that such constructed component is consistent with the specification received from node A.]
	←	<request ack>changing to QAM16</request ack> or <request nack>cannot change to QAM16<request nack>

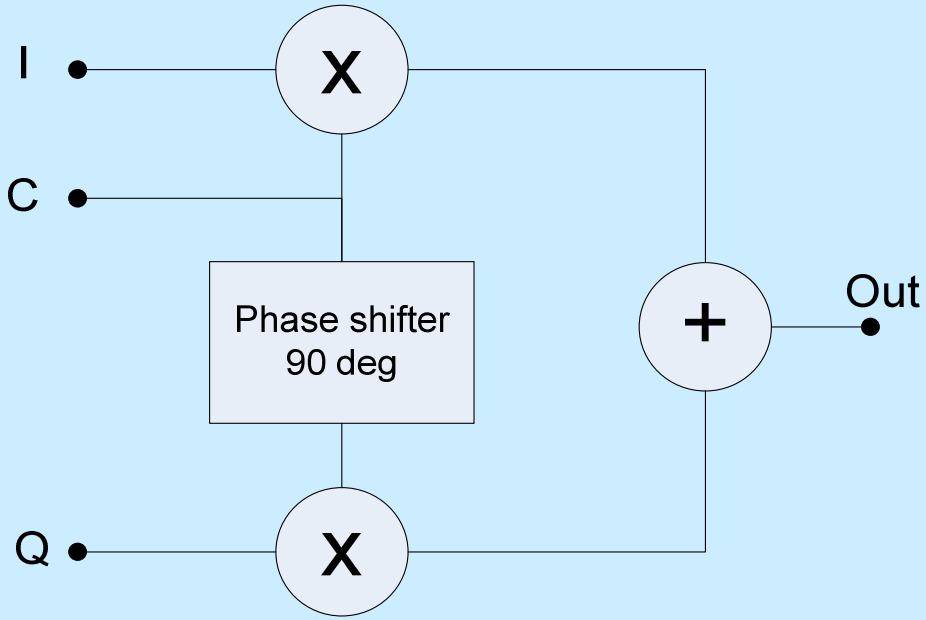
Example (1/2)

- The proposed scenario requires that composite components unknown to a CR could be constructed from other simpler components.
- The component construction algorithm at a certain point has to employ its reasoner to prove that the constructed entity is in fact the desired component.
- Initial research efforts directed towards finding a way to define an OWL class for a composite component (such as quadrature modulator) in terms of other OWL classes representing basic components (such as adder, multiplier, phase shifter etc.)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

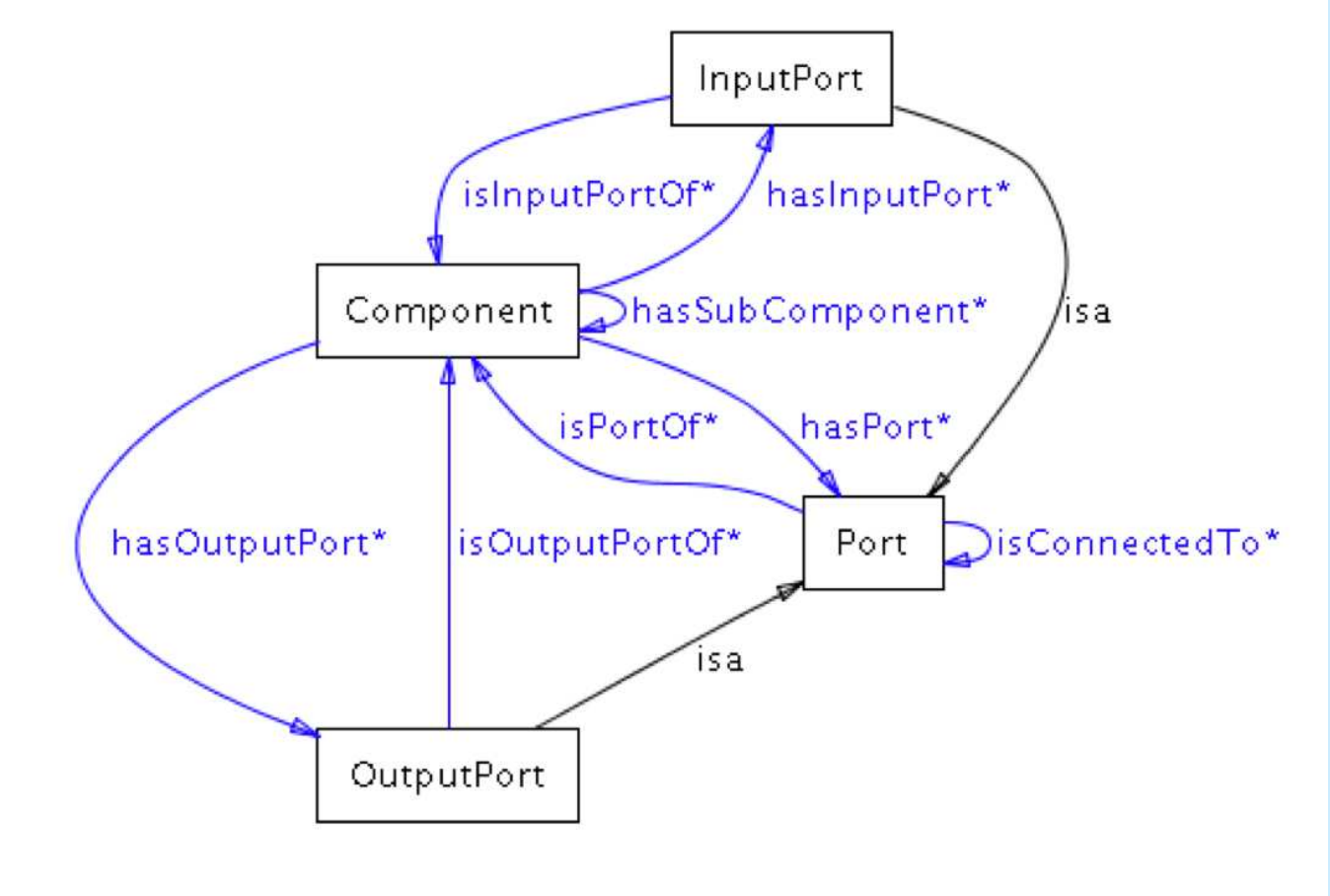
Example (2/2)

- Quadrature Modulator (QM) – perhaps too simplistic, but manageable in a short course



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Ontology: A Small Piece



Class, subclass, individual, property (object, datatype), domain, range, subproperty, fact (triple), restriction, constraints

Ontology in OWL (1/2)

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.LeszekLechowicz.com/RadioTest1.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.LeszekLechowicz.com/RadioTest1.owl">
  <owl:Ontology rdf:about=""/>

  <owl:Class rdf:ID="Port">
    <owl:disjointWith>
      <owl:Class rdf:ID="Component"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="OutputPort">
    <rdfs:subClassOf rdf:resource="#Port"/>
    <owl:disjointWith>
      <owl:Class rdf:ID="InputPort"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#InputPort">
    <rdfs:subClassOf rdf:resource="#Port"/>
    <owl:disjointWith rdf:resource="#OutputPort"/>
  </owl:Class>
  <owl:Class rdf:about="#Component">
    <owl:disjointWith rdf:resource="#Port"/>
  </owl:Class>
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

