Applications of Self-Awareness, Situation Awareness and Feedback Control

Ken Baclawski
College of Computer and Information Science
Northeastern University

Outline

- Cognitive Systems
  - Cognitive Loop
  - System Functionality
- Situation Awareness
  - Formalization
  - Situation Awareness Assistant
- Self-Awareness
  - Functionality
  - Software composition
- Feedback Control
  - Models
  - Stability

Application Domains

- Supply Logistics
- Asset Repair
- Radio Communication
- Data Link Network Communication
- Denial of Service Defense
- Electromagnetic Surveillance
- Non-Preemptive Real-Time Task Scheduling

Outline

- Cognitive Systems
  - Cognitive Loop
  - System Functionality
- Situation Awareness
  - Formalization
  - Situation Awareness Assistant
- Self-Awareness
  - Functionality
  - Software composition
- Feedback Control
  - Models
  - Stability

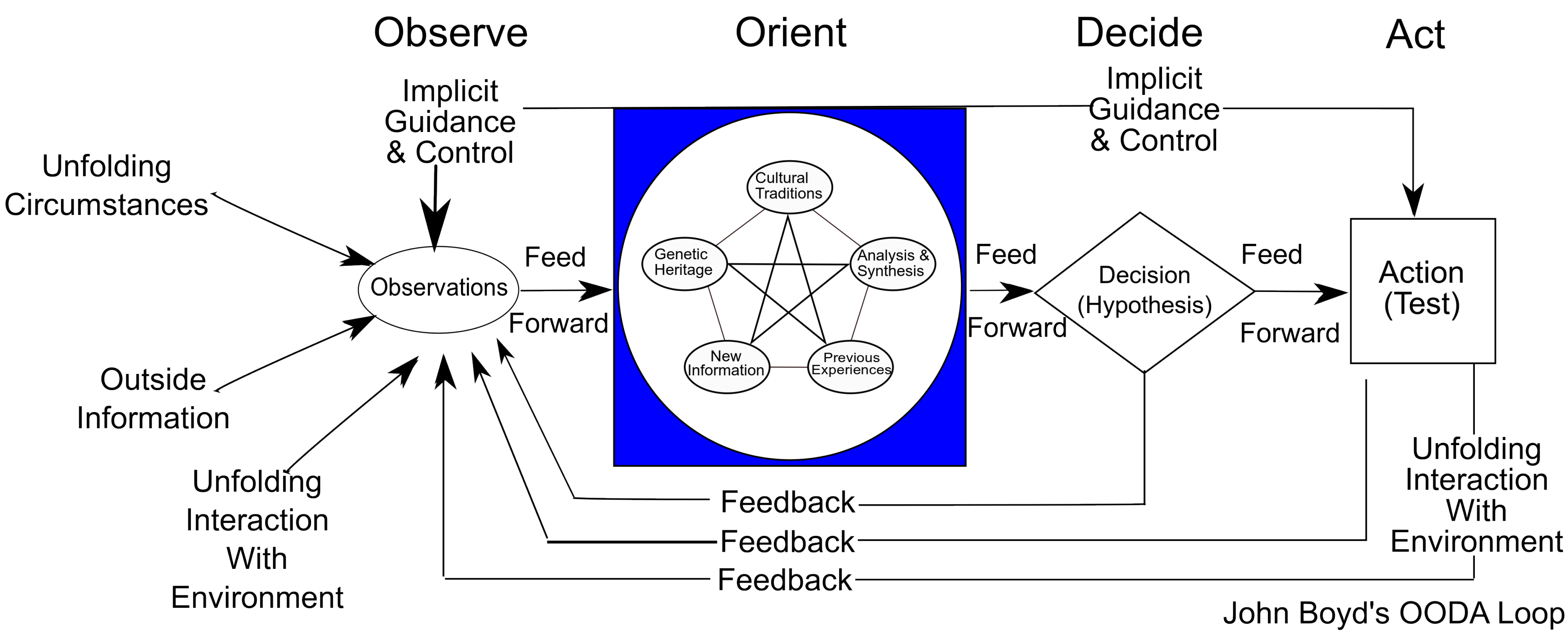
Cognitive System

- Can reason, using substantial amounts of appropriately represented knowledge
- Can learn from its experience so that it performs better tomorrow than it did today
- Can explain itself and be told what to do
- Can be aware of its own capabilities and reflect on its own behavior
- Can respond robustly to surprise.

Cognitive System

- Often viewed according to Boyd's OODA (Observe, Orient, Decide, Act) loop.
- Also presented in the less precise perception–reasoning–action triad.
- The cognitive loop is fundamental to many systems.
  - Often regarded as the most important problem of artificial intelligence research.

OODA Loop



Observe
Orient
Decide
Act
Implicit Guidance & Control
Implicit Guidance & Control
Unfolding Circumstances
Observations
Outside Information
Unfolding Interaction With Environment
Feed Forward
Cultural Traditions
Genetic Heritage
Analysis & Synthesis
New Information
Previous Experiences
Feed Forward
Decision (Hypothesis)
Feed Forward
Action (Test)
Unfolding Interaction With Environment
Feedback
Feedback
Feedback
John Boyd's OODA Loop

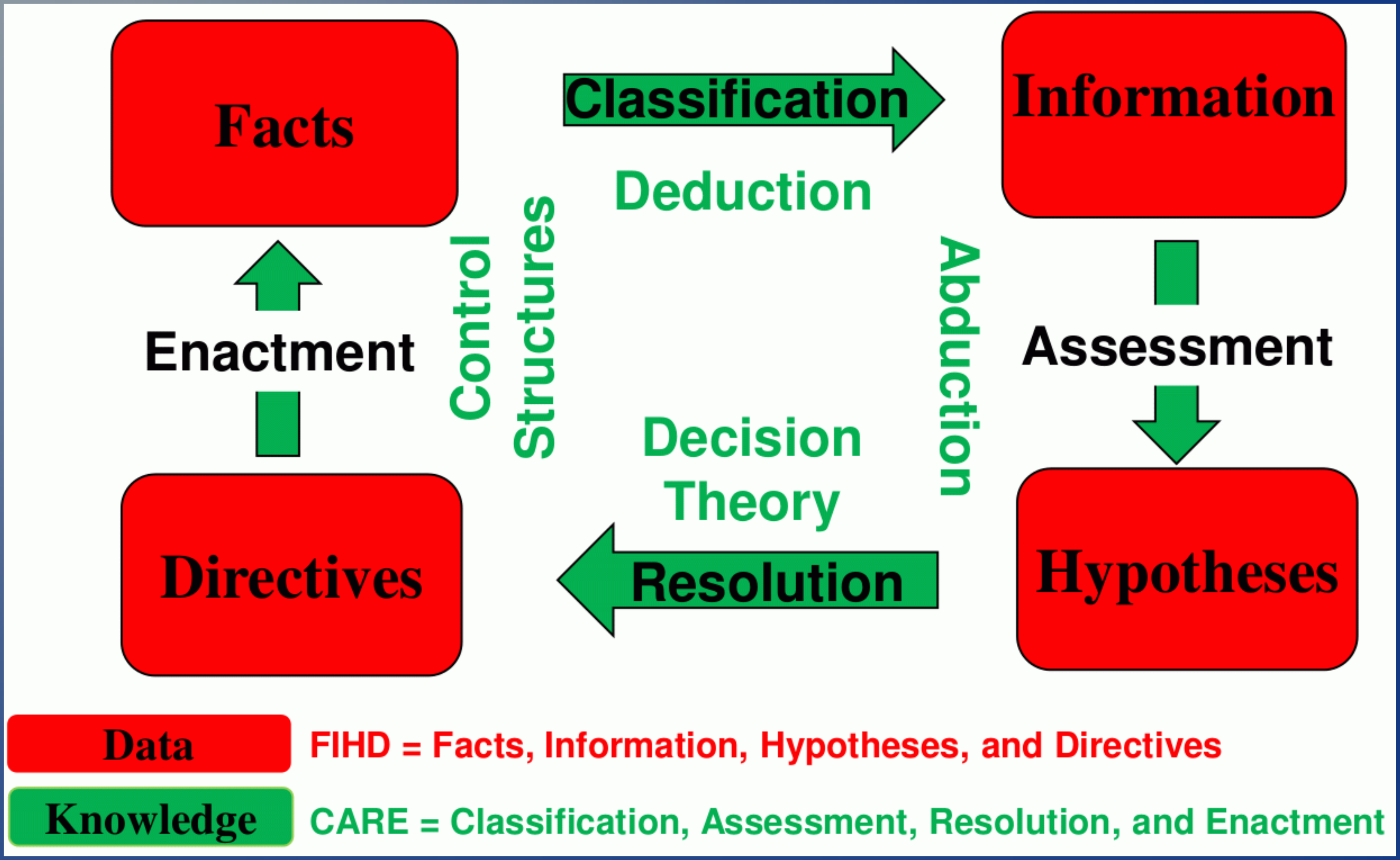
OODA Loop

- Observe, Orient, Decide, and Act (OODA) Loop
  - Observe the entities and environment,
  - Orient the participant to the observations, by cultural tradition, generic heritage, previous experience, analysis and synthesis, new information
  - Decide on the directives based on the hypotheses that best explains the observations, and
  - Act on the directives to interact with the entities and environment, to test the hypothesis
- Developed by a fighter pilot: Colonel John Boyd
  - Now an important concept in litigation, business and military strategy

