Raytheon
BBN Technologies

SILK: Efficiently Building Systems with Complex Behaviors and Varieties

Dr. Carl Andersen
Scientist, Knowledge Engineering Group
Raytheon BBN Technologies
canderse@bbn.com

3rd Annual DoD SOA & Semantic Technology Symposium
July 13, 2011

Outline

Raytheon
BBN Technologies

- SILK Introduction
- SILK Advantages
  - Note: SILK syntax is (mildly) simplified for presentation
- Demonstration – Policy Modeling
- Questions and Discussion

SILK Introduction

Raytheon
BBN Technologies

- Semantic Inferencing on Large Knowledge (SILK)
- A "logic programming" language and engine
  - UI, API (implemented in Java)
  - Flora2 + XSB Prolog back-end reasoner (impl. in C)
- Advanced research component of Vulcan Inc.'s Project Halo
  - Intelligent agents answer AP Biology questions
- In development by BBN and others, with rights for government use

Features of SILK

Raytheon
BBN Technologies

- "Declarative" = system state and behavior expressed as accessible facts and rules – easier for non-programmers to work with
- Capabilities beyond OWL and other rule languages
  - object-oriented + ontological syntax
  - defaults & exceptions
  - contradiction handling
  - easy to express minor variations
- Formal foundations = long-term stability
- Synthesis of existing Semantic Web standards and cutting edge LP research

Why SILK?

Raytheon
BBN Technologies

low-level languages (Java, .NET, XML, protocols)

- Flexible
  - supremely flexible
- Accessible
  - opaque to non-developers
- Ontological
  - code yes, world no
- Business logic
  - none
- Agile
  - far too low-level



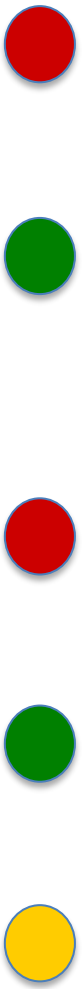


Why SILK?

Raytheon BBN Technologies

business processes (BPMN, BPEL)

- Flexible
  - inflexible, baked
- Accessible
  - intuitive shared representation for managers/developers/machines
- Ontological
  - no
- Business logic
  - yes
- Agile
  - modeling is quick
  - hard to define exceptions



Why SILK?

Raytheon BBN Technologies

OWL+RIF

```
DisjointClasses(
unionOf(a:animal
restriction(a:part_of
someValuesFrom
(a:animal)))
unionOf(a:plant
restriction(a:part_of
someValuesFrom
(a:plant))))
```

- Flexible
  - flexible
- Accessible
  - declarative, but unintuitive
- Ontological
  - yes
- Business logic
  - none
- Agile
  - modeling is quick
  - can't say what's NOT true
  - hard to define exceptions

Why SILK?

Raytheon BBN Technologies

SILK

- Flexible
  - supremely flexible
- Accessible
  - intuitive shared representation for managers/developers/machines
- Ontological
  - yes – built-in syntax
- Business logic
  - can model these and beyond
- Agile
  - can say what's NOT true
  - easy, succinct exceptions
  - graceful contradiction handling

Why SILK?

Raytheon
BBN Technologies

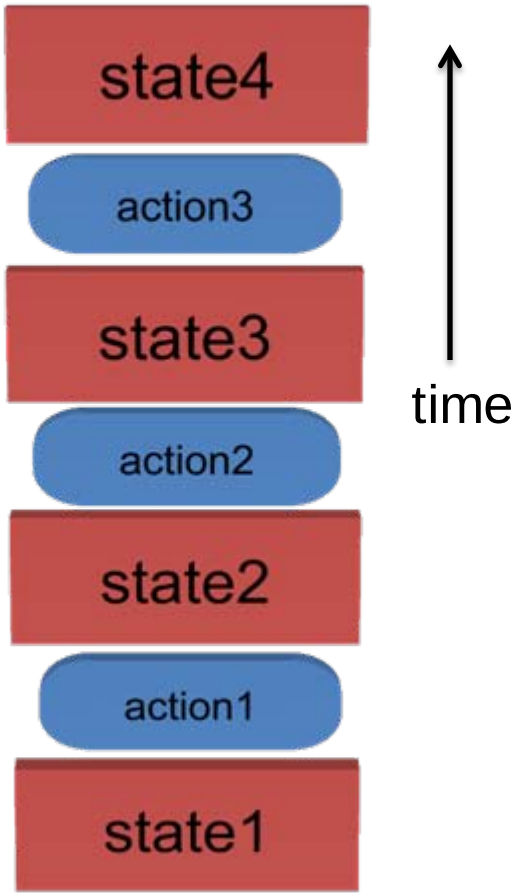
	low-level languages (Java, .NET, XML, protocols)	business processes (BPMN, BPEL)	OWL+RIF	SILK
Flexible				
Accessible				
Ontological				
Business Logic				
Agile				

Mix other approaches, or just use SILK?

Representing variations and exceptions

Raytheon BBN Technologies

- SILK can represent variations of complex rules/structures/processes:
  - easily
  - intuitively
  - concisely – only the differences
- Consider a simple process:
  - a series of actions
  - each action produces a new state
- Obviously, this is a simplification



Representing variations and exceptions

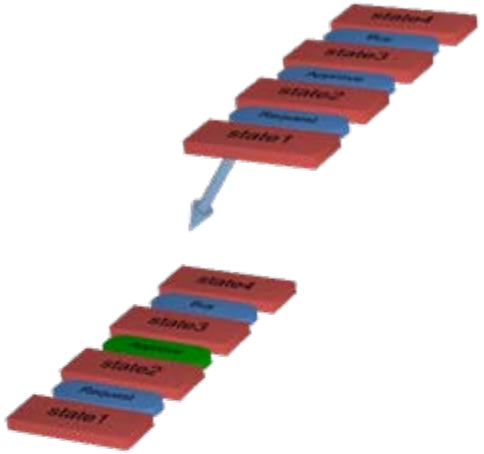
Raytheon
BBN Technologies

SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

SILK is the preferred method for developing ontologies/varieties of complex phenomena

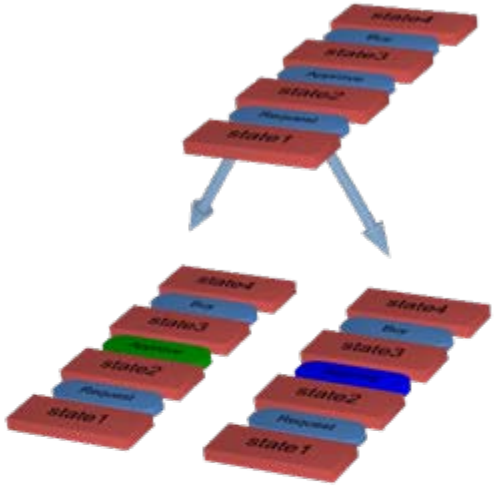
- like processes!