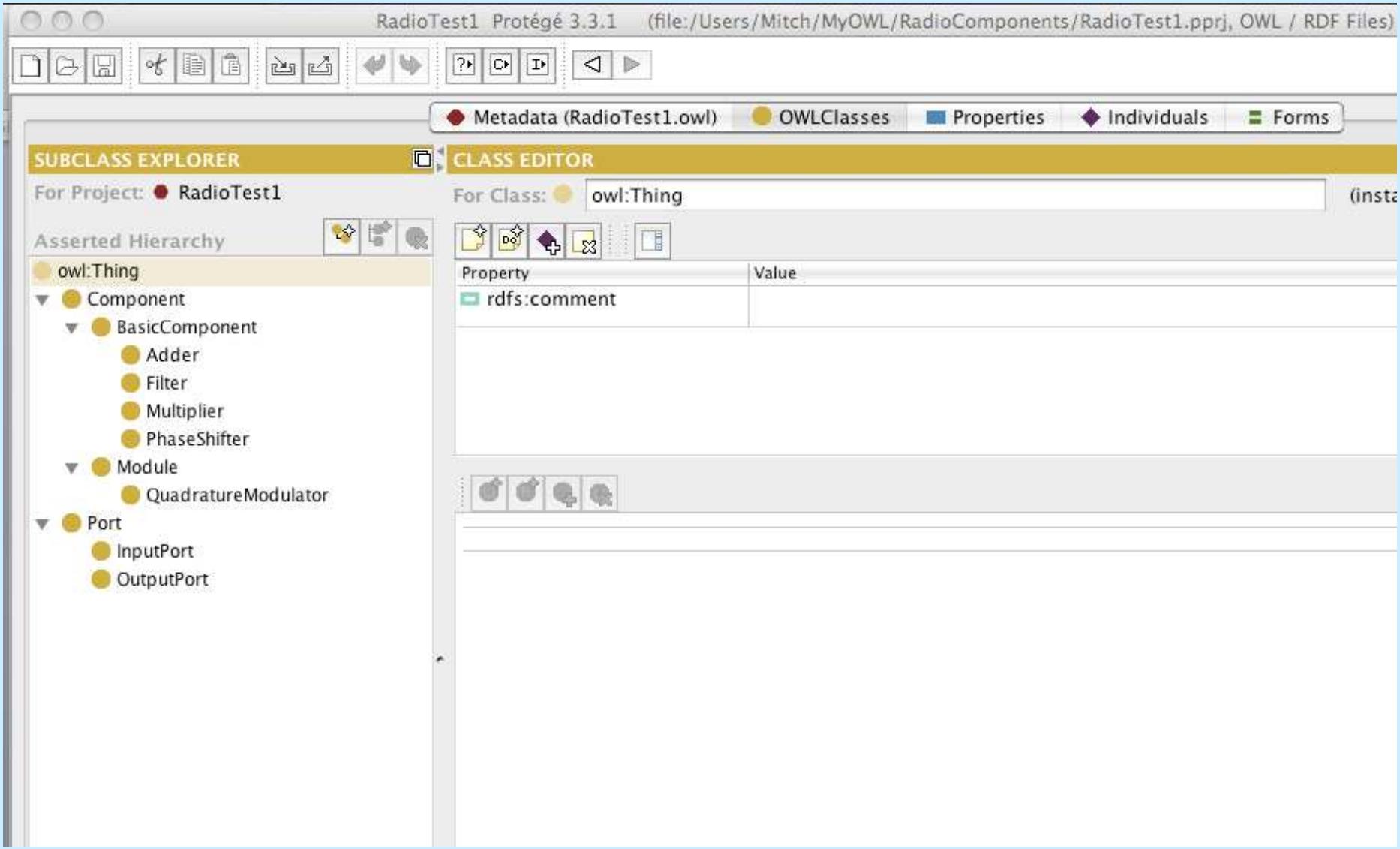
Ontology in OWL (2/2)

```
  <owl:ObjectProperty rdf:ID="hasPort">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="#Port"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="isConnectedTo">
    <rdfs:range rdf:resource="#Port"/>
    <rdfs:domain rdf:resource="#Port"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasInputPort">
    <rdfs:range rdf:resource="#InputPort"/>
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:subPropertyOf rdf:resource="#hasPort"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasSubComponent">
    <rdfs:range rdf:resource="#Component"/>
    <rdfs:domain rdf:resource="#Component"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasOutputPort">
    <rdfs:subPropertyOf rdf:resource="#hasPort"/>
    <rdfs:range rdf:resource="#OutputPort"/>
    <rdfs:domain rdf:resource="#Component"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

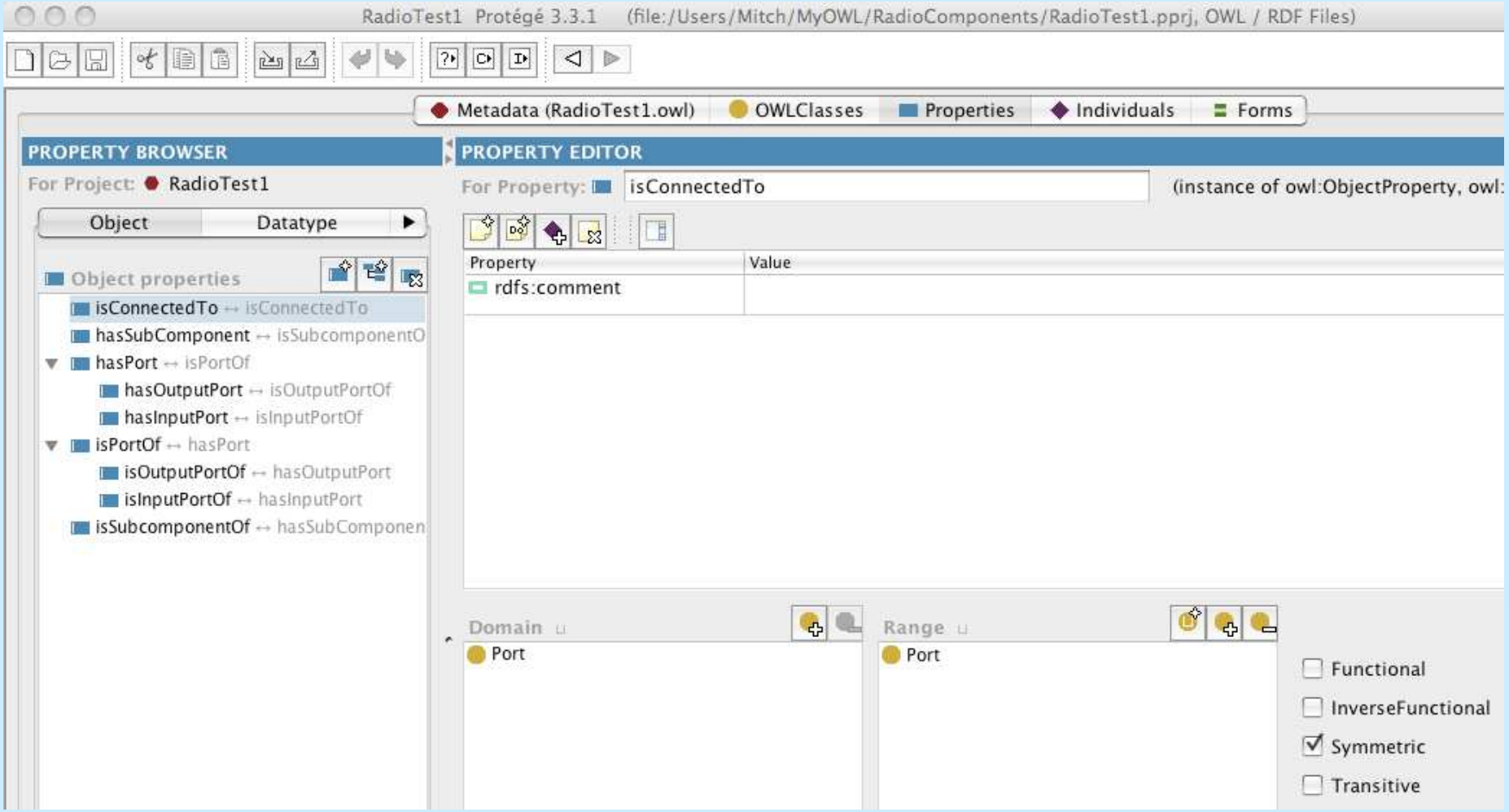
Good news: You don’t need to type this OWL code!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Protégé - Classes

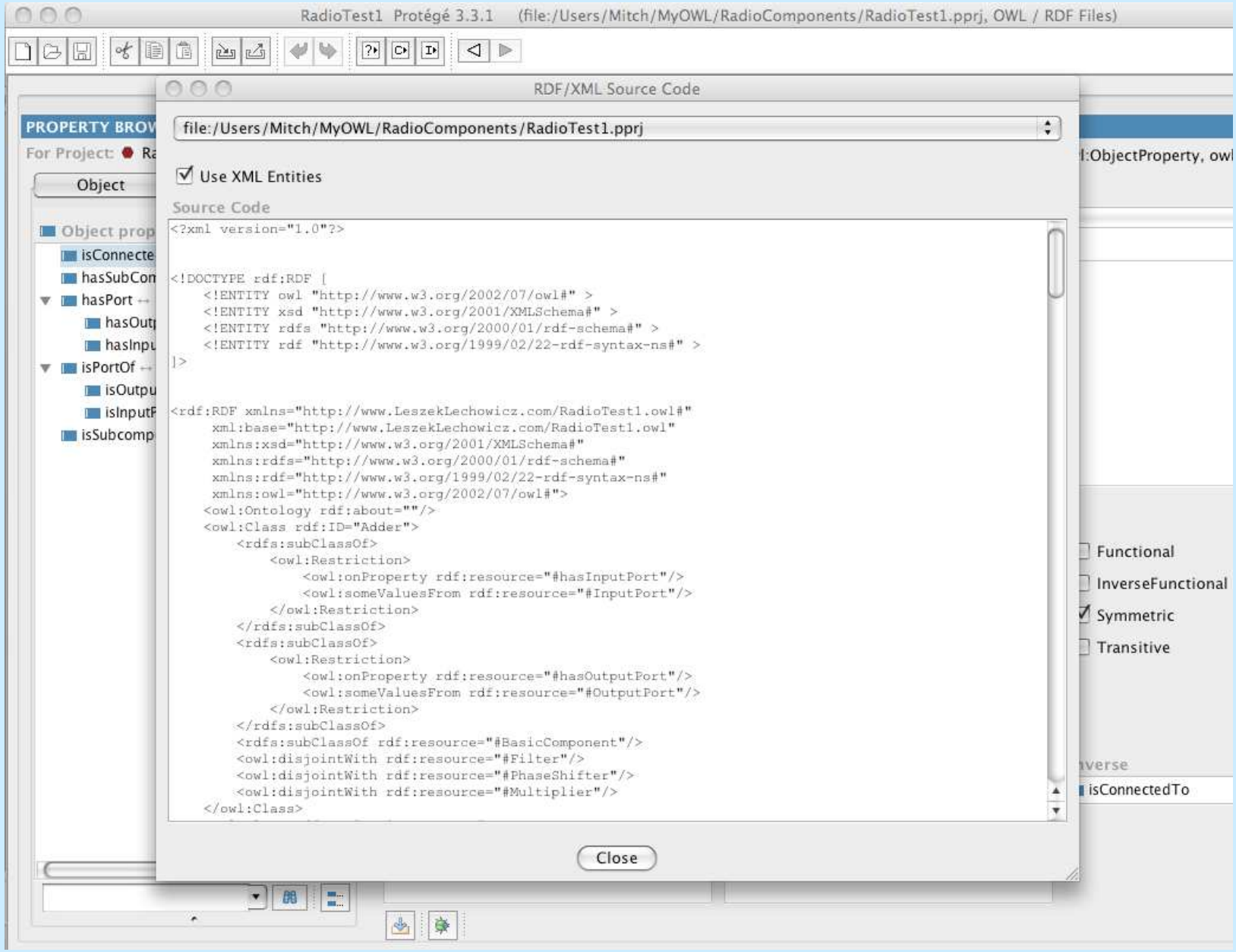


Protégé - Properties



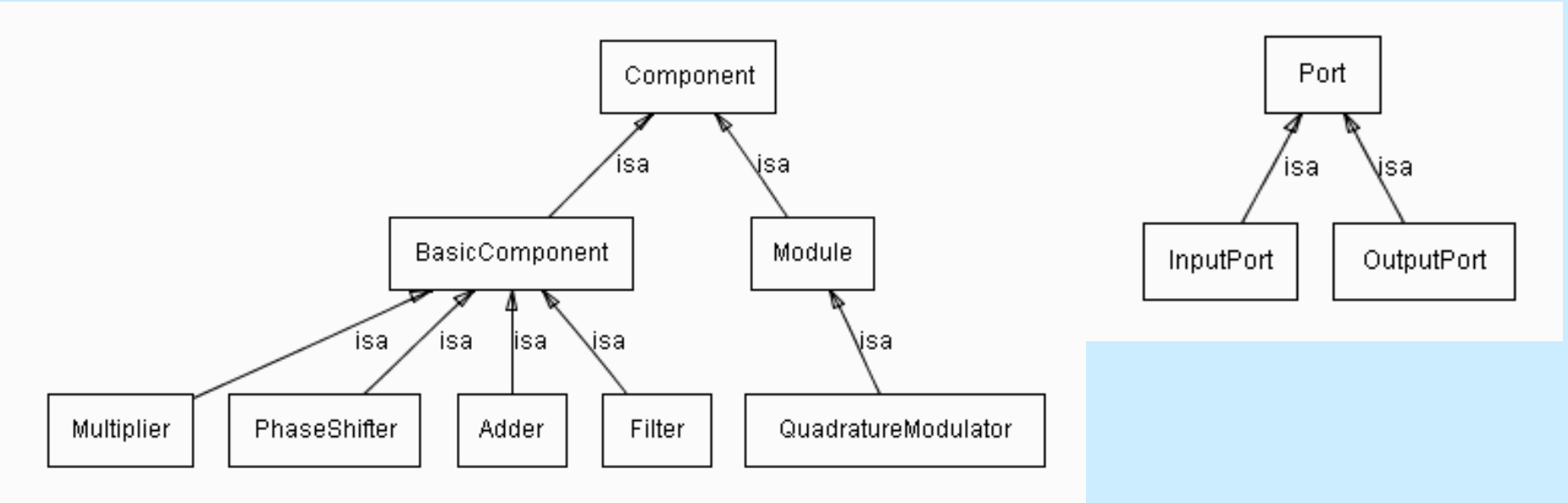
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Protégé – OWL Code



Ontology – Class Hierarchy

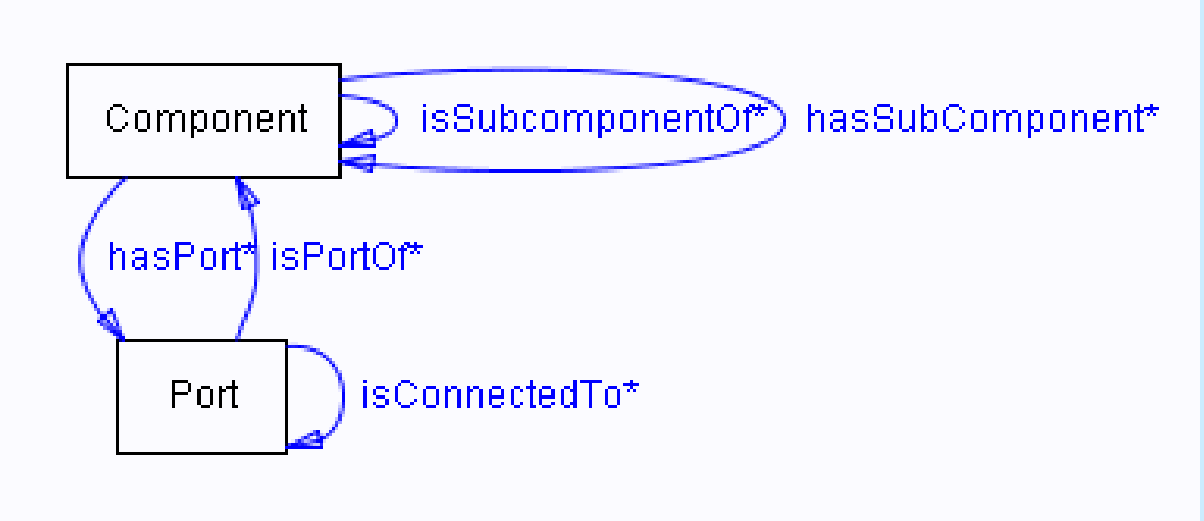
- The experiment used very limited set of classes – the minimum needed to demonstrate the concept.



Proceedings of the SDR '08 Technical Conference and Product Exposition.

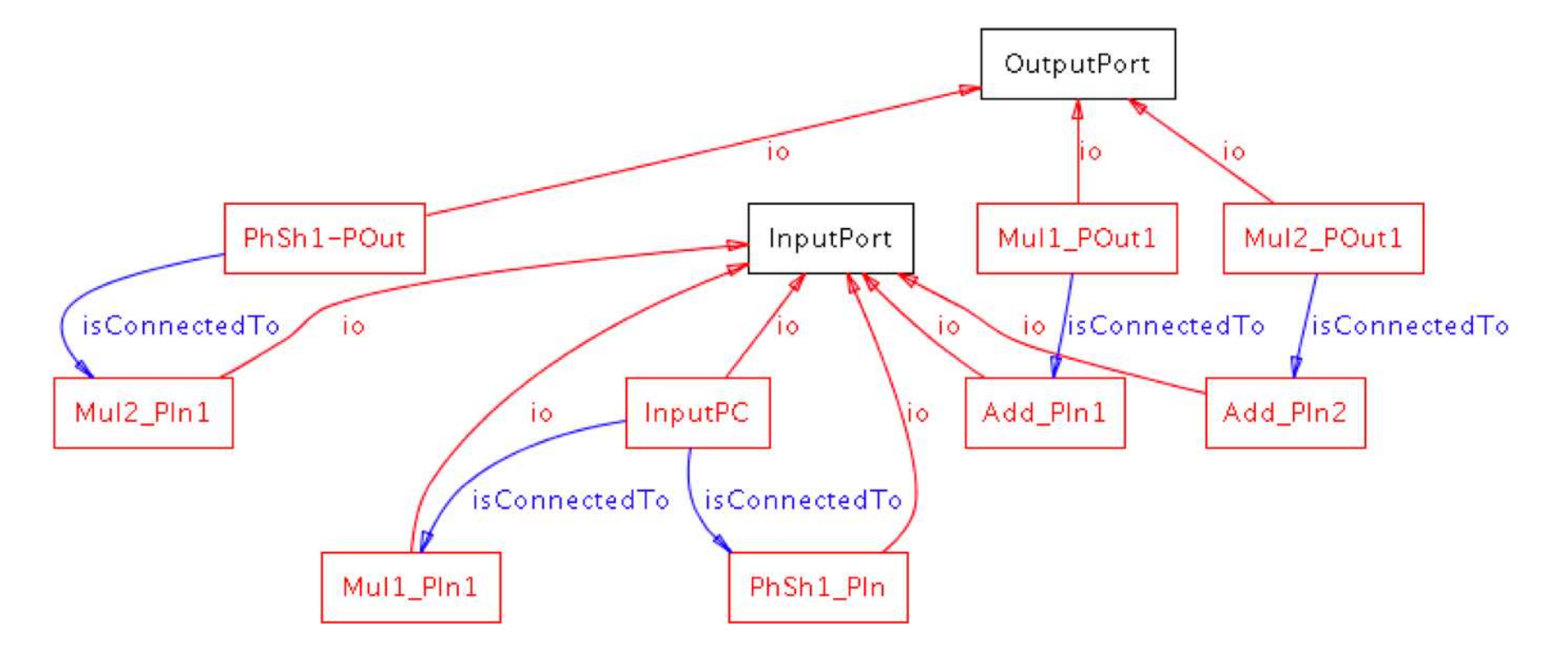
Ontology - Properties

- The domains and ranges of the limited set of properties used in the experiment are shown in the diagram below.



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Ontology (Annotation): Individuals



CR should be able to represent itself in this way and use this representation for telling other radios about itself.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Insufficient Expressiveness of OWL

- Back to the example: CR1 asks CR2 to use QM and gives description of QM class (earlier slide).
- CR2 needs to reconstruct such a component from its own elements and prove it satisfies description of QM.
- The quadrature modulator has two multipliers. In OWL we can express that a particular class (*QuadratureModulator*) is in a relationship with another class (such as *Multiplier*) using a property (in this case *hasSubComponent*). We cannot however distinguish the relationship with one of the multipliers from the relationship with the other one.
- Similarly in OWL we can express the existence of the relationship between *QuadratureModulator* and *InputPort*, but there's no way to distinguish between particular relationships with input ports I, Q and C.

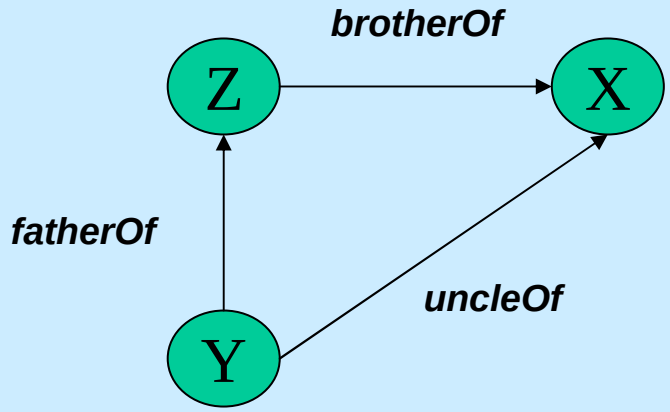
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Can’t represent “uncleOf” in OWL

- Uncle is father’s brother