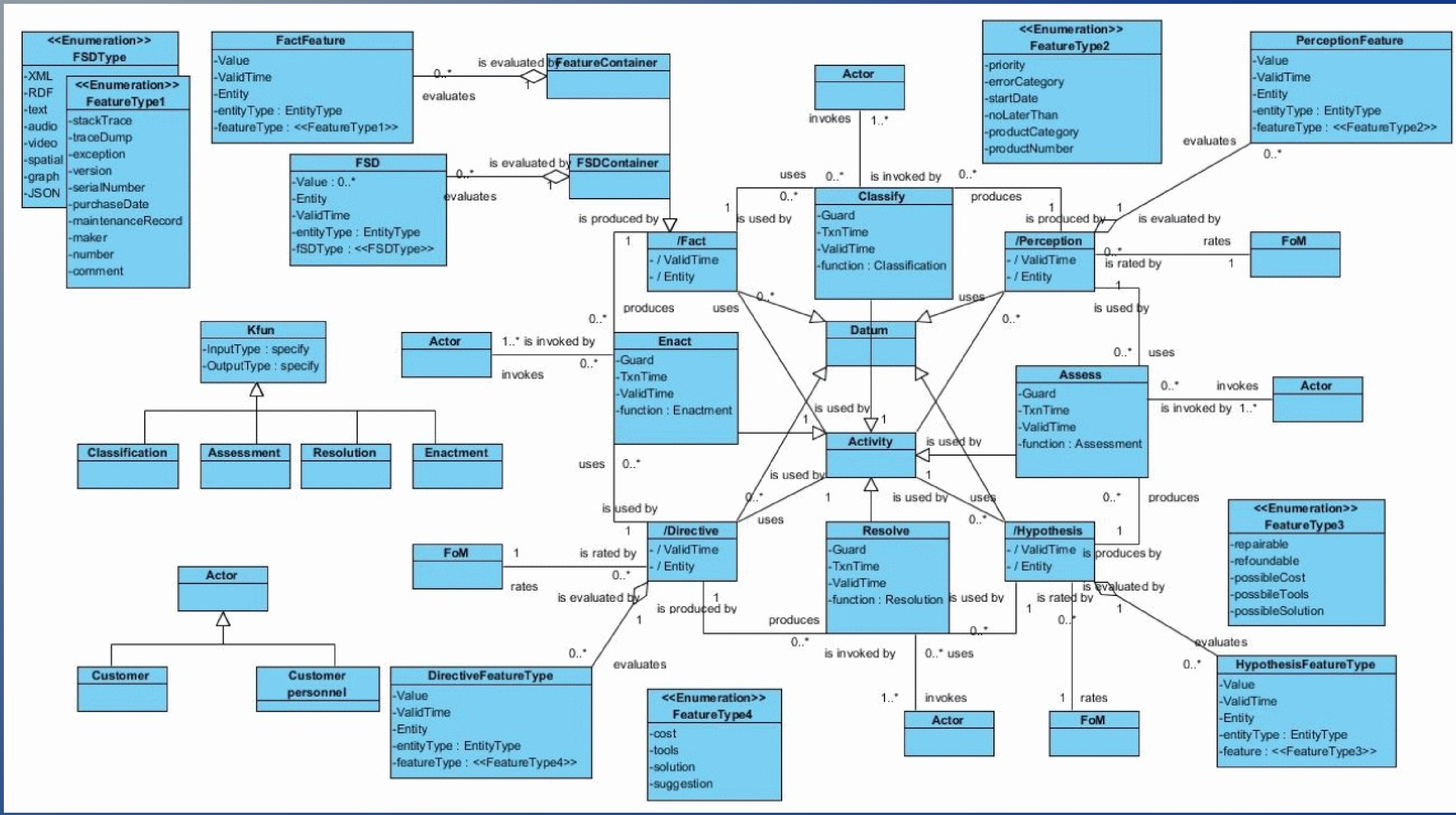
KIDS

- Knowledge Intensive Data-Processing System
  - Developed by Dieter Gawlick, Adel Ghoneimy, Zhen Hua Liu, Eric Chan, and others at Oracle
  - Formalization of the OODA Loop
  - Designed to be customized with a domain ontology and transformation rules
- References
  - www.cidrdb.org/cidr2015/Papers/15_Abstract43GD.pdf
  - Enabling Enhanced OODA Loop with Modern Information Technology, http://ontolog.cim3.net/cgi-bin/wiki.pl?ConferenceCall_2014_02_13#nid468H
  - Situation Aware Computing for Big Data, Workshop on Semantics for Big Data on the Internet of Things (SemBIoT 2014), 2014 IEEE International Conference on Big Data, Oct 27-30, Washington DC.

The KIDS FIHD/CARE Loop



KIDS Ontology



Cognitive System Functionality

- Observe
  - Information collection and fusion
- Orient
  - **Situation awareness**
  - **Self-awareness**
- Decide
  - Awareness of constraints and requirements
  - Flexible rule and query capability

Cognitive System Functionality

- Act
  - Command execution
- Interact
  - Dynamic interoperability at any layer
  - Negotiation for resources
- Control
  - **Controller for robustness and stability**

Outline

- Cognitive Systems
  - Cognitive Loop
  - System Functionality
- Situation Awareness
  - Formalization
  - Situation Awareness Assistant
- Self-Awareness
  - Functionality
  - Software composition
- Feedback Control
  - Models
  - Stability

Situation Awareness

- Situation Awareness (SAW) (Endsley)
  - The perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future
- Situation Assessment
  - A process that estimates and updates that state (belief revision)