- …where each child is a subtle variation of its parent

SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

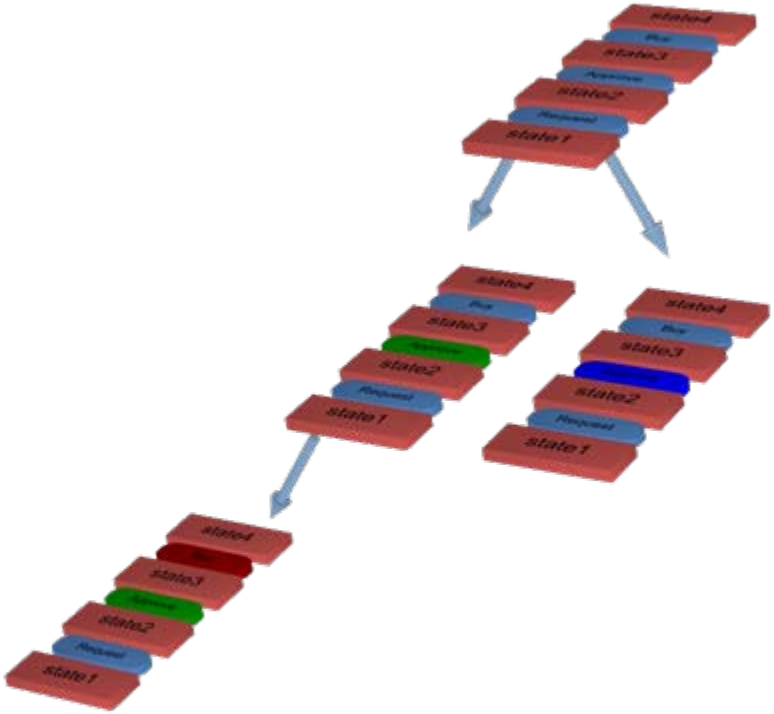


- …where each child is a subtle variation of its parent

SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

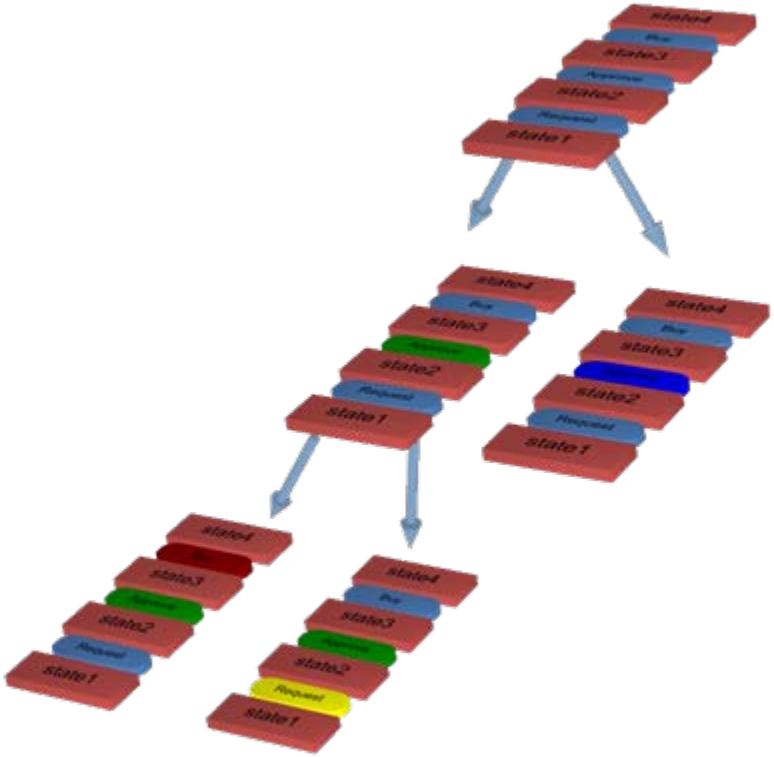
- …where each child is a subtle variation of its parent



SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

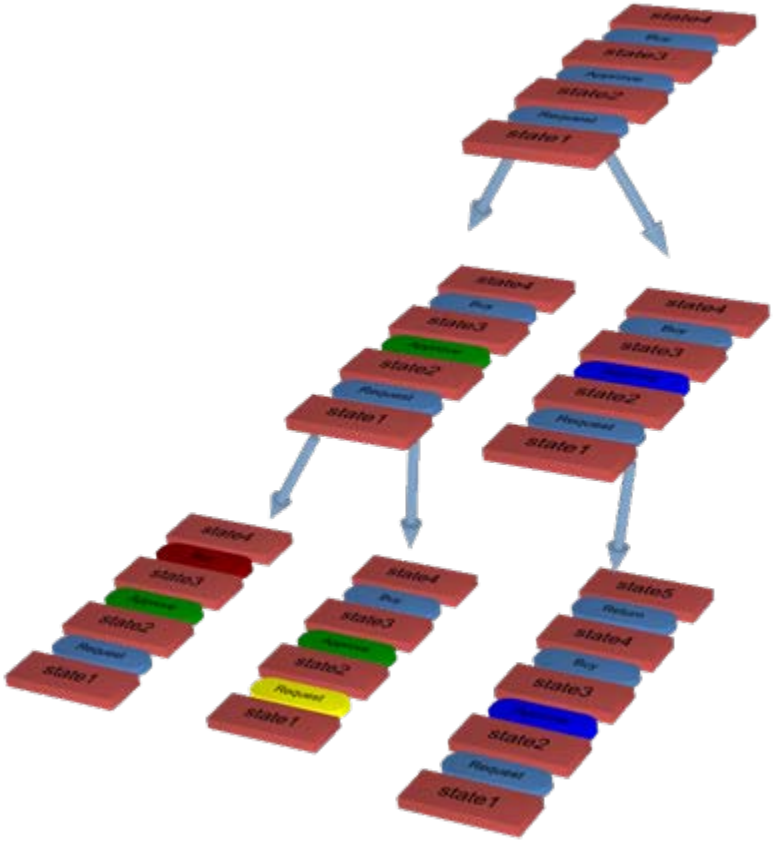
- …where each child is a subtle variation of its parent



SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

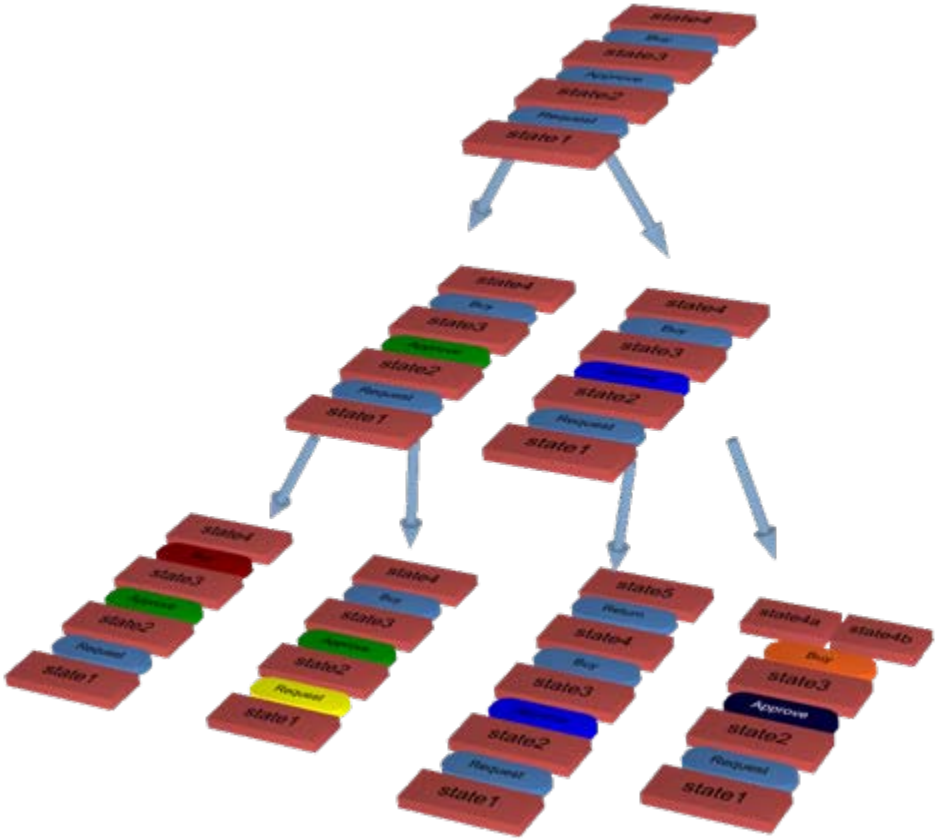
- …where each child is a subtle variation of its parent



SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

- …where each child is a subtle variation of its parent

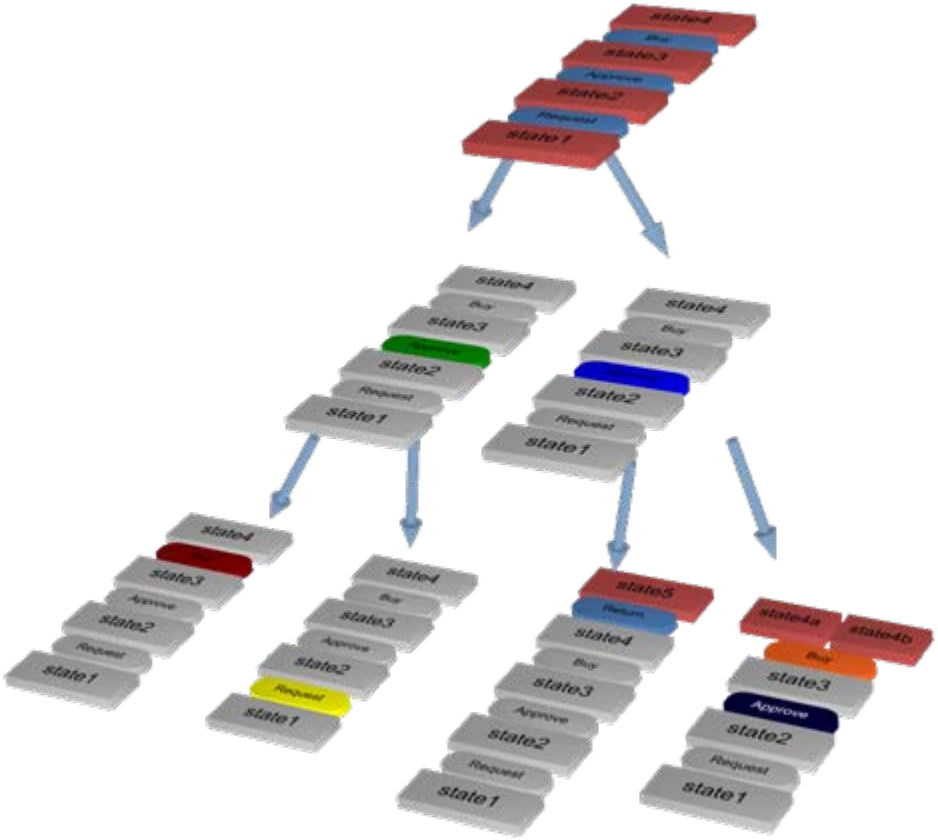


Raytheon BBN Technologies

SILK is the preferred method for developing ontologies/varieties of complex phenomena

- like processes!

- …where each child is a subtle variation of its parent
- …where we only need to enter each child's differences from its parent