```
uncleOf(?X,?Y) :-
    father(?Z,?Y), brotherOf(?X,?Z).
```

- OWL does not have a construct for *composing* properties (SWRL does)



Proceedings of the SDR '08 Technical Conference and Product Exposition.

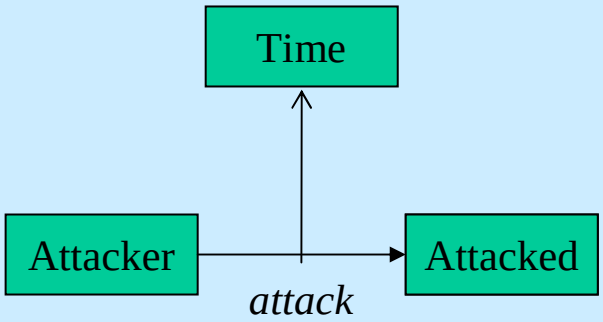
SWRL

- W3C's Semantic Web Rule Language
- Extends representational power of OWL by adding implication in the form of Horn Clauses (i.e., a form of if-then rules)
- Leverages the descriptive capabilities of OWL DL
- Leverages the rule and variable syntax of RuleML

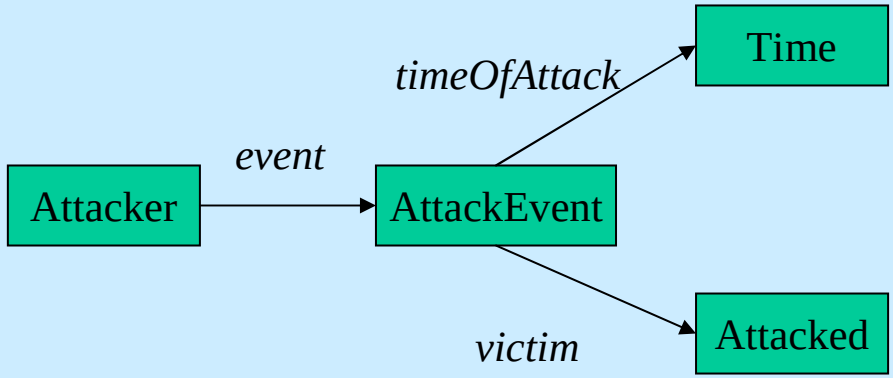
Proceedings of the SDR '08 Technical Conference and Product Exposition.

SWRL: N-ary Relations

- Can't represent
  attack (Attacker, Attacked, Time)
- Instead, need to reify the relation, e.g.,
  event (Attacker, AttackEvent)
  timeOfAttack (AttackEvent, Time)
  victim (AttackEvent, Attacked)
- Have to introduce additional Class and three Properties: *AttackEvent, event, timeOfAttack, victim*
- Problem: Verbose! Multiplicity of choices of reification



Can't do this!



Solution

Horn Clauses

$$\neg A_1 \vee \neg A_2 \vee \ldots. \vee \neg A_n \vee B$$

$$A_1 \wedge A_2 \wedge \ldots. \wedge A_n \Rightarrow B$$

$$B :- A_1 , A_2 , \ldots. , A_n$$

uncleOf(?Z,?Y) :- fatherOf(?X,?Y) $\wedge$ *brother(?X,?Z).*

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Augmenting OWL with Rules

- To express scenarios, like the one with describing component structure, OWL has to be augmented with Rules in order to be able to express more complicated relationships among classes.

hasQMConnections(?QM) :-
Module(?QM),
hasSubComponent(?QM, ?M1),
type(?M1, Multiplier),
hasInputPort(?M1, ?InPortM1),
hasInputPort(?QM, I),
isConnected(I,InPortM1), …,
hasSubComponent(?QM, M2),
!=(M1,M2), …

Proceedings of the SDR '08 Technical Conference and Product Exposition.

BaseVISor Reasoner

- Note that these rules are built “on top of” an ontology. Need an inference engine that can process such rules.
- BaseVISor is a forward-chaining inference engine developed by VIStology, Inc.
- Based on the Rete network optimized for the processing of RDF triples.
- Incorporates axioms and consistency checks for R-entailment which supports all of the RDF/RDFS and a part of OWL-DL and OWL-Full semantics.
- For the price of not supporting all of the OWL-DL, BaseVISor provides P-SPACE performance for ground RDF graphs (complexity will be discussed later in this tutorial).
- BaseVISor is available (free for research purposes) at:

http://www.vistology.com/BaseVISor

Proceedings of the SDR '08 Technical Conference and Product Exposition.

BaseVISor Rule Language (1/2)

- Facts defined by triples consisting of subject, predicate and object