Formalization of Situation

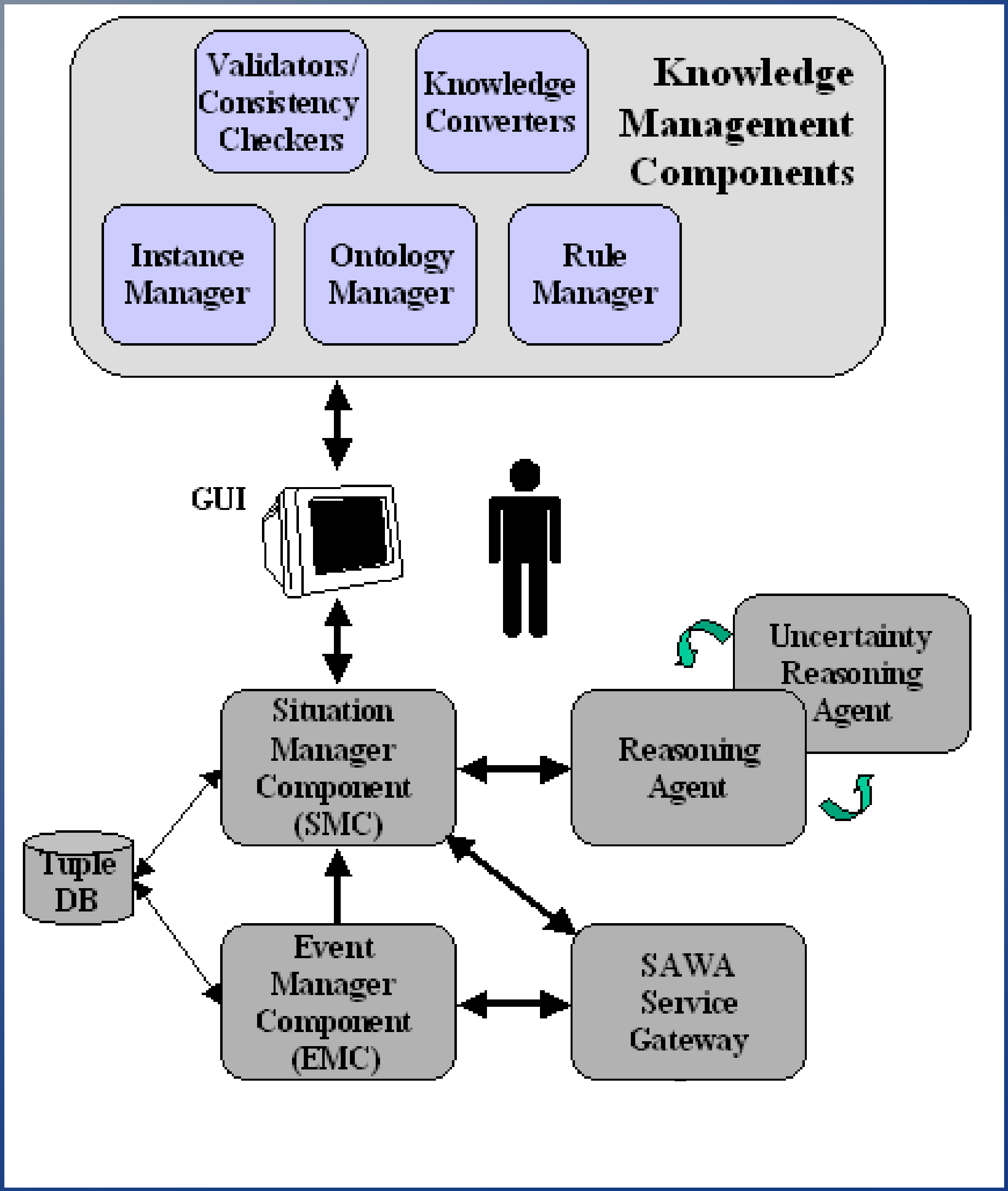
- Situation Theory of Barwise (1987) and Devlin (1991)
  - Situations are objects that can be part of other situations.
  - The fundamental notion is the *infon:*

    If L is a location in a situation s, then L is of type LOC,

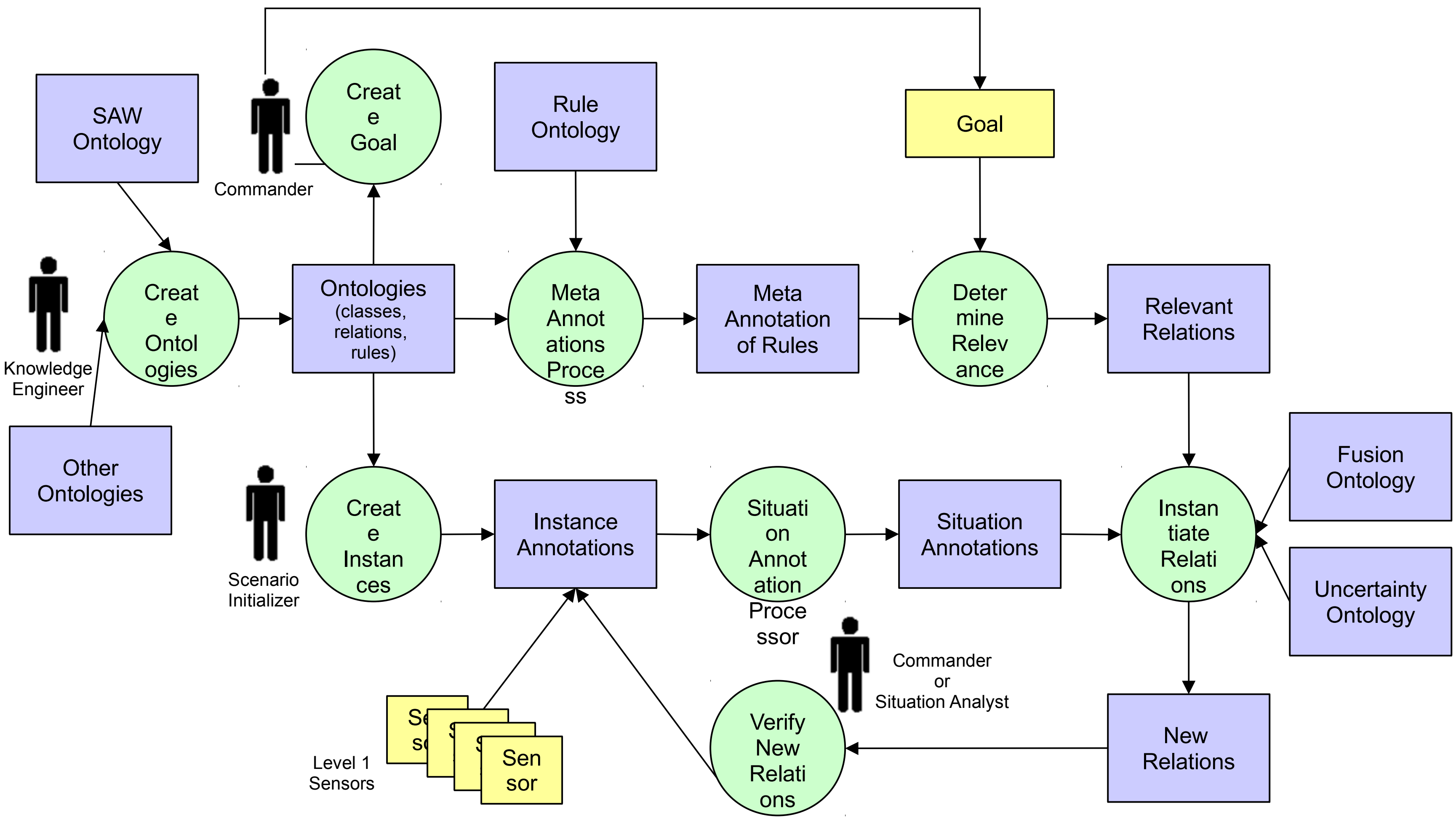
    the infon is <<of-type, L, LOC, 1>>

    and one has s $\models$ <<of-type, L, LOC, 1>>
- Formulated as the Situation Theory Ontology using OWL in Matheus, Kokar, Baclawski (2003)
  - See www.ccs.neu.edu/home/kenb/STO.owl
  - Inference is implemented with rules which can be logical or probabilistic.
  - Prototype Situation Awareness Assistant implemented STO.