SILK Expressiveness

Raytheon
BBN Technologies

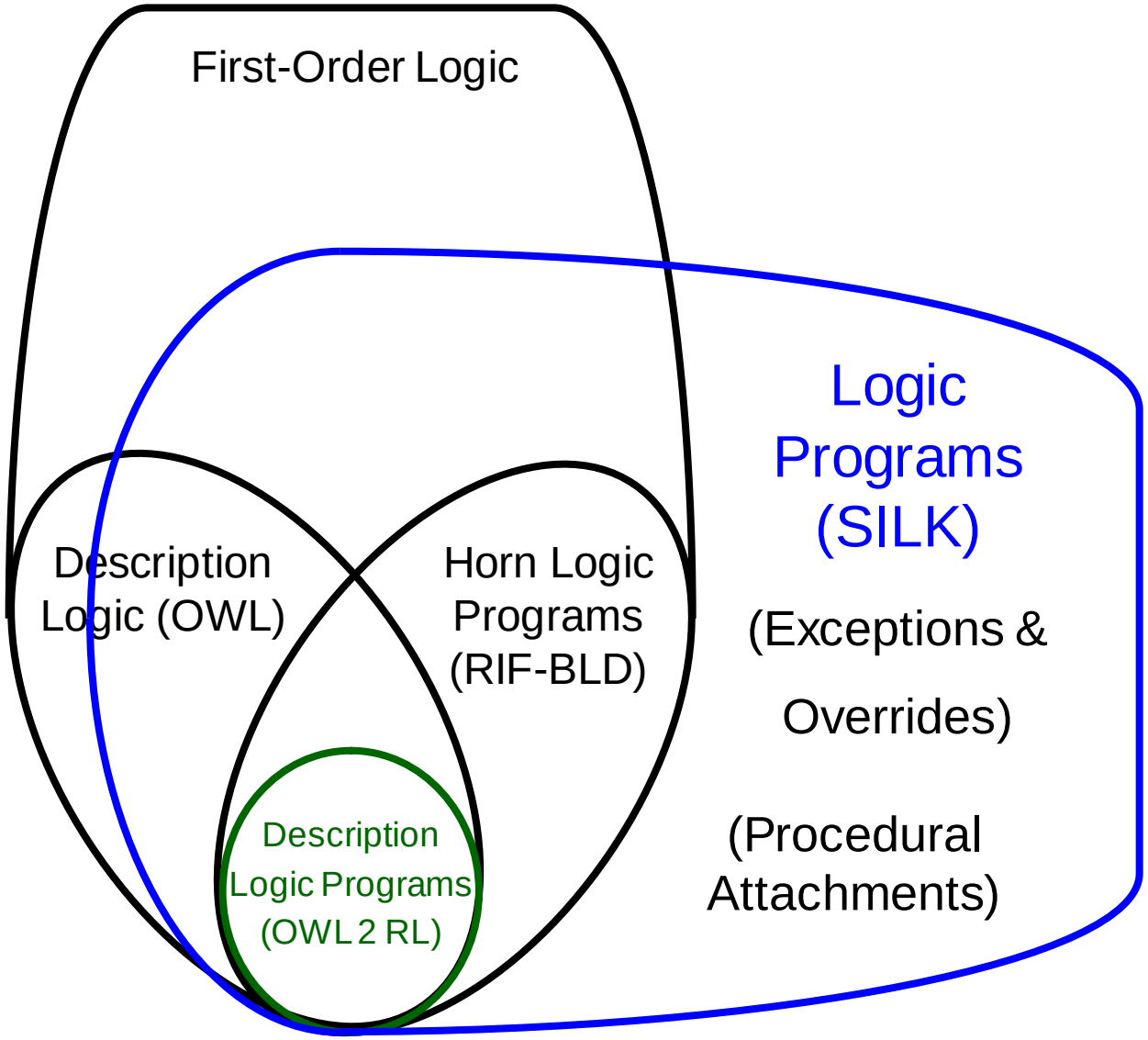
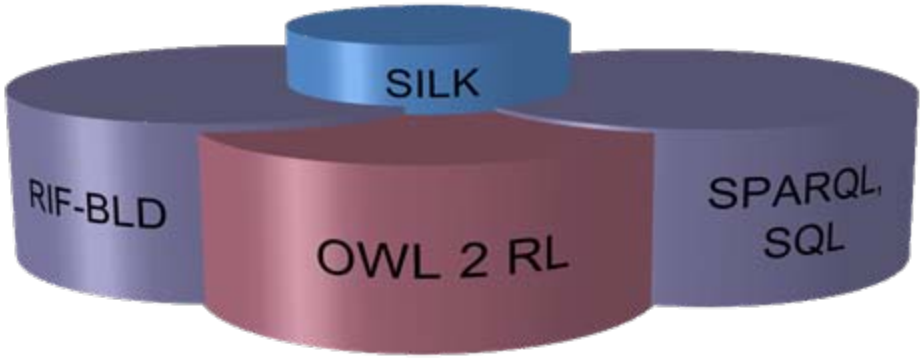


Image courtesy of Vulcan Inc., Benjamin Grosof, Mike Dean, and Michael Kifer.

SILK Builds Upon Standards

Raytheon
BBN Technologies

- SILK supports OWL 2 RL, RIF-BLD, RDFS imports & reasoning
- SILK supports federated data via remote SPARQL, SQL queries
- SILK is Sem Web friendly – supports URIs, XML Schema Datatypes, SPARQL

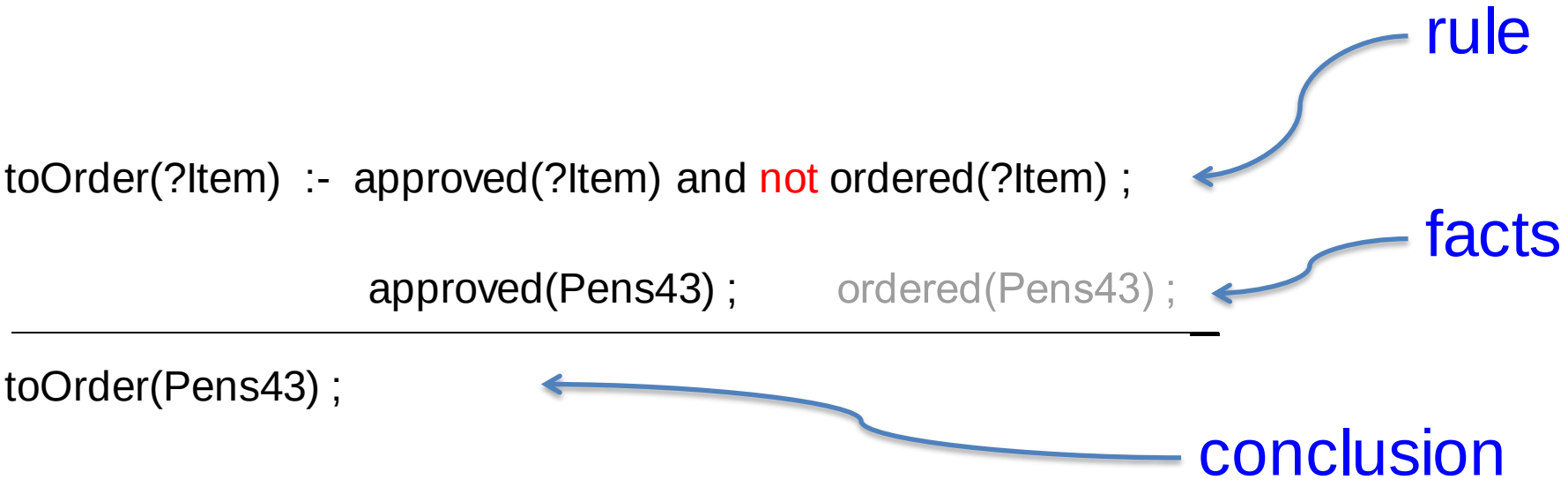


- Users can adhere to standards for the bulk of processing, accessing SILK's extra power only when necessary