```
<triple>
  <subject resource=”ll:QuadratureModulator/>
  <predicate resource=”ll:hasComponent”/>
  <object resource=”ll:Multiplier”/>
</triple>
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

BaseVISor Rule Language (2/2)

- Rules consist of body and head elements.

```
<rule name="hasMultiplier rule">
  <body>
    <triple>
      <subject variable="comp" />
      <predicate resource="ll:hasSubComponent" />
      <object variable="mul" />
    </triple>
    <triple>
      <subject variable="mul" />
      <predicate resource="rdf:type" />
      <object resource="ll:Multiplier" />
    </triple>
  </body>
  <head>
    <assert>
      <triple>
        <subject variable="comp"/>
        <predicate resource="#hasMultiplier "/>
        <object variable="mul" />
      </triple>
    </assert>
  </head>
</rule>
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

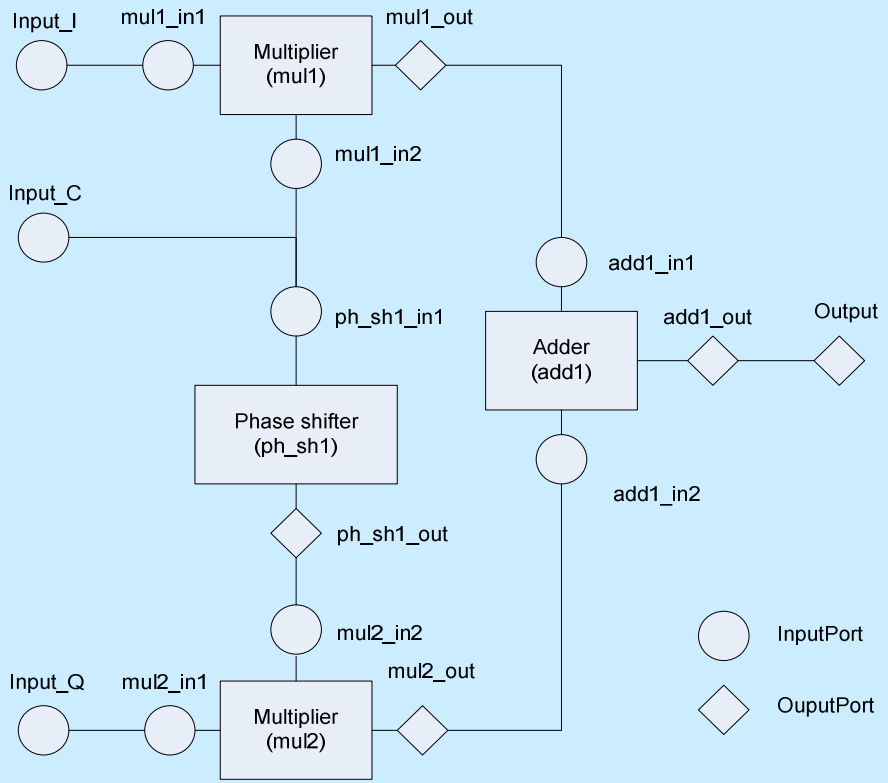
Open vs. Closed World Reasoning

- OWL uses the *Open World Assumption* (OWA)
- Logical Negation: facts that have not been explicitly asserted to be true are not presumed to be false, they simply are *unknown*.
- Monotonic inference: only unknown facts can be proven to be false or true.
- Closed World Assumption (CWA): Everything that is true is known.
- Negation as Failure (NAF): facts that are not known to be true, are false (e.g., flight schedule).
- Non-monotonic inference: new facts must disprove previously known facts (true or false).
- Both OWA and CWA are useful in real life:
  - OWA only: If none of the policies for “can transmit” hold, can’t conclude “cannot transmit”
  - CWA only: If don’t know a CR has a QM, then conclude “it does not have a QM”
- OWL supports “closed domain assumption” (class limited to a specific set of individuals)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Quadrature Modulator Expressed in OWL and Rules

- Graphical representation of the rules' structure.



Port-Component Relationship in Open World Reasoning

- We can express a relationship between a component (e.g. mul1) and its input port (e.g. mul1_in1) by defining a rule for an exclusive input port. Such a rule has to state two things:
  - That there is a relationship between mul1_in1 and mul1
  - It is the only relationship for mul1_in1
- If we create a rule for non-exclusive input port (i.e. an input in relationship with more than one component), then we can use that rule to define exclusive input port simply by excluding all relationships that are non-exclusive
- Semantic Web community is working on the problem of co-existence of OWL and Rules
- For now, the burden is on the implementer of CR

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example: Exclusive vs. Non-Exclusive Port

isExclusiveInputPortOf(P,C) :- inputPortOf(P,C), not(isNonExclusiveInputPortOf(P,C)).

isNonExclusiveInputPortOf(P,C) :- inputPortOf(P,C), inputPortOf(P,D), not(C=D).

- The "*not*" is a NAF negation
- Without this would not be able to prove that a given component satisfies the class description since it might have more components (incomplete knowledge).
- Implementer's responsibility to insure consistency
  - retraction of non-monotonic inferences
  - re-start the inference process
  - insure no non-monotonic inference (no not's)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

So What Have We Accomplished?

- Define a Class of complex components
- Prove that a given component *is* or *is not* an instance of that class
- In other words, given that a CR node A sends a description of a Class of component to CR B that B does not know, B is able to decide whether one of the components that either exists in its library, or a component that it constructs, *is* or *is not* an implementation of the Class that A wants B to use
- Note: This applies to the “same structure” only

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Relevant Work

- D. Preuveneers and Y. Berbers [3]
  - Describe components in OWL
  - Only at “instance level”
- Cannot compare classes of components
  - Did not specify classes
  - Thus could not formulate the problem

Proceedings of the SDR '08 Technical Conference and Product Exposition.

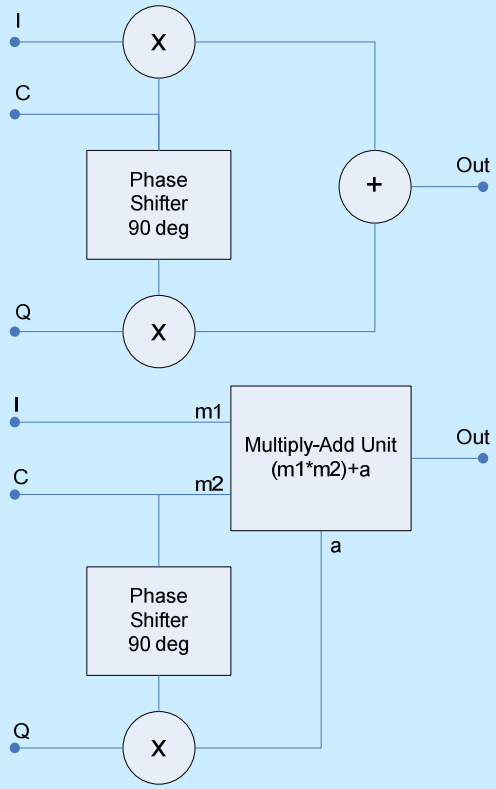
Limitations of Structure Based Approach

- Equivalent software modules can have very different internal structures, for example
  - *f(a,b,c,d) = (a+b)(c+d)* is equivalent to
  - *f(x,y,v,u) = (xv + xu + yv + yu)*.
- The same functional modules operating on different data types are seen as two different structures as the structure-based approach doesn't allow for easy separation of the abstract functionality from the underlying data type
- The lack of "understanding" of the functionality might lead to implementation inefficiencies. For example if a node has a specialized unit for multiply-add operation it might not "realize" that it could use it in place of an adder and a multiplier in a composite module described in terms of base components (see next slide).
- There is no obvious way to express dynamics – time dependencies and constrains.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Limitations of structure based approach

- If a CR node is given a recipe for the quadrature modulator expressed in simple components (as shown in the top picture), unless that node “understands” the desired functionality it will not be able to infer an equivalent but more efficient implementation of such module (as shown in the bottom picture). Structurally those two modules are different, functionally – the same.



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Functionality

- Can express “equivalent property” in OWL, but no mechanism for proving equivalence
- Rules can define a procedure for computing values of a function, but cannot express that two functions are equivalent.
- Use more expressive language for describing components. For instance, chose Metaslang which supports composition and category theory constructs like morphism and colimit.
- Can investigate how to express the fact that two syntactically different components are semantically the same.
- Moreover, we can use Metaslang to capture common parts of different components.
- Use a tool (Specware) that supports the use of Metaslang allowing abstract specifications of radio components and their further refinement through morphisms and colimit operations.
- Use a theorem prover (SNARK) to prove conjectures on functional equivalences of components
- Problem: computational complexity

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Specware

- Specware is Kestrel Institute's framework implementing some of their research results in application of category theory in formalized software development.
- Specware supports systematic construction of software from abstract specifications to executable code through a series of *refinements*.
- An automated theorem prover (such as SRI's SNARK) can be used in each of the refinement steps to prove its correctness.
- If this process is followed rigorously, the resulting code is correct (i.e. it strictly adheres to the axioms defined in the abstract specification).

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Simple Specification in Metaslang

```
BinRel = spec
  type E
  op le : E*E -> Boolean
endspec
```

Can import BinRel and then add additional constraints:

```
PreOrder = spec
  import BinRel
  axiom reflexivity is
    fa(x) x le x = true
  axiom transitivity is
    fa(x,y,z)
      ( x le y ) && ( y le z ) => ( x le z )
endspec
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Specification Morphism