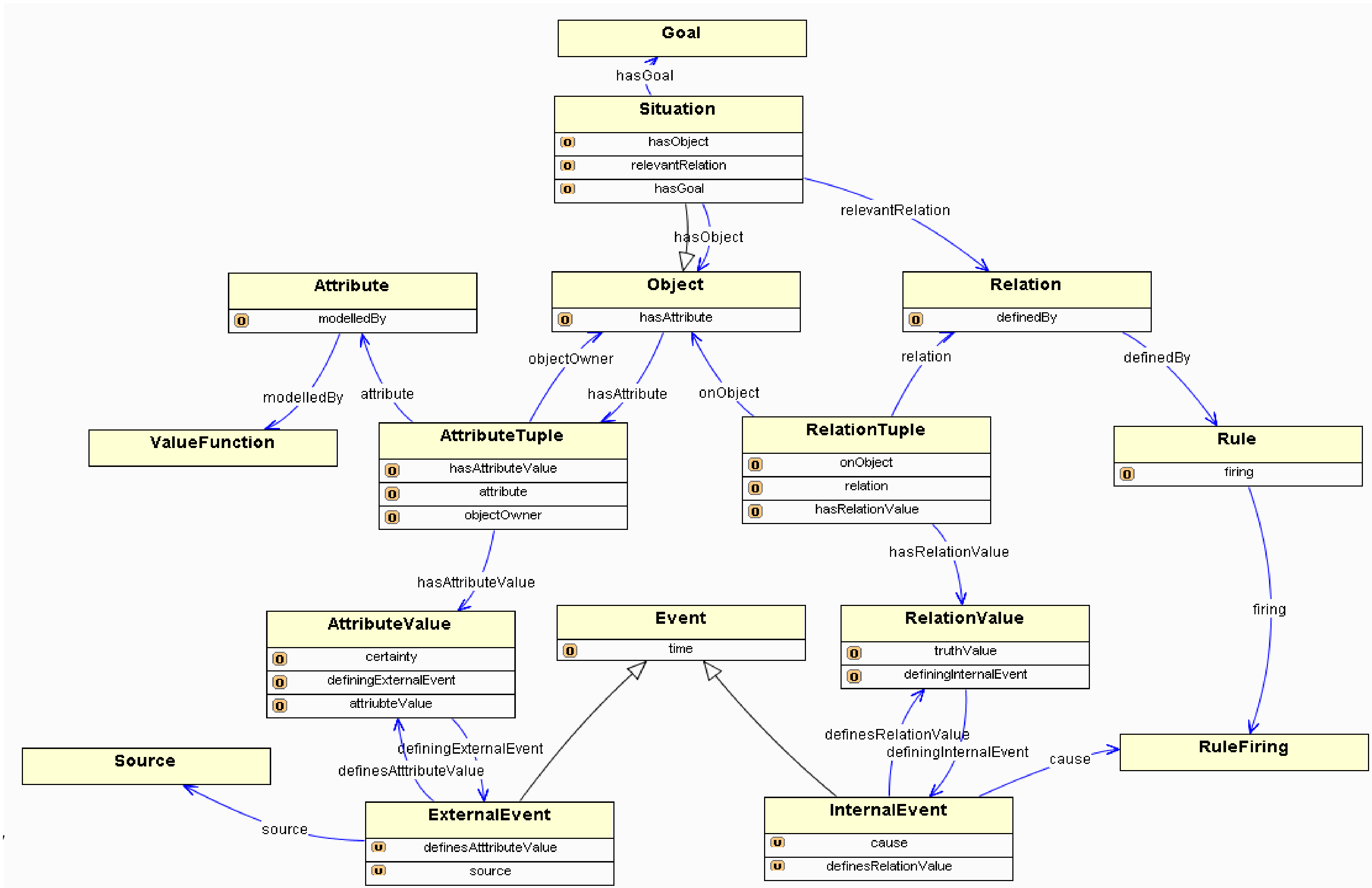
Situation Awareness Assistant (SAWA)