How SILK (Logic Programming) Rules Work

Raytheon BBN Technologies

- SILK is akin to a "deductive" relational database
- Rules in the "knowledge base" (KB) allow us to conclude new facts



- SILK elaborates upon this paradigm

SILK's Override Mechanism

Raytheon BBN Technologies

- SILK allows us to say things and later make exceptions

- All purchases need approval of a manager
- BUT - expensive purchases need VP approval
- Express this exception: the order>1K rule automatically "overrides" the order rule
- Result: purchases continue to only need manager approval UNLESS they are expensive, in which case VP approval is needed

```
@order doOrder(?Item) :- approvedMgr(?Item) ;

@order>1K not doOrder(?Item) :- cost(?Item) >
1000 and not approvedVP(?Item) ;

overrides(order>1K,order) ;

SILK>  ?- doOrder(Pens43) ;
TRUE

SILK>  ?- doOrder(DBSoftware11) ;
FALSE
```

SILK's Override Mechanism

Raytheon BBN Technologies

- This mechanism can also automatically detect conflicts/bugs

```
@order doOrder(?Item) :- approvedMgr(?Item) ;

@order>1K not doOrder(?Item) :- cost(?Item) >
1000 and not approvedVP(?Item) ;


overrides(order>1K,order) ;


SILK>  ?- doOrder(Pens43) ;
TRUE

SILK>  ?- doOrder(DBSoftware11) ;
ALARM!  CONTRADICTION!
```

Object-oriented/ontologies, but with exceptions

Raytheon BBN Technologies

- SILK has special syntax to express exceptions within ontologies

- Define Item and its subclass

```
Pens43 # Item ;
DBSoftware11  # ExpensiveItem ;
ExpensiveItem ## Item ;
```

- Define an (overridable) property. An Item is typically approved by a manager, but a subclass can override this

```
Item[approvedBy → Mgr] ;
ExpensiveItem[approvedBy → VP] ;
```

Object-oriented/ontologies, but with exceptions

Raytheon BBN Technologies

- SILK has special syntax to express exceptions within ontologies

- Define Item and its subclass

```
Pens43 instance-of Item ;
DBSoftware11 instance-of ExpensiveItem ;
ExpensiveItem subclass-of Item ;
```

- Define an (overridable) property. An Item is typically approved by a manager, but a subclass can override this

```
Item[approvedBy → Mgr] ;
ExpensiveItem[approvedBy → VP] ;
```

Object-oriented/ontologies, but with exceptions

Raytheon BBN Technologies

- SILK has special syntax to express exceptions within ontologies

- Define Item and its subclass

```
Pens43 instance-of Item ;
DBSoftware11 instance-of ExpensiveItem ;
ExpensiveItem subclass-of Item ;
```

- Define an (overridable) property. An Item is typically approved by a manager, but a subclass can override this

```
Item[approvedBy → Mgr] ;
ExpensiveItem[approvedBy → VP] ;
```

- Similar to OO polymorphism, this kind of specialization is the way the world works – "every type has an exception".

```
SILK> ?- DBSoftware11[approvedBy → Mgr] ;
FALSE
```

- Intuitive; matches the way people think of exceptions

```
SILK> ?- DBSoftware11[approvedBy → VP] ;
TRUE
```

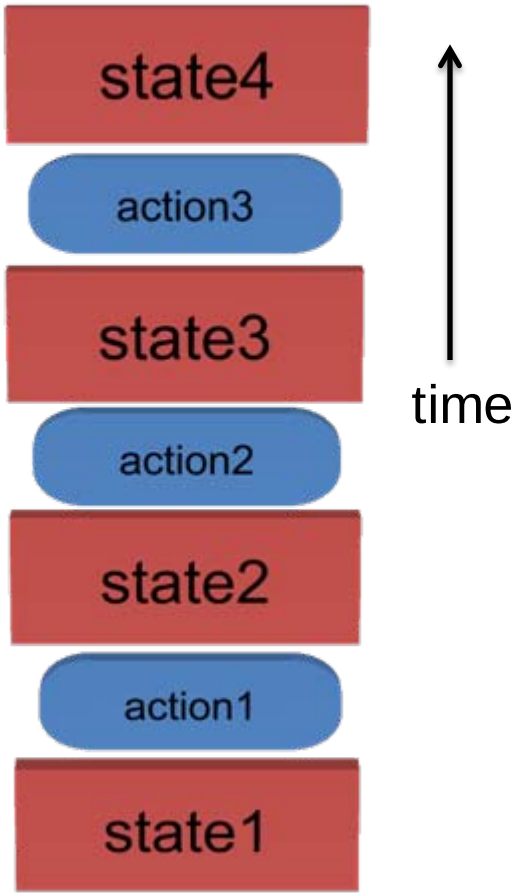
- Beyond OWL's powers

- SILK's syntax is also ideal for modeling complex business phenomena, such as processes

- Consider a simple process:
  - a series of actions
  - each action produces a new state

- Obviously, this is a simplification

- Policies can be thought of as constraints (on processes) – "the approved way to do something"



Raytheon BBN Technologies

It's easy(er) to model processes & policies using SILK's logical syntax:

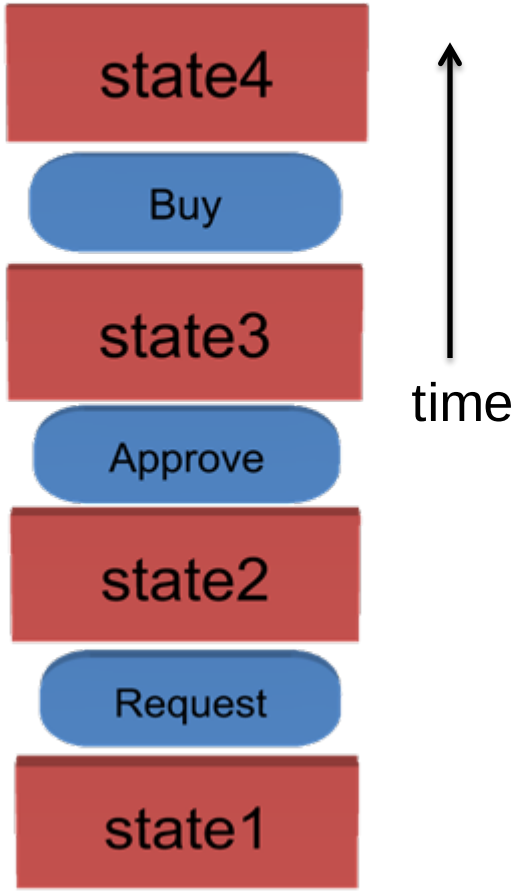
- e.g. actions:

"An Approval action cannot begin until a request to purchase the item is received."