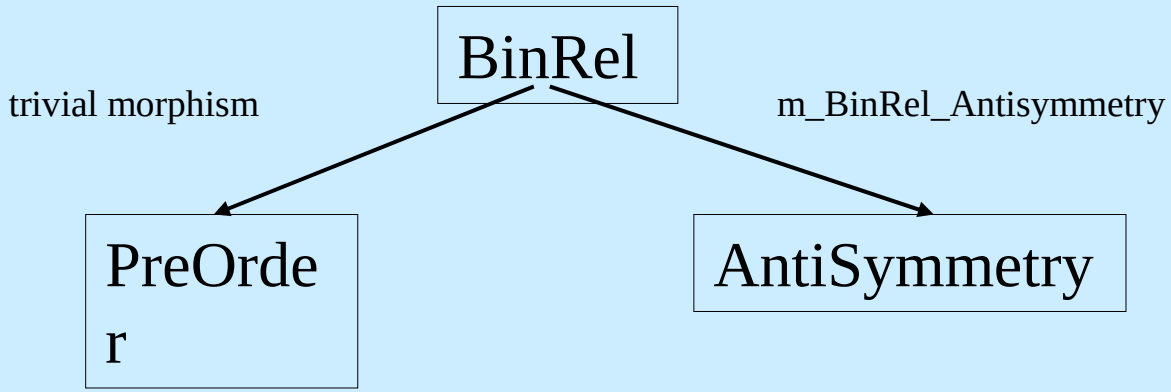
```
Antisymmetry = spec
  type X
  op binOp : X*X -> Boolean
  axiom antisymmetry is fa(x,y)
      binOp(x,y) && binOp(y,x) => x = y
endspec

m_BinRel_Antisymmetry =
     morphism BinRel-> Antisymmetry
       { E +-> X, le+->binOp }
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Specification Diagram

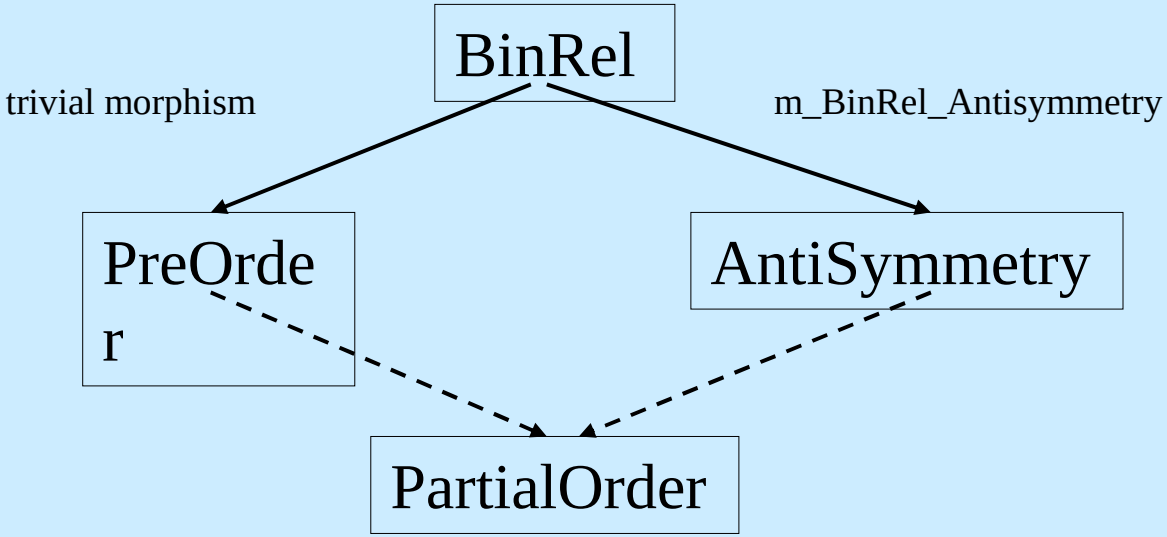
```
BinRelDiag = diagram {
  n1 +-> BinRel,
  n2 +-> PreOrder,
  n3 +-> Antisymmetry,
  e1: n1->n2 +-> morphism BinRel -> PreOrder {},
  e2: n1->n3 +-> m_BinRel_Antisymmetry
}
```



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Colimit

```
PartialOrder = colimit BinRelDiag
```



All theorems from BinRel, PreOrder and AntiSymmetry carry over to PartialOrder.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Abstraction and Commonality of Components

- In the structure-based interoperability approach it is difficult to separate abstract functionality from underlying data type.
- For example a multiply-add unit processing real samples represented by floating point numbers will be composed quite differently than a unit processing pairs of integers representing complex samples.
- In structure-based scheme the commonality of those two modules is lost as they are treated as two completely different entities.
- Specware supports abstract specification and its refinement through morphism and colimit operation.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Abstract Specification with Refinements

```
Samples = spec
  type Sample
  ...
  op Sample.multiply:
       Sample*Sample->Sample
  op Sample.add:
       Sample*Sample->Sample
  ...
endspec

IntSamples = spec
  import Samples
  type Sample = Integer
  ...
  def Sample.multiply(x,y) = x*y
  def Sample.add(x,y) = x+y
  def Sample.minus(x) = -x
  ...
endspec
```

```
CplxIntSamples = spec
  import Samples
  type Sample = { re:Integer, im:Integer}
  ...
  def Sample.multiply(x,y) =
  { re = ( x.re*y.re - x.im*y.im ),
    im = ( x.re*y.im + x.im*y.re ) }
  def Sample.add(x,y) =
  { re = (x.re+y.re), im = (x.im+y.im) }

    ...
endspec

MorphInt =
  morphism Samples -> IntSamples {}
MorphCplxInt =
  morphism Samples -> CplxIntSamples {}
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Specification of Adder with Refinements through Spec Substitution

```
Adder= spec
  import SampleSpec#Samples
  op Adder.Func: Sample*Sample -> Sample
  def Adder.Func(x,y) = Sample.add(x,y)
endspec

Adder_Int = Adder[MorphInt]
Adder_CplxInt = Adder[MorphCplxInt]
```

- An abstract specification of Adder is refined to two concrete specifications through spec substitution (square brackets) operation, which is a simplified colimit.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Functional Equivalence

- The use of theorem provers makes it possible to prove that two software modules are equivalent even when they differ in the structure.
- In Specware one can create conjectures about important properties of the specification. Those conjectures can be later proved by the inference engine.
- The limitation of SNARK is its ability to reason in the first order logic only. More advanced refinements requiring higher order logic might not be provable with SNARK even though they are correct.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example: Equivalent Functions

```
Funcs = spec
  import Adder
  import Multiplier
  op Funcs.Func1: Sample*Sample*Sample -> Sample
  def Funcs.Func1(a,b,c) =
       Multiplier.Func( a, Adder.Func(b,c) )
  op Funcs.Func2: Sample*Sample*Sample -> Sample
  def Funcs.Func2(a,b,c) =
        Adder.Func( Multiplier.Func(a,b),
                    Multiplier.Func(a,c) )
  conjecture Funcs_eq_conj is
    fa(a:Sample, b:Sample, c:Sample)
      Func1(a,b,c) = Func2(a,b,c)
endspec

p0 = prove Funcs_eq_conj in Funcs options
        "(use-resolution t) (use-paramodulation t)"
```

a*(b+c)

a*b + a*c

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Temporal Logic Elements

- In practical implementations of CR algorithms timing constraints have to be considered when constructing composite modules.
- *Temporal logics* try to tackle different aspects of time in complex system without introducing time explicitly. In our experiments we used a simple concept from Linear Temporal Logic – the operator X (neXt).
- In discrete time systems (as all sample-based systems are) the operator X is simply a delay element.
- Operator X is not a function as understood by functional languages – it uses a side effect (remembered value) to compute the returned value.
- Since Metaslang is a functional language it cannot implement the X operator. That limitation however does not prevent us from defining the X operator as an abstract operation with some axioms thus enabling reasoning about it.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Example - Delay Specification

```
UnitDelaySpec = spec
  import Samples
  op     UnitDelay.Func: Sample -> Sample
  axiom UnitDelay_commutativity is
    fa( f:( Sample->Sample ), x:Sample )
      UnitDelay.Func( f(x) ) = f( UnitDelay.Func(x) )

  axiom UnitDelay_commutativity2 is
    fa( f:( Sample*Sample->Sample ), x:Sample, y:Sample )
      UnitDelay.Func( f(x,y) ) = f( UnitDelay.Func(x), UnitDelay.Func(y) )
endspec
```

Note: Quantification over functions!

```
AdderDelay = spec
  import UnitDelaySpec
  op Adder.Func: Sample*Sample -> Sample
  def Adder.Func(x,y) = UnitDelay.Func( Sample.add( x, y ) )
endspec

MultiplierDelay = spec
  import UnitDelaySpec

  op Multiplier.Func: Sample*Sample -> Sample
  def Multiplier.Func(x,y) = UnitDelay.Func( Sample.multiply(x,y) )
endspec
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Behavioral Equivalence

```
MACSpec = spec
  import UnitDelaySpec
  op MAC.Func: Sample*Sample*Sample -> Sample
  def MAC.Func(m1, m2, a) =
    UnitDelay.Func(
       Sample.add(
          UnitDelay.Func( Sample.multiply(m1, m2) ),
          UnitDelay.Func( a ) ) )
endspec

CompositeMACSpec = spec
  import AdderDelay
  import MultiplierDelay
  import MACSpec
  op CompositeMAC.Func: Sample*Sample*Sample -> Sample
  def CompositeMAC.Func(m1, m2, a) =
      Adder.Func( Multiplier.Func(m1,m2), UnitDelay.Func(a) )

  conjecture CompositeMAC_conj is
    fa( m1:Sample, m2:Sample, a:Sample )
      CompositeMAC.Func( m1, m2, a ) = MAC.Func( m1, m2, a )
endspec