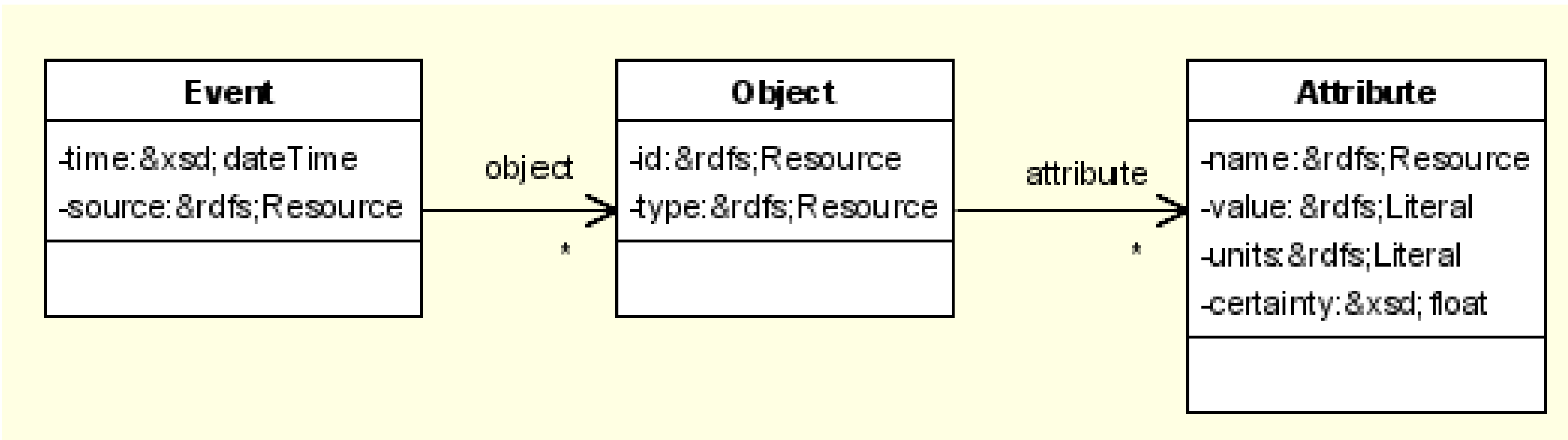
Situation Awareness Process

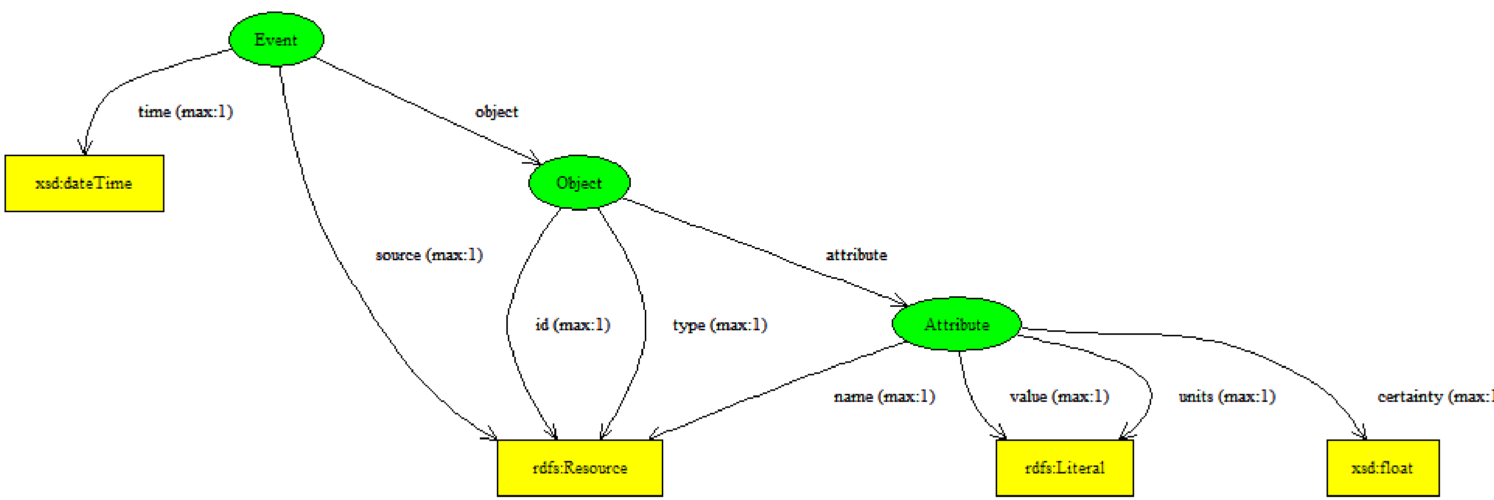


Situation Awareness Core Ontology

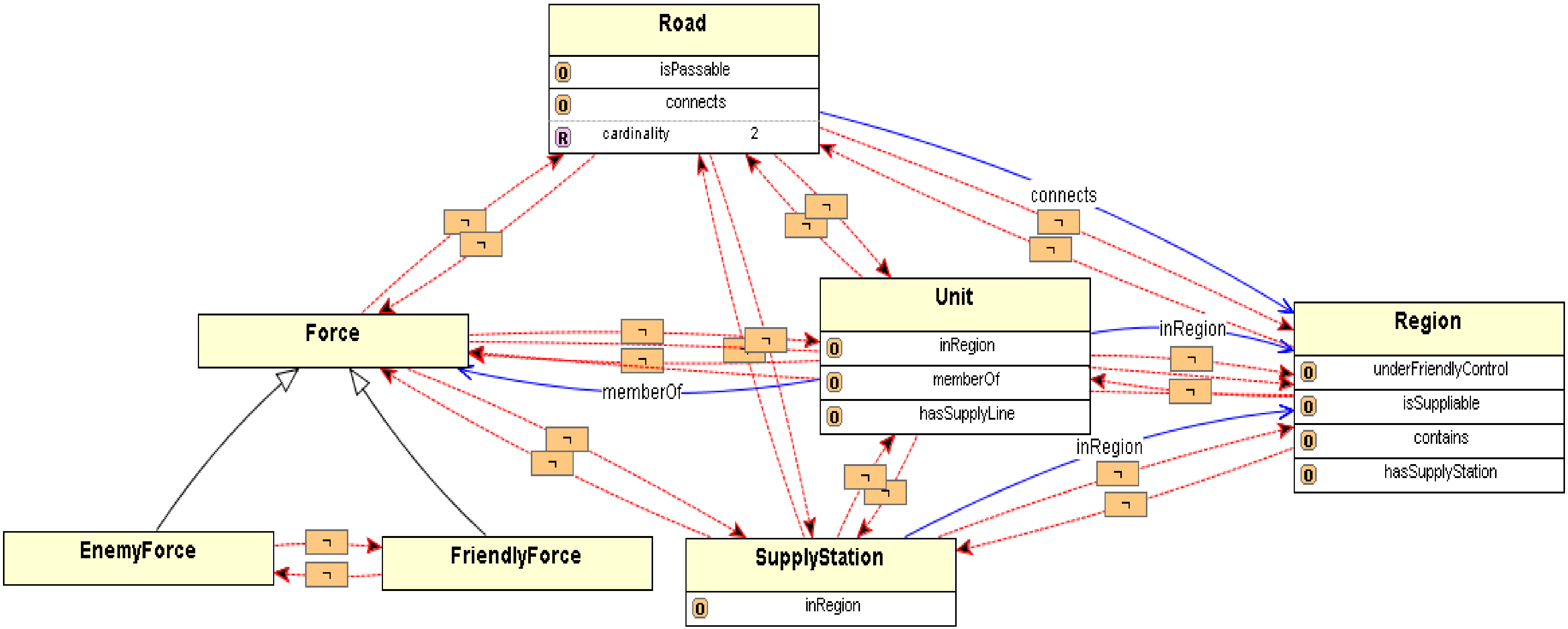


Event Ontology





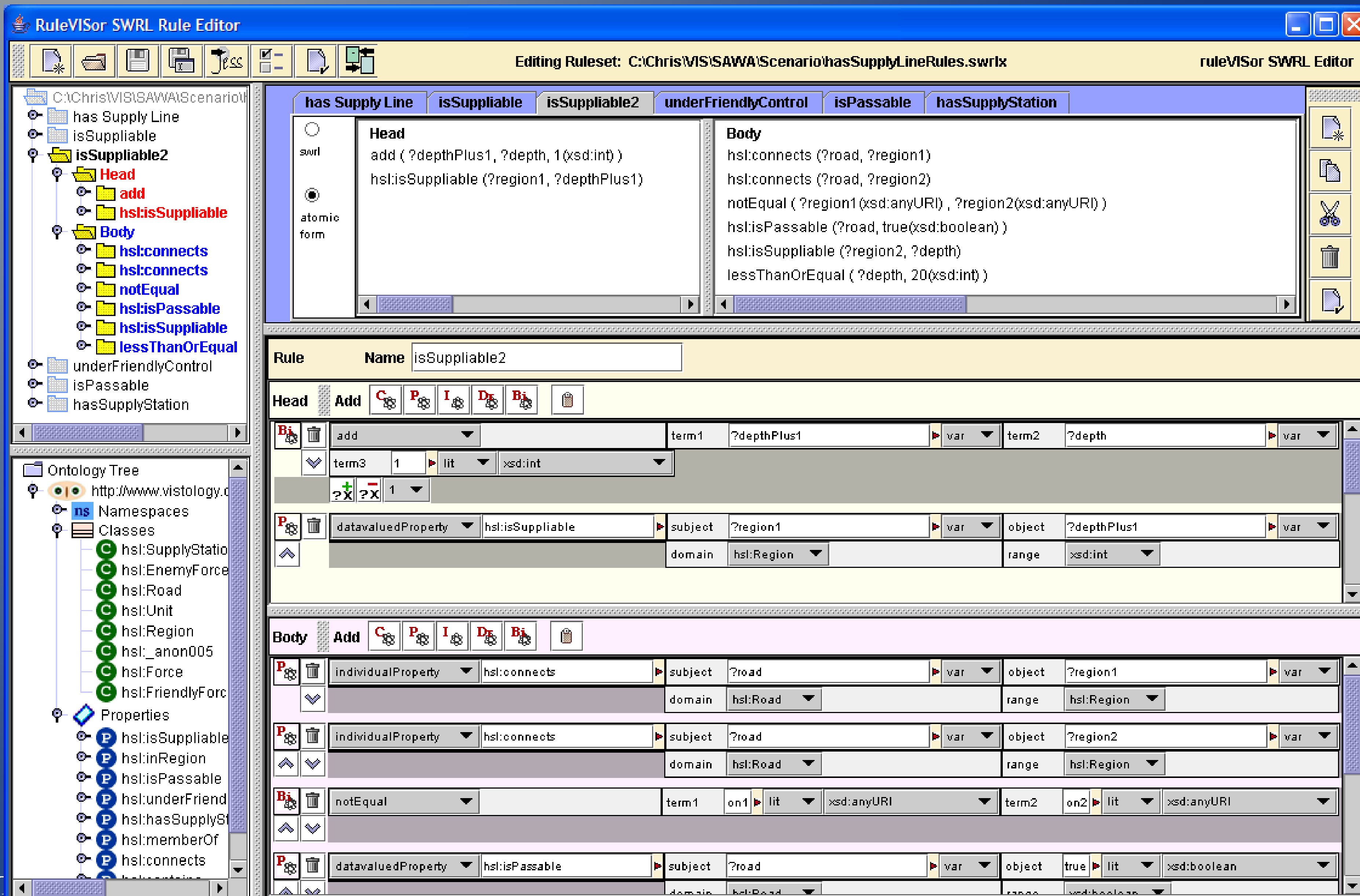
Supply Logistics Ontology



RuleVISor Rule Editor

- Graphical SWRL Editor
- Support for
  - all RuleML capabilities (everything in SWRL from ruleml: namespace)
  - all new SWRL elements (from swrlx: namespace, e.g., swrlx:builtin)
- Does not support arbitrary embedded OWL constructs
  - OWL Ontologies are required to be external
- Ontologies used as basis for rule building blocks

RuleVISor GUI



Supply Logistics Rule Set

```
<rule rlab="has Supply Line">
   <body>
     <hsl:inRegion     sub="?unit"     data="?region"/>
     <hsl:isSuppliable sub="?region"   data="true"/>
   </body>
   <head>
     <hsl:hasSupplyLine        sub="?unit"     data="true"/>
   </head>
 </rule>

 <rule rlab="isSuppliable">
   <body>
     <hsl:hasSupplyStation     sub="?region"   data="true"/>
     <hsl:underFriendlyControl sub="?region"   data="true"/>
   </body>
   <head>
     <hsl:isSuppliable sub="?region"   data="true"/>
   </head>
 </rule>

 <rule rlab="isSuppliable2">
   <body>
     <hsl:connects     sub="?road"     data="?region1"/>
     <hsl:connects     sub="?road"     data="?region2"/>
     <swrlb:notEqual
       arg1="?region1"
       arg2="?region2"/>
     <hsl:isPassable   sub="?road"     data="true"/>
     <hsl:isSuppliable sub="?region2"  data="true"/>
   </body>
   <head>
     <hsl:isSuppliable sub="?region1"  data="true"/>
   </head>
 </rule>

<rule rlab="underFriendlyControl">
  <body>
    <hsl:inRegion     sub="?unit"     data="?region"/>
    <hsl:memberOf     sub="?unit"     data="?force"/>
    <hsl:FriendlyForce ind="?force"/>
  </body>
  <head>
    <hsl:underFriendlyControl sub="?region"   data="true"/>
  </head>
</rule>

<rule rlab="isPassable">
  <body>
    <hsl:connects     sub="?road"     data="?regionA"/>
    <hsl:connects     sub="?road"     data="?regionB"/>
    <swrlb:notEqual
      arg1="?regionA"
      arg2="?regionB"/>
    <hsl:underFriendlyControl sub="?regionA"  data="?force1"/>
    <hsl:underFriendlyControl sub="?regionB"  data="?force2"/>
  </body>
  <head>
    <hsl:isPassable   sub="?road"     data="true"/>
  </head>
</rule>

<rule rlab="hasSupplyStation">
  <body>
    <hsl:inRegion     sub="?X"        data="?region"/>
    <hsl:SupplyStation ind="?X"/>
  </body>
  <head>
    <hsl:hasSupplyStation     sub="?region"   data="true"/>
  </head>
</rule>
```