"The action's effect is to approve the purchase of the item."

"Allow 3 days for approval."

Approve[object → ?Item,
pre → receivedRequest(?Item),
post → isApproved(buy(?Item)),
leadTime → 3 days]



Raytheon BBN Technologies

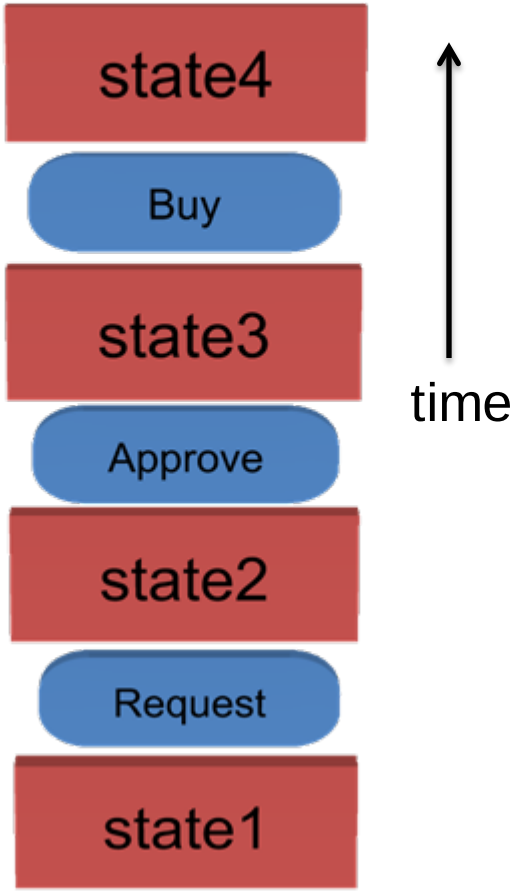
It's easy(er) to model processes & policies using SILK's logical syntax: especially exceptions!

"An Approval action cannot begin until a request to purchase the item is received."

"The action's effect is to approve the purchase of the item."

"Allow 3 days for approval."

Approve[object → ?Item,
pre → receivedRequest(?Item),
post → isApproved(buy(?Item)),
leadTime → 3 days]



It's easy(er) to model processes & policies using SILK's logical syntax: especially exceptions!

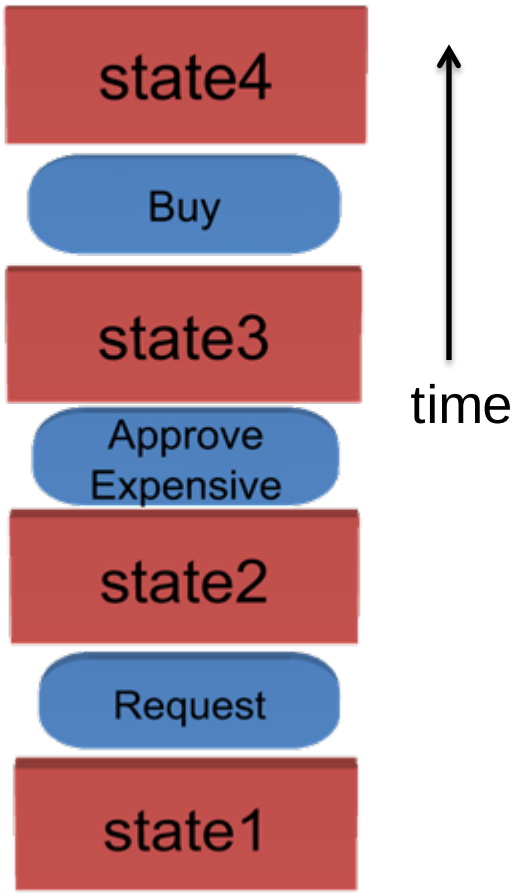
- an expensive ?Item:

"An Approval action cannot begin until a request to purchase the item is received."

"The action's effect is to submit a request for purchase to the VP."

"Allow 5 days for approval."

ApproveExpensive[object → ?Item,
pre → receivedRequest(?Item),
post → submitRequest(?Item,VP),
leadTime → 5 days]



Raytheon BBN Technologies

It's easy(er) to model processes & policies using SILK's logical syntax: especially exceptions!

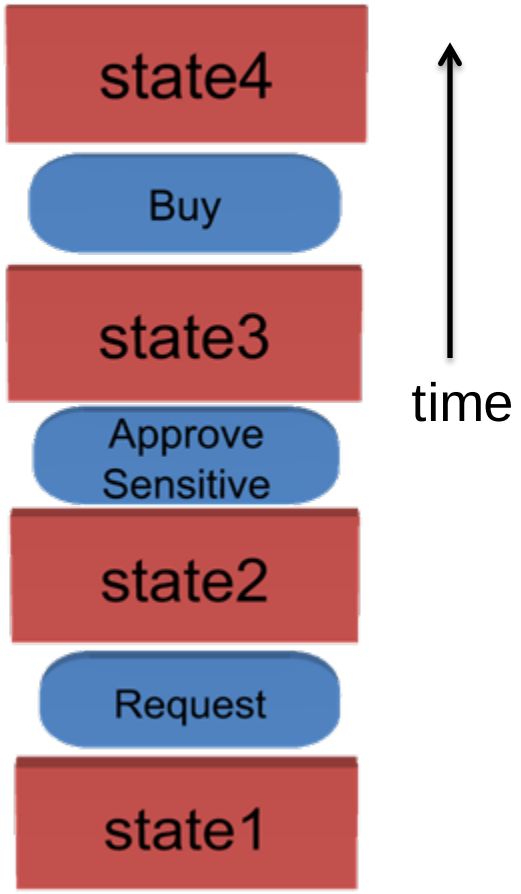
- an ?Item requiring special handling

"An Approval action cannot begin until a request to purchase the item is received in triplicate."