p0 = prove CompositeMAC_conj in CompositeMACSpec options
           "(use-resolution t) (use-paramodulation t)"
```


Proceedings of the SDR '08 Technical Conference and Product Exposition.

Conclusions so far

- We applied elements of category theory to Cognitive Radio interoperability problem.
- Demonstrated the feasibility of this approach as we were able to overcome the shortcomings of the structure-based approach we proposed previously.
- Current theorem provers that are necessary for this solution are relatively slow (need investment!)
- The fact that Metaslang is a functional language limited our success with the application of elements of temporal logic to software component application. We were unable to generate source code for modules using the X operator. We were able however to use its abstract definition and axioms in proving functional equivalence of module specifications.

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Problem Complexity

The **time complexity** of a problem is the number of steps that it takes to solve an instance of the problem, as a function of the **size of the input**, (usually measured in bits) using the *most efficient* **algorithm**.

For sufficiently large n: $\log n < n < n \log n < n^2 < n^3 < 2^n$

Since we consider the worst case, we really take into account only *order* of the function – *big-Oh* notation:

$O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$

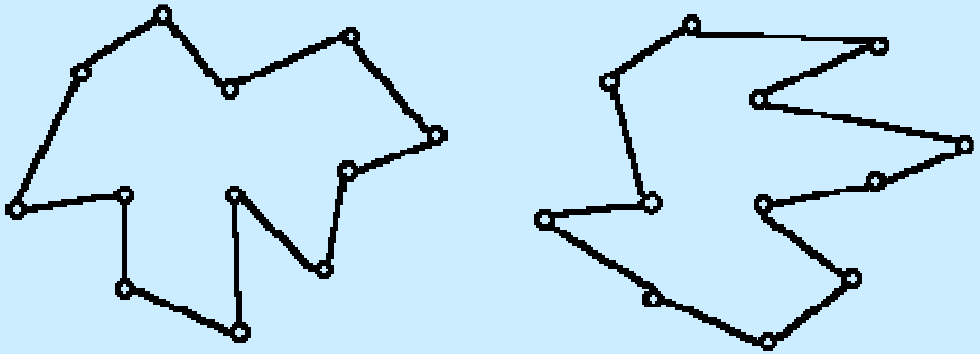
Examples: O(g(n))

Sequential search:	O(n)
Binary search:	O(log n)
Merge sort:	O(n log n)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

TSP: Traveling Salesman Problem

Problem: *Given the coordinates of n cities, find the shortest closed tour which visits each city exactly once.*



Brute force (enumeration of all possible tours): O(n!)

Assume: n=10 takes 1 sec
Result: n=20 would take 20!/10! = 20,000 years

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Faster CPUs?

Time complexity function	Size of Largest Problem Instance solvable in 1 Hour		
	With present computer	With computer 100 times faster	With computer 1000 times faster
n	N1	100N1	1000N1
n^2	N2	10 N2	31.6 N2
n^3	N3	4.64 N3	10 N3
2^n	N4	N4+6.64	N4+9.97
3^n	N5	N5+4.19	N5+6.29

If a TSP for 300 cities solvable in a reasonable time limit, a 1000 faster CPU would not be able to solve a 302 city TSP!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Decision (Recognition) and Optimization Problems

Optimization problems: minimize/maximize a cost function given constraints

Decision (recognition) problems: require a “yes” or “no” answer

- Special case of optimization (no cost function)
- For each optimization problem there is a decision version
- Complexity results also hold for the original optimization problem

Example: TSP Decision Problem:

Is there a closed tour passing each city only once, with total length $\leq$ L?

- Have a TSP solution – then just compare it to L
- Have a TSP-DP solution – use binary search to find the optimal solution

Proceedings of the SDR '08 Technical Conference and Product Exposition.

The SAT Problem (satisfiability)

Problem: Given a Boolean formula in Conjunctive Normal Form (CNF), is it satisfiable? That is, is there a set of "true-false" values to be assigned to the various variables, such that the compound proposition is true?

Example:

$(x_1 \text{ OR } x_2 \text{ OR } x_3)$ and $(x_1 \text{ OR } \neg x_2)$ and $(x_2 \text{ OR } \neg x_3)$ and $(x_3 \text{ OR } \neg x_1)$ and $(\neg x_1 \text{ OR } \neg x_2 \text{ OR } \neg x_3)$

Proceedings of the SDR '08 Technical Conference and Product Exposition.

P and NP

P – recognition problems for which a polynomial time algorithm exists

NP – Non-deterministic Polynomial – both encoding and verification of solution are polynomial, even if the solution is guessed

Example: Integer Programming Problem

Given an m x n integer matrix A and an integer m-vector b,
is there a n-vector x, with elements 0 or 1, such that Ax=b ?

If we have a potential solution, it's easy to check whether it is a solution or not (a valid *certificate* exists)

NP – class of "reasonable" problems.
P is subset of NP

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Polynomial Reducibility

Problem A reduces in polynomial-time to another problem B, if and only if:

1. there is an algorithm for A which uses a subroutine for B, and
2. each call to the subroutine for B counts as a single step, and
3. the algorithm for A runs in polynomial-time.

Write: $A \propto B$

If $A \propto B$ and B is in P, then A is in P

If $A \propto B$ then B is at least as "hard" as A

Example:

TSP $\propto$ TSP Decision Problem

SAT $\propto$ Integer Programming Decision Problem

Proceedings of the SDR '08 Technical Conference and Product Exposition.

NP-Completeness

A problem is NP-complete if:

- it belongs to class NP
- all other problems in NP are reducible to it

Theorem: SAT is NP-complete

Cook's Theorem:

Let A be a recognition problem.

Then, the following propositions are equivalent:

1. $A \in NP$
2. A is polynomially solvable by a non-deterministic algorithm
3. A polynomially transforms to SAT ($A \propto SAT$)

Proceedings of the SDR '08 Technical Conference and Product Exposition.

NP-hard

A problem *Pr* is NP-hard if SAT $\propto$ *Pr*

Example:
SAT $\propto$ Integer Programming Problem, thus it is NP-hard

Note: NP-complete problems are NP-hard, but the opposite does no have to be true

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Still Other Classes

EXPTIME – problems solvable (deterministically) in $O(2^{p(n)})$ time, where *p* – polynomial
Note: complexity guaranteed to be exponential, unlike in NP

NEXPTIME – problems solvable by a non-deterministic algorithm in $O(2^{p(n)})$ time, where *p* – polynomial

UNDECIDABLE – if there is no algorithm that can always give the correct answer.