BaseVISor Rule Engine

- Forward chaining, Rete-based rule engine
- Native support for RDF triples
- Support for recursive What-If scenarios
- Support for uncertainty propagation using Bayesian networks
- High performance
  - Next slide compares BaseVISor with Jess
- Implemented in Java

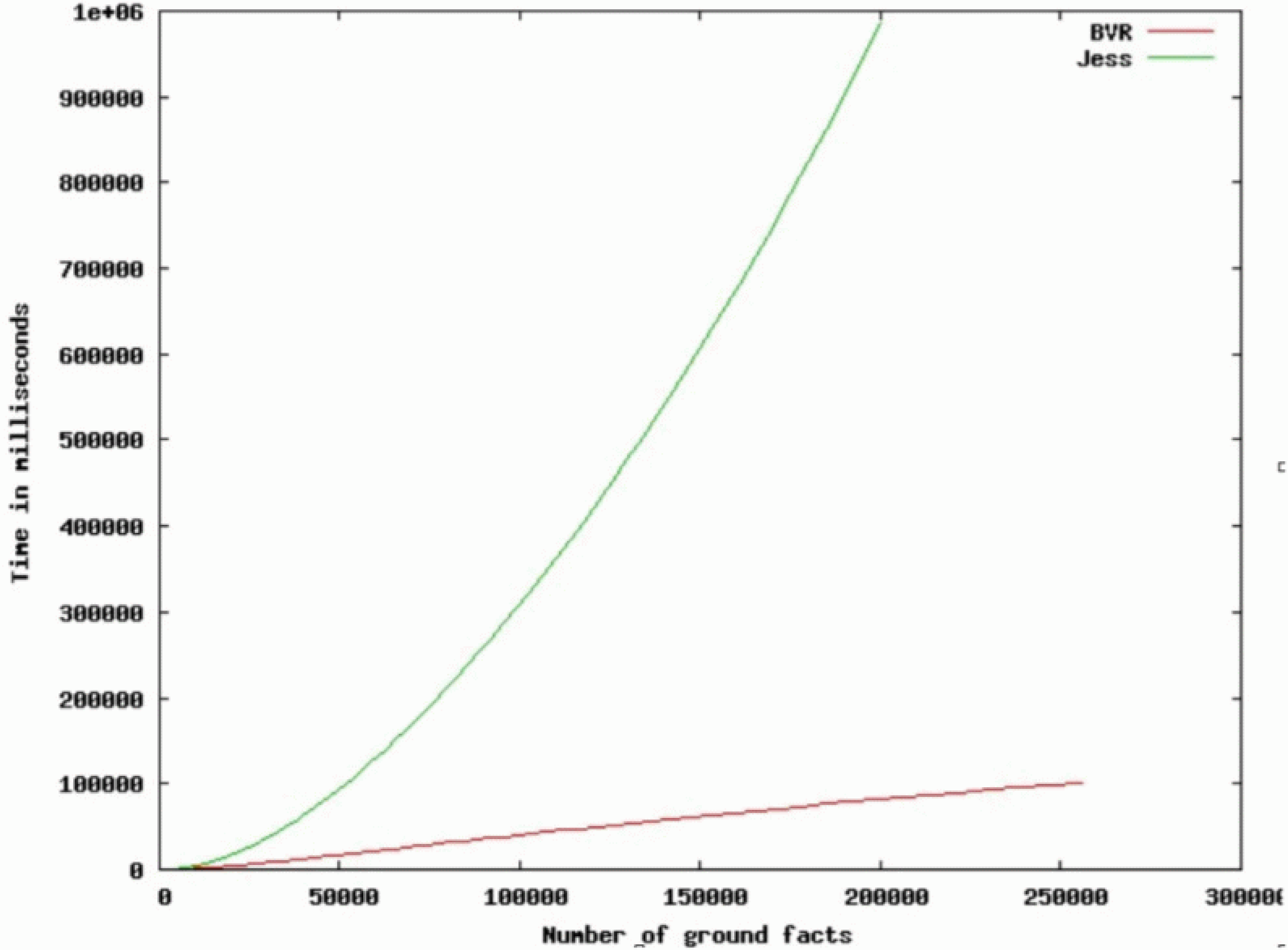
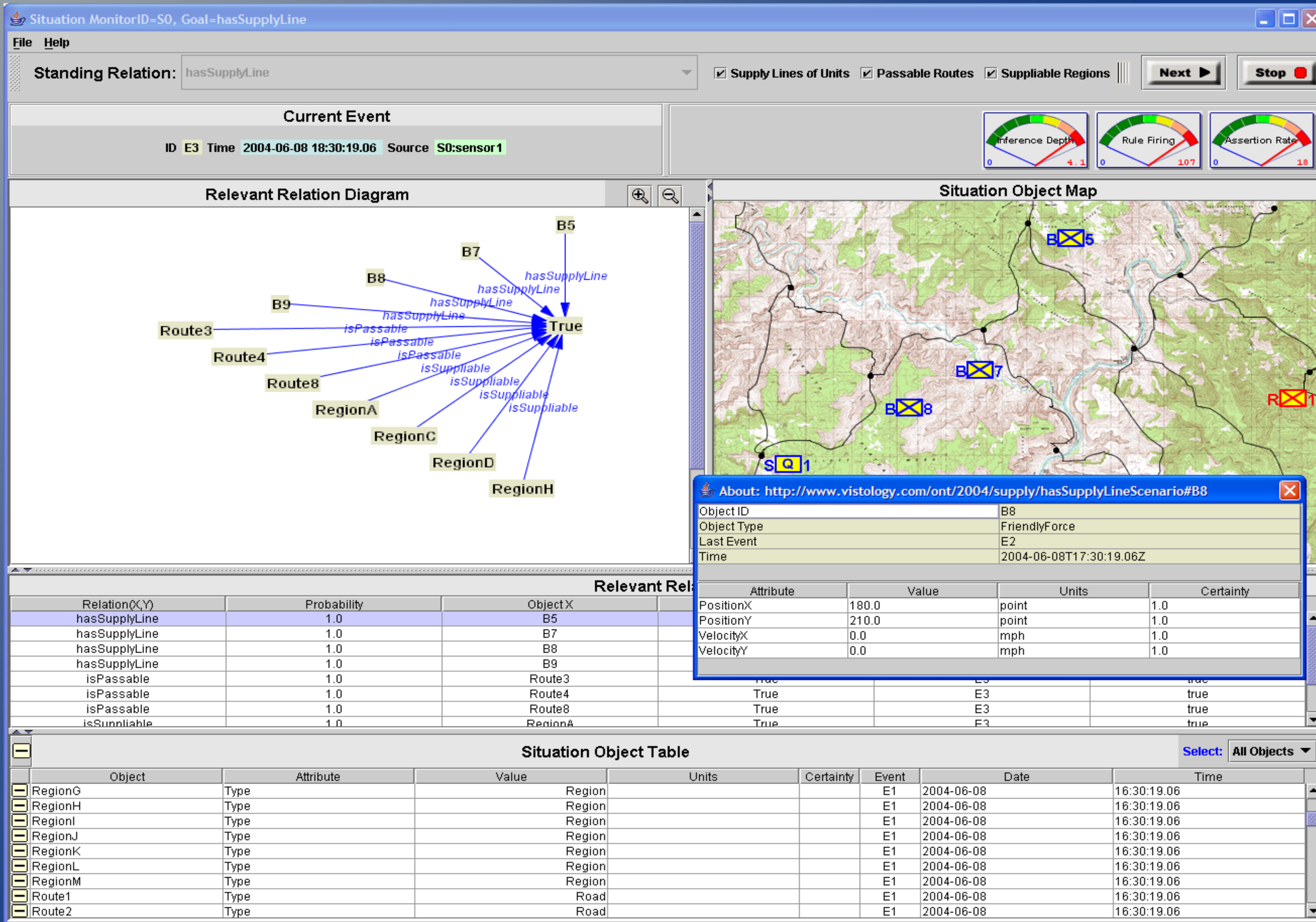


Figure 5. BaseVISor vs. Jess performance.