"The action's effect is to approve the purchase of the item."

"Allow 3 days for approval."

ApproveSensitive[object → ?Item,
pre → receivedRequestX3(?Item),
post → isApproved(buy(?Item)),
leadTime → 3 days]



Raytheon BBN Technologies

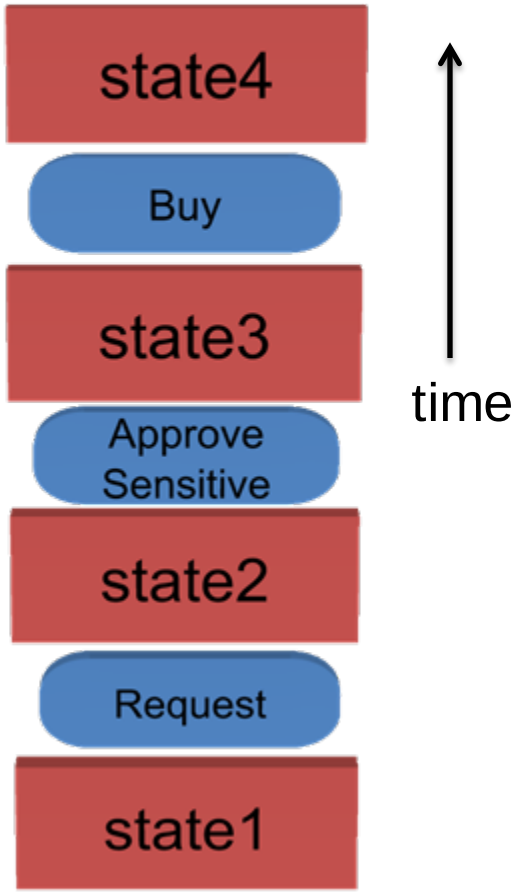
The best part is – you only have to say what's different.

"An Approval action cannot begin until a request to purchase the item is received in triplicate."

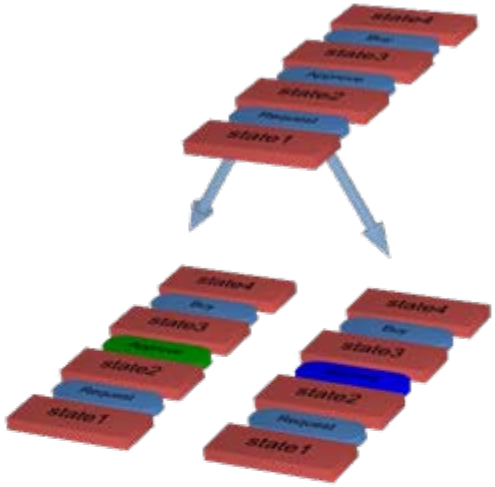
"The action's effect is to approve the purchase of the item."

"Allow 3 days for approval."

ApproveSensitive[object → ?Item,
pre → receivedRequestX3(?Item),
post → isApproved(buy(?Item)),
leadTime → 3 days]