Example: The Halting Problem: *Given a description of an algorithm and its initial input, determine whether the algorithm, when executed on this input, ever halts (the alternative is that it runs forever without halting).*

Note: Individual instances of the problem can be solvable!

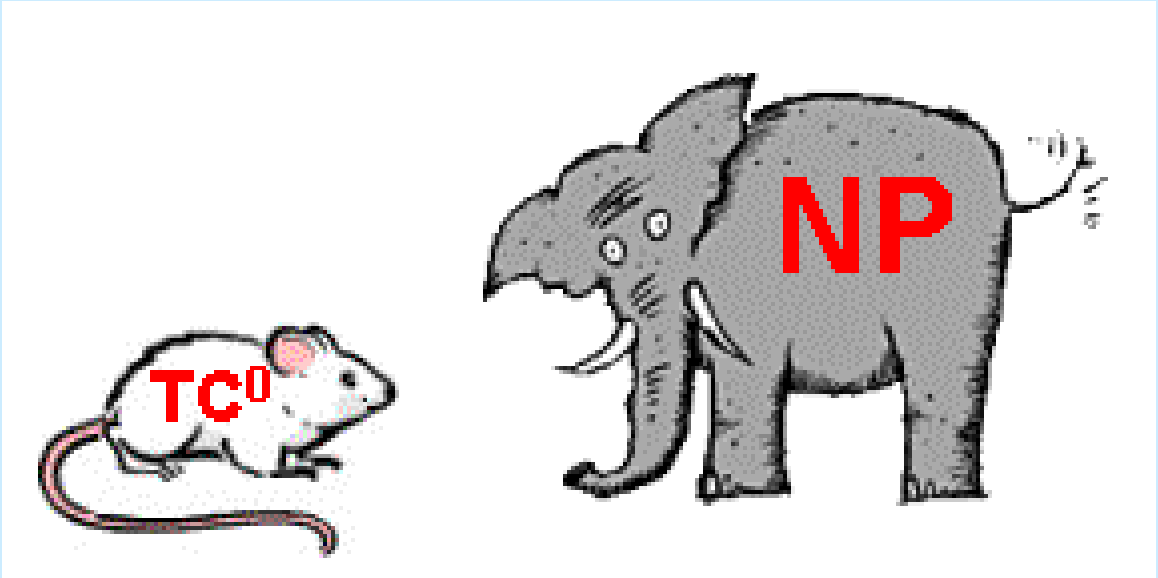
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Complexity Classes

0-1-NP_C - $\#AC^0$ - #L - #L/poly - #P - #W[t] - +EXP - +L - +L/poly - +P - $+SAC^1$ - A_0PP - AC - AC^0 - $AC^0[m]$ - ACC^0 - AH - AL - AlgP/poly - AM - AM-EXP - AM intersect coAM - AM[polylog] - AmpMP - AmpP-BQP - AP - AP - APP - APP - APX - AUC-SPACE(f(n)) - AVBPP - AvE - AvP - AW[P] - AWPP - AW[SAT] - AW[*] - AW[t] - βP - BH - BPE - BPEE - $BP_HSPACE(f(n))$ - BPL - BP•NP - BPP - BPP^{cc} - BPP^{KT} - BPP-OBDD - BPP_{path} - BPQP - BPSPACE(f(n)) - BPTIME(f(n)) - BQNC - BQNP - BQP - BQP/log - BQP/poly - BQP/qlog - BQP/qpoly - BQP-OBDD - BQP_{tt}/poly - BQTIME(f(n)) - k-BWBP - $C_=AC^0$ - $C_=L$ - $C_=P$ - CFL - CLOG - CH - Check - C_kP - CNP - coAM - $coC_=P$ - cofrIP - Coh - coMA - $coMod_kP$ - compIP - compNP - coNE - coNEXP - coNL - coNP - $coNP^{cc}$ - coNP/poly - coNQP - coRE - coRNC - coRP - coSL - coUCC - coUP - CP - CSIZE(f(n)) - CSL - CZK - D#P - $Δ_2P$ - δ-BPP - δ-RP - DET - $DiffAC^0$ - DisNP - DistNP - DP - DQP - DSPACE(f(n)) - DTIME(f(n)) - DTISP(t(n),s(n)) - Dyn-FO - $Dyn\text{-}TC^0$ - E - EE - EEE - EESPACE - EEXP - EH - ELEMENTARY - EL_kP - EPTAS - k-EQBP - EQP - EQTIME(f(n)) - ESPACE - ExistsBPP - ExistsNISZK - EXP - EXP/poly - EXPSPACE - FBQP - Few - FewP - FH - FNL - FNL/poly - FNP - FO(t(n)) - FOLL - FP - $FP^{NP[log]}$ - FPR - FPRAS - FPT - FPT_{nu} - FPT_{su} - FPTAS - FQMA - frIP - F-TAPE(f(n)) - F-TIME(f(n)) - GA - GAN-SPACE(f(n)) - $GapAC^0$ - GapL - GapP - GC(s(n),C) - GI - GPCD(r(n),q(n)) - G[t] - H_kP - HVSZK - IC[log,poly] - IP - IPP - L - LIN - L_kP - LOGCFL - LogFew - LogFewNL - LOGNP - LOGSNP - L/poly - LWPP - MA - MA' - MAC^0 - MA-E - MA-EXP - mAL - MaxNP - MaxPB - MaxSNP - $MaxSNP^0$ - mcoNL - MinPB - MIP - MIP*[2,1] - MIP_{EXP} - $(M_k)P$ - mL - mNC^1 - mNL - mNP - Mod_kL - Mod_kP - ModP - $ModZ_kL$ - mP - MP - MPC - mP/poly - mTC^0 - NC - NC^0 - NC^1 - NC^2 - NE - NE/poly - NEE - NEEE - NEEXP - NEXP - NEXP/poly - NIQSZK - NISZK - $NISZK_h$ - NL - NL/poly - NLIN - NLOG - NP - NPC - NP^{cc} - NP_C - NPI - NP intersect coNP - (NP intersect coNP)/poly - NPMV - NPMV-sel - $NPMV_t$ - $NPMV_t$-sel - NPO - NPOPB - NP/poly - (NP,P-samplable) - NP_R - NPSPACE - NPSV - NPSV-sel - $NPSV_t$ - $NPSV_t$-sel - NQP - NSPACE(f(n)) - NT - NTIME(f(n)) - OCQ - OptP - P - P/log - P/poly - $P^{\#P}$ - $P^{\#P[1]}$ - PAC^0 - PBP - k-PBP - P_C - P^{cc} - PCD(r(n),q(n)) - P-close - PCP(r(n),q(n)) - PermUP - PEXP - PF - PFCHK(t(n)) - PH - PH^{cc} - $Φ_2P$ - PhP - $Π_2P$ - PINC - PIO - P^K - PKC - PL - PL_1 - $PL_{infinity}$ - PLF - PLL - PLS - P^{NP} - $P^{NP[k]}$ - $P^{NP[log]}$ - P-OBDD - PODN - polyL - PostBQP - PP - PP/poly - PPA - PPAD - PPADS - PPP - P^{PP} - PPSPACE - PQUERY - PR - P_R - $Pr_HSPACE(f(n))$ - PromiseBPP - PromiseBQP - PromiseP - PromiseRP - PrSPACE(f(n)) - P-Sel - PSK - PSPACE - PT_1 - PTAPE - PTAS - PT/WK(f(n),g(n)) - PZK - QAC^0 - $QAC^0[m]$ - $QACC^0$ - QAM - QCFL - QCMA - QH - QIP - QIP(2) - QMA - QMA+ - QMA(2) - QMA_{log} - QMAM - QMIP - $QMIP_{le}$ - $QMIP_{ne}$ - QNC^0 - QNC_f^0 - QNC^1 - QP - QPSPACE - QSZK - R - RE - REG - RevSPACE(f(n)) - R_HL - RL - RNC - RP - RPP - RSPACE(f(n)) - S_2P - S_2-EXP•P^{NP} - SAC - SAC^0 - SAC^1 - SAPTIME - SBP - SC - SEH - SelfNP - SF_k - $Σ_2P$ - SKC - SL - SLICEWISE PSPACE - SNP - SO-E - SP - SP - span-P - SPARSE - SPL - SPP - SUBEXP - symP - SZK - SZK_h - TALLY - TC^0 - TFNP - $Θ_2P$ - TreeBQP - TREE-REGULAR - UAP - UCC - UE - UL - UL/poly - UP - US - VNC_k - VNP_k - VP_k - VQP_k - W[1] - WAPP - W[P] - WPP - W[SAT] - W[*] - W[t] - $W^*[t]$ - XOR-MIP*[2,1] - XP - $XP_{uniform}$ - YACC - ZPE - ZPP - ZPTIME(f(n))

Proceedings of the SDR '08 Technical Conference and Product Exposition.

The Complexity Zoo



The Complexity Zoo: http://www.complexityzoo.com/#nexp
Zookeeper: Scott Aaronson
407 classes and counting ...

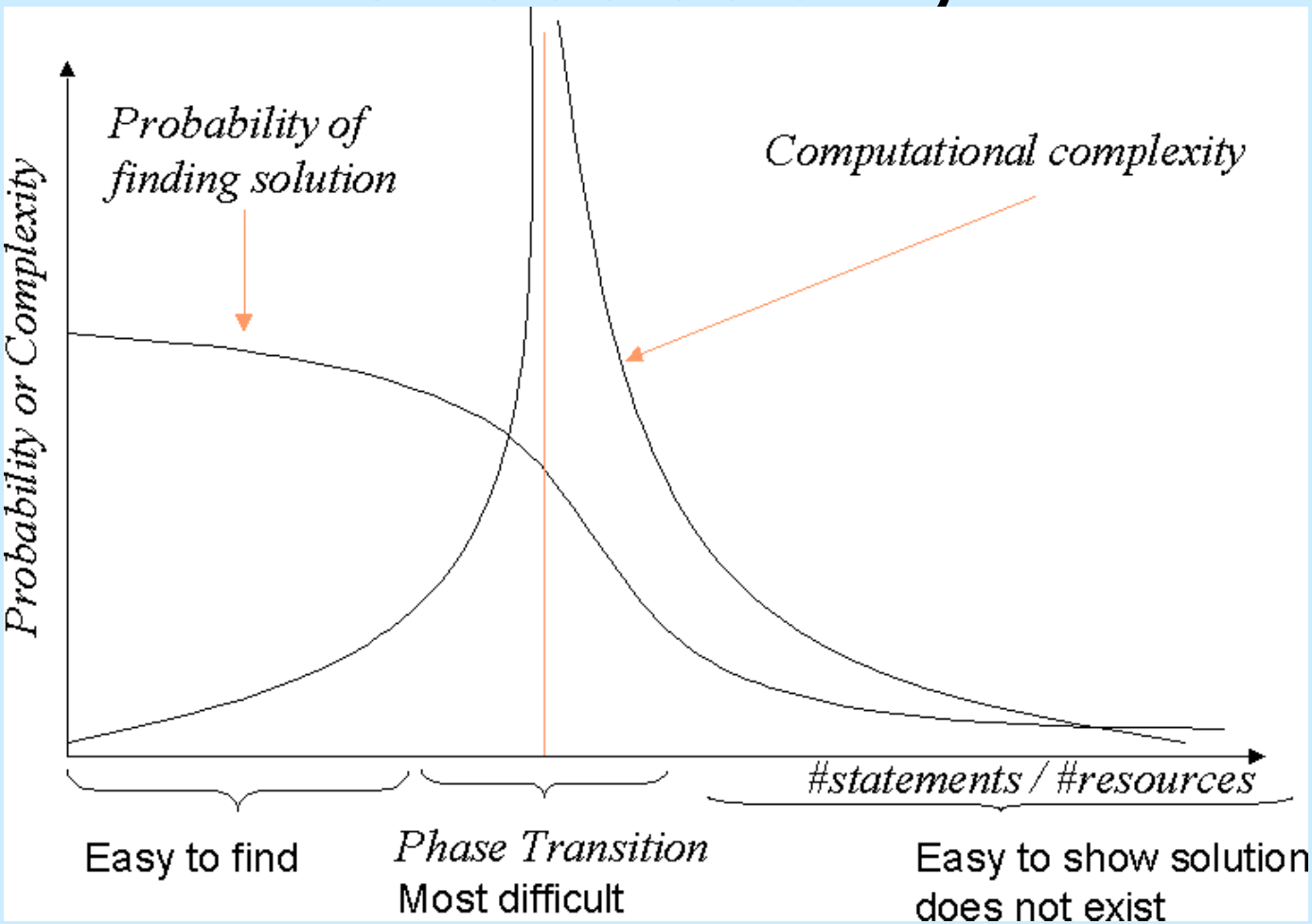
Proceedings of the SDR '08 Technical Conference and Product Exposition.

OWL in the Zoo

Language	***Logic***	***Complexity***
OWL Lite	*SHOIN(D)*	EXPTIME
OWL DL	*SHIF(D)*	NEXPTIME
OWL Full	Subset of FOL	Undecidable
DL+SWRL	Subset of FOL	Undecidable

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Phase Transitions and Undecidability



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Summary of Languages

Language	Expressivity	Relevance to CR Domain Advantages/Limitations
OWL	Classes, individuals, binary relations	Composition can be described only at the individuals level
OWL plus Rules	Composition of relations	Compositions can be described at the class level
CL	Functions, limited quantification	Description of functionality; limited function inference capability
Metaslang	Functions, full quantification, colimits	Equivalence of functionality; refinement; abstract specifications; composition of specifications
LTL	Temporal aspects of the system	Temporal aspects (time delays, etc.)
Accord	Behavioral descriptions, system's state	Dynamic behaviors (state and temporal aspects)

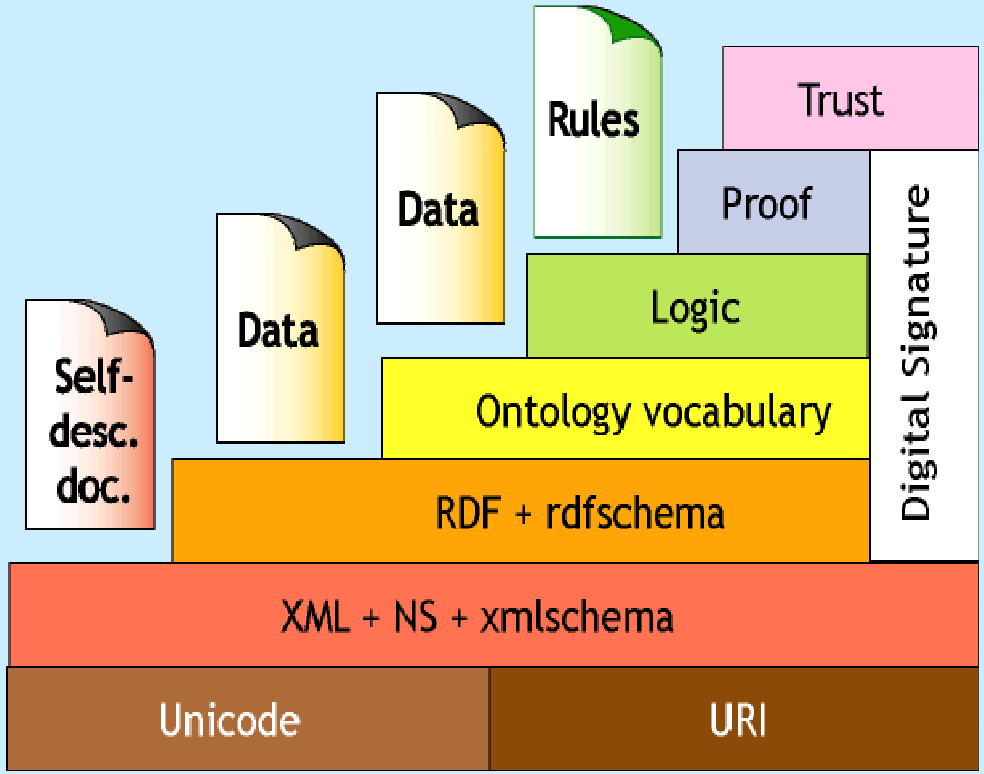
Proceedings of the SDR '08 Technical Conference and Product Exposition.

Approach: Logic

- Treat potential objects and relations as theories of the world
- Theories are the subject of logic (well defined and understood)
- They describe possibilities or potentiality (e.g., the fact that we know a theory of “car”, doesn’t mean there’s a car in this room)
- Add collected data to the theory - theory specialization
- Formulate any queries – in formal logic these are conjectures
- Use a general purpose theorem prover to prove conjectures
- Use the trace of the proof as an answer to the query
- Objects can be complex (compositions of simpler objects)