SAWA Runtime GUI



SAWA Accomplishments

- SAWA is a general purpose assistant for situation awareness:
  - monitors the evolution of relevant higher-order relations within a situation.
  - supports formal reasoning techniques for level-2 fusion.
  - based on the Semantic Web languages OWL and SWRL.
  - performs relevance reasoning.
- The domain ontology and rules are constructed and checked using an ontology editor, rule editor and consistency checker.
- At runtime events are processed to determine relevance and to infer higher-order relations.
- As higher-order relations are detected they are passed to the GUI, which displays them in both tabular and graphical forms.
- The query capability allows for both ordinary and “what if” queries.

Outline

- Cognitive Systems
  - Cognitive Loop
  - System Functionality
- Situation Awareness
  - Formalization
  - Situation Awareness Assistant
- Self-Awareness
  - Functionality
  - Software composition
- Feedback Control
  - Models
  - Stability

Self-Awareness

- Self-awareness is one part of cognition in general:
  - System is aware of its own capabilities and can reflect on its own behavior
  - System can modify its behavior to improve its performance
- Application domains
  - Radio communication (waveforms)
  - Data link layer communication
  - Defending against denial of service attacks

Ordinary Software

- Local information is stored in a data model that does NOT have high expressivity and machine processable semantics
  - Scalar variables and some simple structures can be exchanged using XML or JSON.
  - The capabilities and structure of a component cannot be exchanged.
- Messages between communication nodes are limited to the structure defined by the protocol
  - Messages in XML or JSON must be fully explicit

Cognitive Software

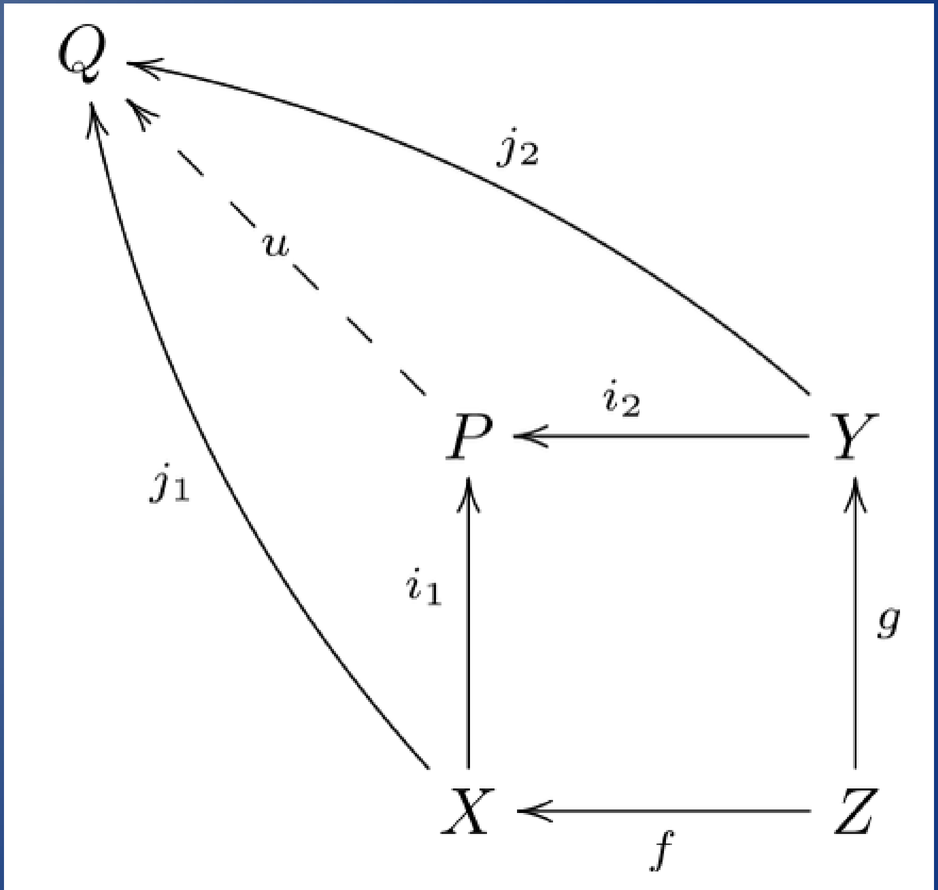
- Self-awareness enables
  - Full access to all processing variables (via reflection)
  - Inference can be used to reduce the communication overhead significantly (via ontology and rules)
  - Full access to all component capabilities and structure (via ontology and rules)
  - Dynamic reconfigurability (using a library of annotated modules)
- These were first demonstrated in joint work with my colleagues

Self-Awareness Demonstration

- Generation of Waveforms from Descriptions (L. Lechowicz, Ph.D. thesis)
- Objective: Verify that dynamic Ontology-based radio reconfigurability is feasible
- Transfer of knowledge (description of BPSK31, QPSK31, RTTY waveforms)
- Transferred knowledge integrated in the local knowledge base
- A waveform described in OWL/Rules constructed from its description
- Finite state machine built from the ontological description
- A complex software module composed from simpler software modules dynamically

Colimits

- The *colimit* of a commutative diagram of module morphisms (for example, X, Y and Z in the figure) is the module P in the figure.
- This example is a *pushout*. An actual system has a much larger number of modules.



Outline

- Cognitive Systems
  - Cognitive Loop
  - System Functionality
- Situation Awareness
  - Formalization
  - Situation Awareness Assistant
- Self-Awareness
  - Functionality
  - Software composition
- Feedback Control
  - Models
  - Stability

Feedback Control

- The basic function of the software system is regarded as a *Plant* to be controlled.
- The behavior of the Plant and the Environment is modeled *dynamic system*.
- Measurable inputs to the Plant are identified and split into *control inputs* and *disturbances*.
  - Control inputs are used for controlling the behavior of the Plant, while
  - Disturbances alter the behavior of the Plant in an unpredictable way.