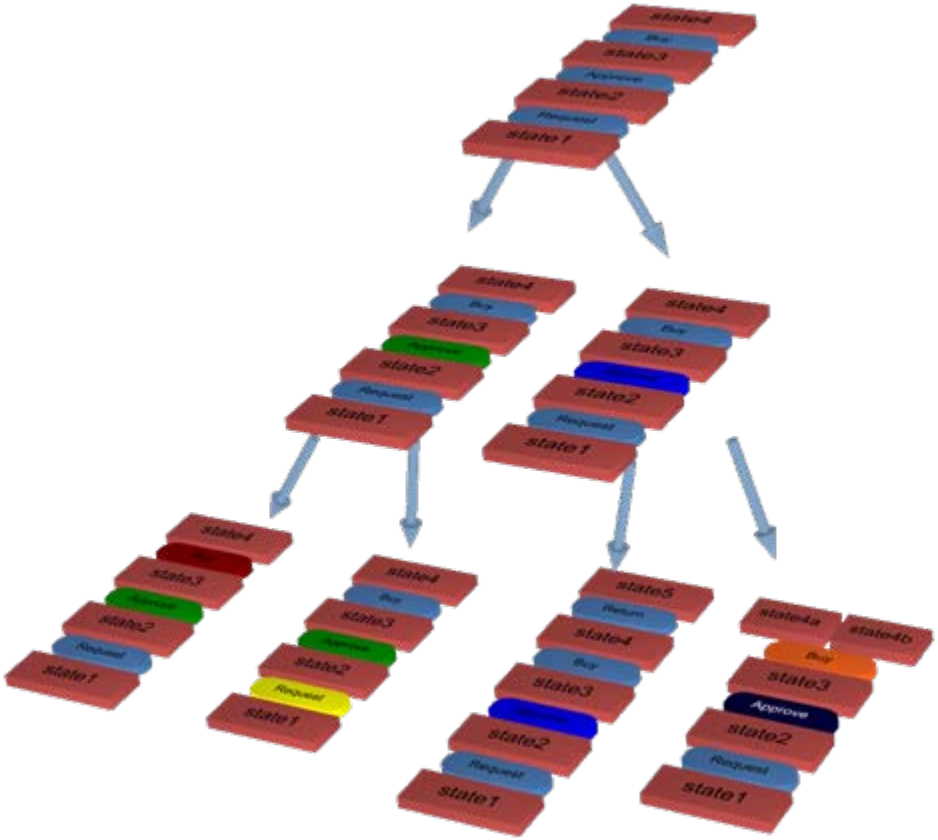


We can use this approach to develop ontologies of processes/policies



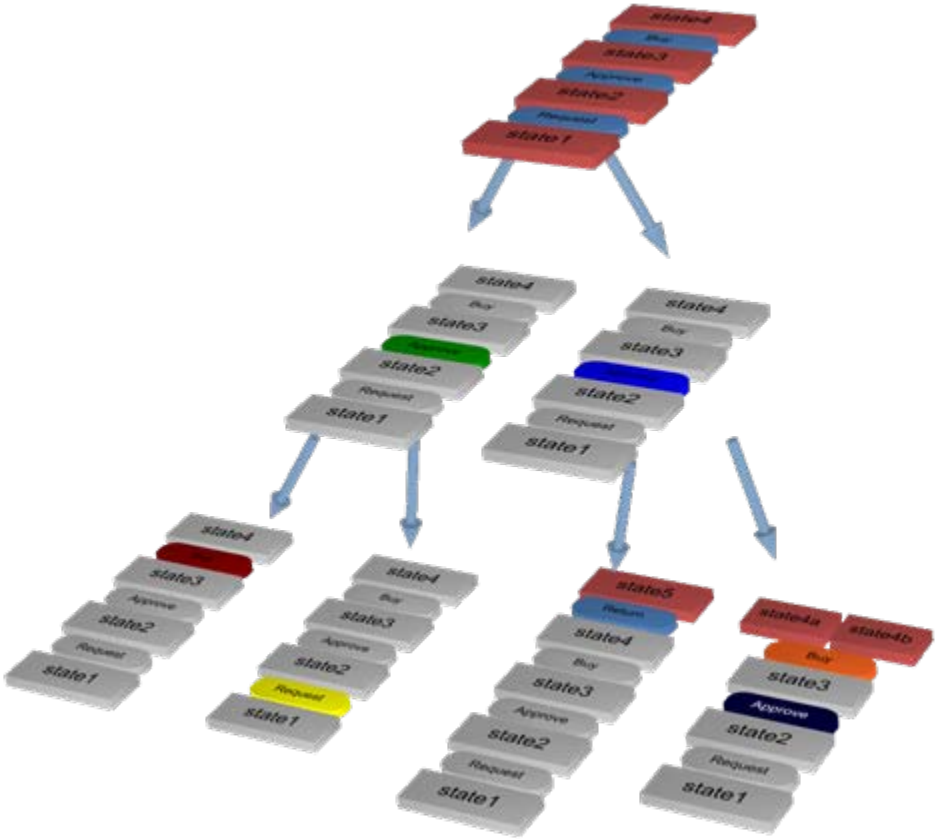
We can use this approach to develop ontologies of processes/policies

- …where each child is a subtle variation of its parent



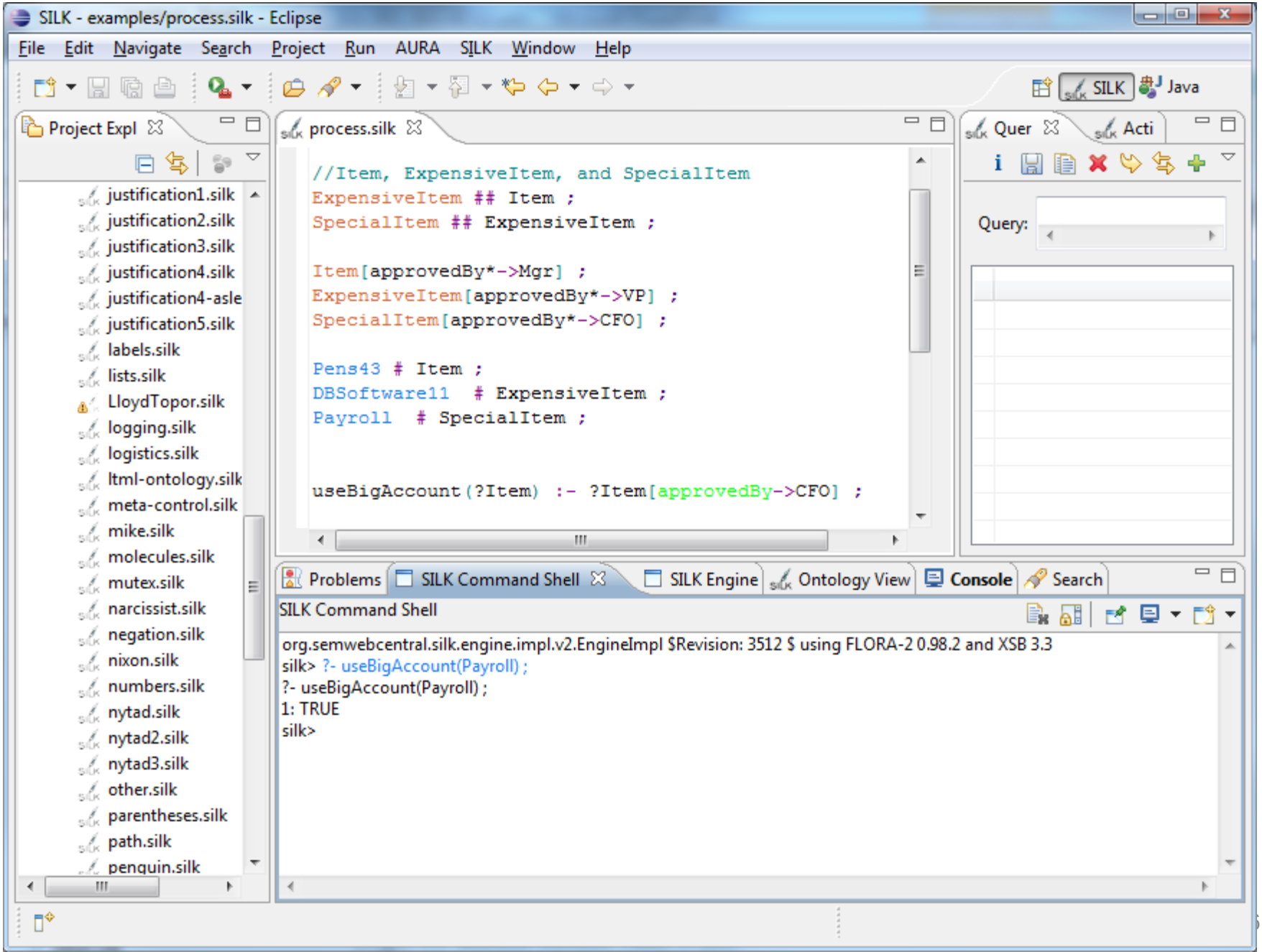
We can use this approach to develop ontologies of processes/policies

- …where each child is a subtle variation of its parent
- …where we only need to enter each child's differences from its parent



Demonstration of Policy Modeling

Raytheon
BBN Technologies



Additional Information

Raytheon
BBN Technologies

- http://silk.semwebcentral.org
  - Public SILK site
  - Presentations and conference papers
  - RIF SILK dialect
- http://www.mit.edu/~bgrosof/paps/talk-iswc2010-rules-tutorial.pdf
  - Grosof/Dean/Kifer Web rules tutorial
  - 219 slides with theoretical and application details

Thanks! Questions?