- Ideally, theories for complex situations are compositions of simple theories
- Combining theories using colimit of category theory

Proceedings of the SDR '08 Technical Conference and Product Exposition.

The “Layer Cake” (Tim Berners-Lee)



Proceedings of the SDR '08 Technical Conference and Product Exposition.

Language Constructs: OWL Lite

- *Class*
- *rdf:Property*
- *rdfs:subClassOf*
- *rdfs:subPropertyOf*
- *rdfs:domain*
- *rdfs:range*
- *Individual*
- *equivalentClass*
- *equivalentProperty*
- *sameAs*
- *differentFrom*
- *allDifferent*
- *inverseOf*
- *TransitiveProperty*
- *SymmetricProperty*
- *FunctionalProperty*
- *InverseFunctionalProperty*
- *allValuesFrom*
- *someValuesFrom*
- *minCardinality (only 0 or 1)*
- *maxCardinality (only 0 or 1)*
- *cardinality (only 0 or 1)*
- *intersectionOf*
- *Imports*
- *priorVersion*
- *…more*

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Language Constructs: DL & Full

- *one of*
- *disjointWith*
- *equivalentClass*
  (applied to class expressions)
- *rdfs:subClassOf*
  (applied to class expressions)
- *unionOf*
- *intersectionOf*
- *complementOf*

Arbitrary Cardinality

- *minCardinality*
- *maxCardinality*
- *cardinality*

- *hasValue*

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Differences

- OWL Lite
  - simplest; simple constraints
  - but easy to develop tools
- OWL DL (for Description Logics)
  - more complex and expressive
  - still decidable
- OWL Full
  - expressive
  - no computational guarantees
- But for Cognitive Radio we need Rules, Functions and Behaviors!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Examples of reasoning

- Inferring classifications (if subClass is not stated explicitly, yet necessary and sufficient conditions exist)
  - Relevance to fusion: classification
  - Query: Is Air Base a subclass of Repair Facility?
- Infer “type” relation: Given an individual, which classes is it member of?
  - Relevance to fusion: object classification, situation types
  - Example: given features of a flying object, determine it’s an F16, and consequently an aircraft
  - Example: given info on parts in stock and demand, determine whether this is the case of “nominal”, “marginal” or “critical” situation type
- Inferring identity – is this individual sameAs another?
  - Relevance to fusion: association
  - Infer that two reports about an aircraft are about the same one
- Inferring relations (properties) from ontology
  - Relevance to fusion: situation awareness
  - This aircraft belongs to the same squadron as another
- Inferring relations from rules
  - Infer that a specific part at airbase for aircraft is critical, marginal, nominal

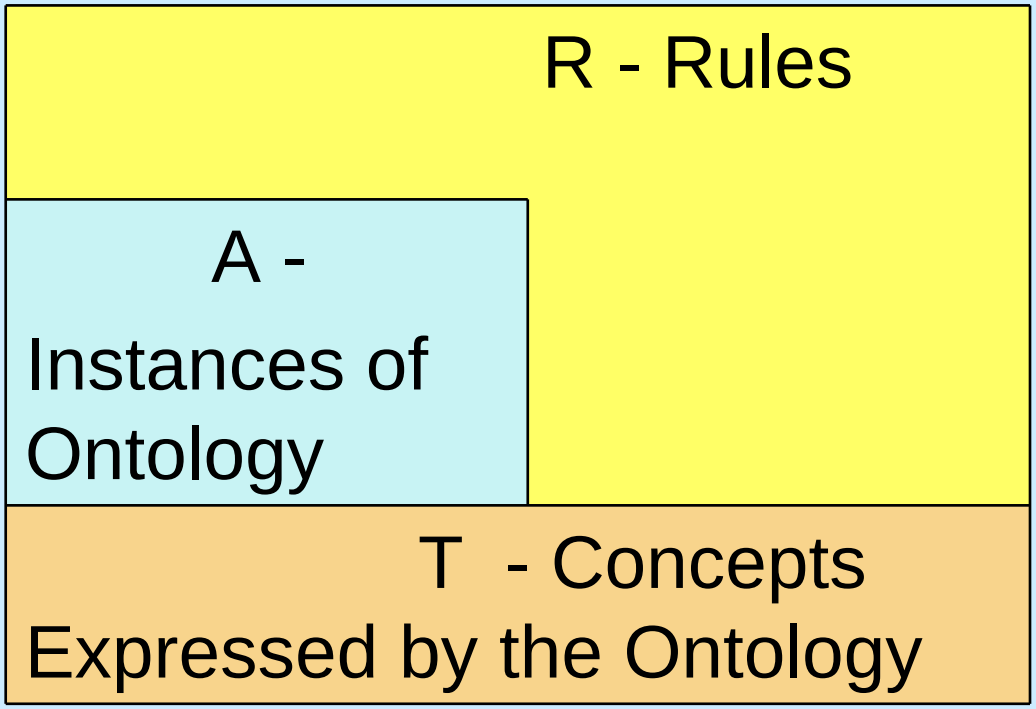
Proceedings of the SDR '08 Technical Conference and Product Exposition.

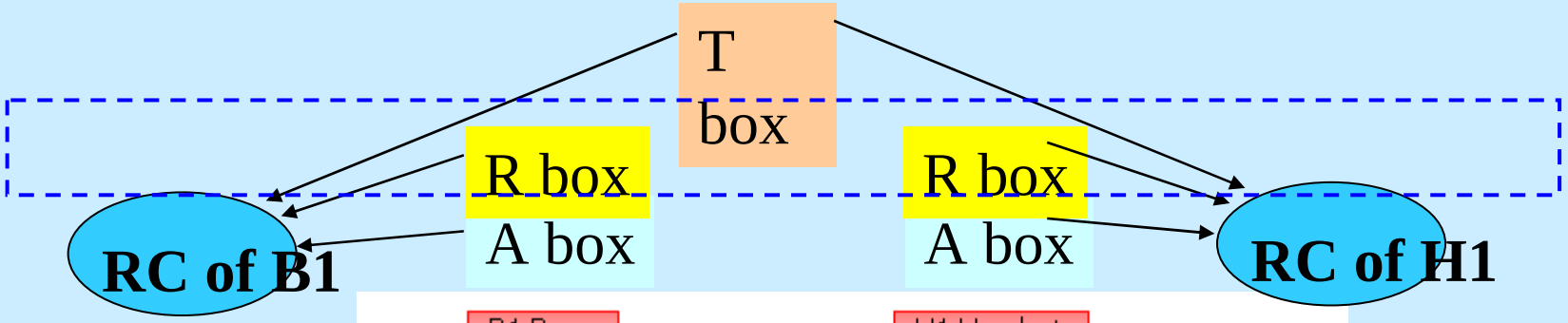
OWL + Rules

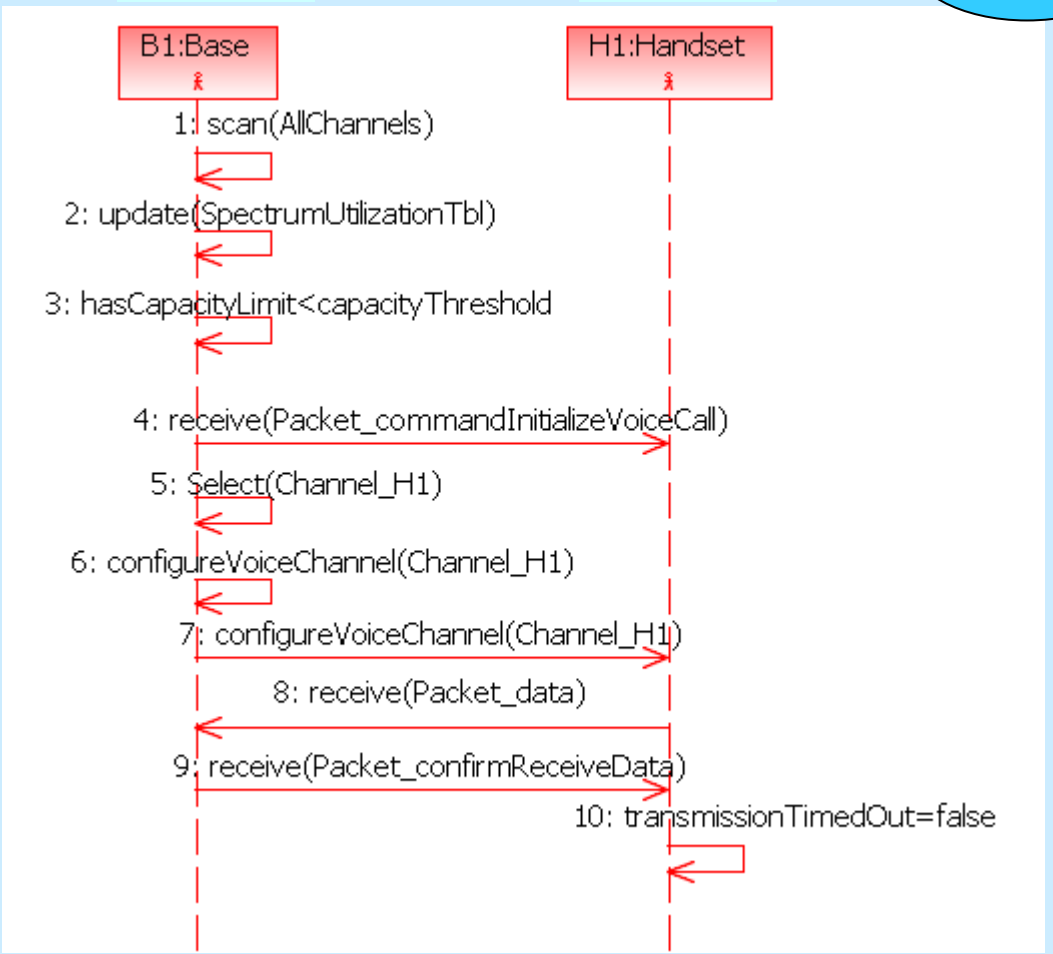
R Box: policies (rules)

A Box: instances of the ontology

T Box: concepts defined in the ontology



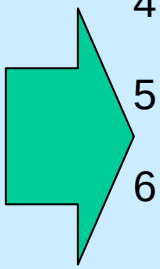




Policies (Rachel Li)

Functional Capabilities

- Determining which radios are connected from the base
- Finding and identifying peer radios;
- Identifying and authenticating compatible reconfigurable radios;
- Forming a satisfactory network extension route to the infrastructure from each affected radio using non-interfering frequencies for each “hop”;
- Adjusting the network topology as responders arrive and depart from the area where coverage is unavailable;
- Preserving the level of security of the baseline network in the network extensions;
- Providing either full duplex (simultaneous receive and transmit) operation or including a “store and forward” capability for user voice and/or data communications.



Rules include:

1. check the signal strength to determine the connectivity status;
2. check whether the received packet is destined to itself;
3. query of neighbor’s information when a radio is disconnected from the base;
4. send back an answer message when a radio receives a query from others;
5. process the answer message and update the routing table;
6. send a route reply message traversing back along the desired path to the starting hop when a radio finds a path to the base;
7. store and forward the data packet to the next hop;
8. send back an end-to-end acknowledgement when the data packet arrives at the destination.

13 rules implemented for the Network Extension Use Case

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Summary

- If you expect that your CR will need to modify its behavior in run time, the declarative language approach seems to be the way to go
- Decide which language you need
  - As simple as possible, but not simpler than that!
- If not expecting any change/flexibility of the application, don't use the O-B approach
- Be prepared to deal with lots of detail in the implementation of policies (rules)
- Be prepared to deal with the issue of computational complexity

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Summary

- Challenge: Develop “universals” (ontology) for the radio/networking domain
- Join the Internet (business model and services)
- A community-wide coordinated effort needed
- SDR Forum seems to be the right community for this task
- MLM Work Group has already started the effort
- Advantages to all players – manufacturers, ISPs, operators and USERS!

Proceedings of the SDR '08 Technical Conference and Product Exposition.

Language Standardization Efforts

- MLM Working Group of the Software Defined Radio Forum (SDRF)
- IEEE SCC41 P1900.5: Policy Language and Architectures for Managing Cognitive Radio for Dynamic Spectrum Access Applications Working Group: http://grouper.ieee.org/groups/scc41/5/index.htm
- DARPA and performers on DTN, WANN and WNaN programs
- E3
- IEEE 802.21
- VITA 49
- Regulators (expected participation)
- Others Welcome!

Proceedings of the SDR '08 Technical Conference and Product Exposition.