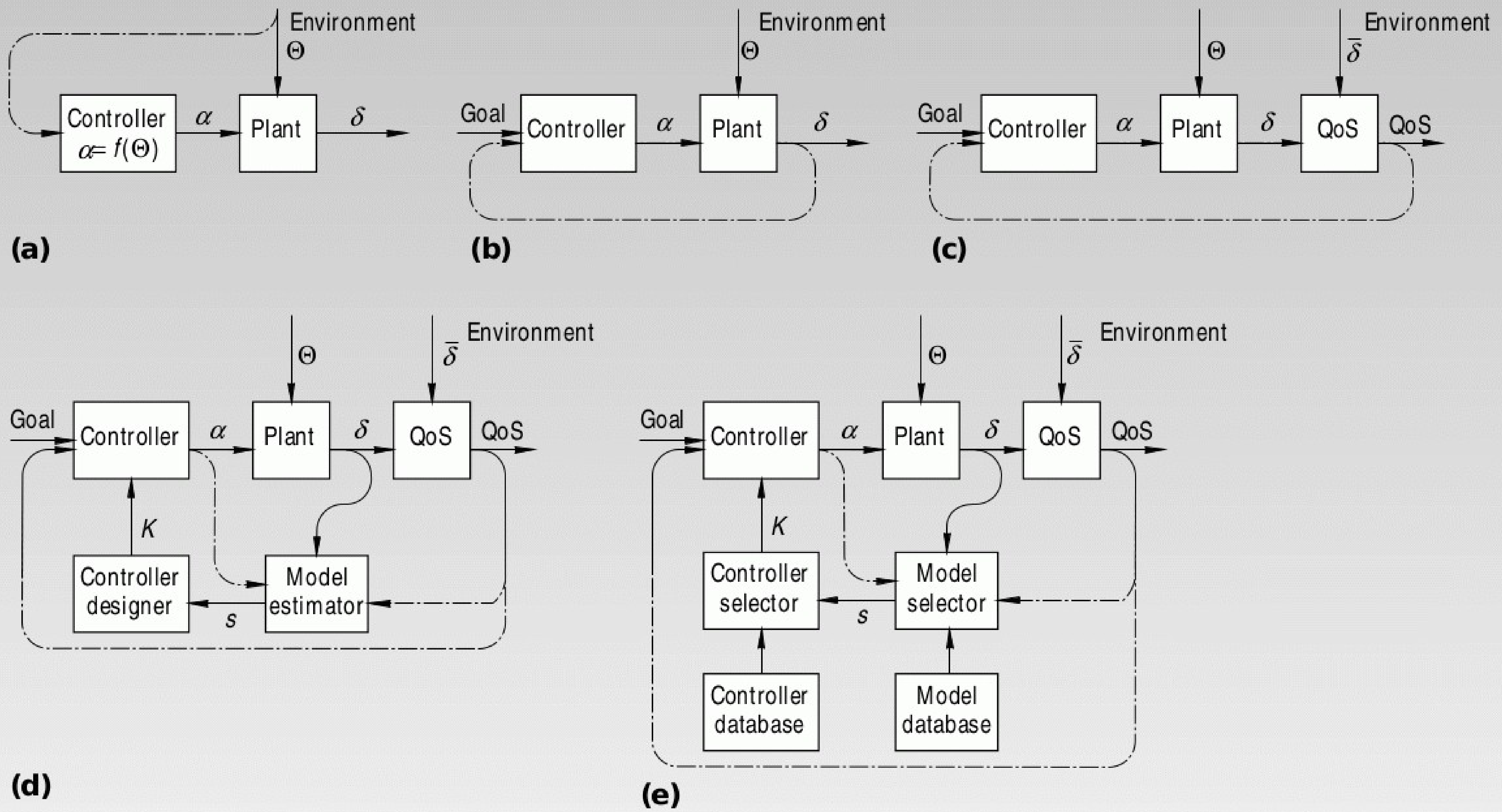
Feedback Control

- An additional subsystem is added for changing the values of the control inputs to the Plant, called the *Controller* subsystem.
- Yet another subsystem can be added for computing feedback, called the *Quality of Service (QoS)* subsystem.
  - This feedback is used by the Controller to compute control inputs.

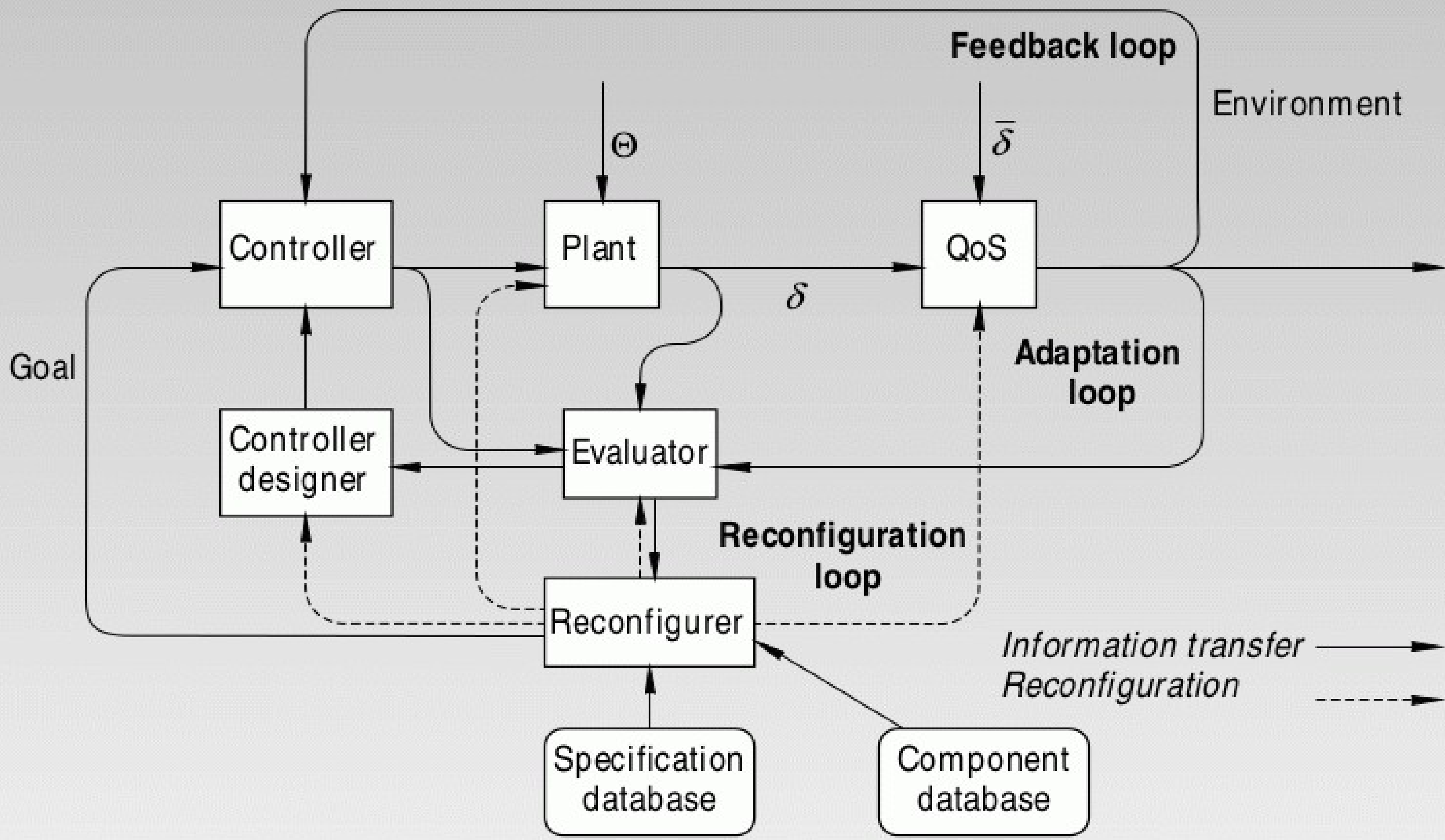
Feedback Control

- An additional subsystem is added for changing the values of the control inputs to the Plant, called the *Controller* subsystem.
- Yet another subsystem can be added for computing feedback, called the *Quality of Service (QoS)* subsystem.
  - This feedback is used by the Controller to compute control inputs.

Feedback Control Models



Self-Controlling Software Model



Self-Controlling Software Model

- Feedback loop
  - The Controller sets parameters to the Plant based upon goal and feedback received from the Quality-of-Service subsystem.
- Adaptation loop
  - The Evaluator evaluates the behavior and performance to determine whether the model of the Plant is appropriate, and
  - adapts the model,
  - which in turn triggers a change in the control law.

Self-Controlling Software Model (SCSM)

- Reconfiguration loop
  - Relatively costly action.
  - Performed by the Reconfigurer on request of the Evaluator.
  - Reconfiguration can involve structural changes in the Plant model, Quality-of-Service, Evaluator, Controller, Controller Designer, goal, or even the Plant.
  - The Reconfigurer uses
    - Specification Database
    - Component Database

Self-Controlling Software Model

- Specification Database
  - Component specifications
  - High-level system requirement
  - High-level system goal
- Component Database
  - Modules used for assembling the system
- Module composition
  - Based on the category theory notion of colimit
  - Requires checking commutativity of the morphisms
  - Requires formal proof of correctness of system requirement

Stability

- A software system is modeled as a discrete event system (DES).
- There are two dozen or so notions of stability for DES, such as:
  - stability in the sense of Lyapunov
  - asymptotic stability
  - asymptotic stability in the large
  - exponential stability
  - exponential stability in the large
  - stability in the sense of Lagrange
  - uniform boundedness
  - uniform ultimate boundedness

Stability

- Sufficient conditions for stability use a discrete analog of Lyapunov functions.
  - Difficult to find a Lyapunov function for complex dynamical systems
  - Not even possible, if the software system is too complex to have a closed-form mathematical formulation
- However, one can often find a bound
  - Bound is a form of worst case analysis
  - Bound is much simpler and tractable
  - Efficiency will depend on quality of the bound
  - Continual evaluation is required
  - The SCSM is designed for this purpose

Other Issues

- Controllability
- Observability
- Robustness (graceful degradation)
- Autonomy
- Generality
- Chattering
- Scheduling
- Proactive reconfiguration
- Efficiency

Bibliography/Acknowledgements

- General Publications
- Classified Publications
- Control Theory Publications
- Self-Awareness Publications
- Situation Awareness Publications
- Wireless Communication Publications
- Website: www.ccs.neu.edu/home/kenb