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

• SILK Introduction

• SILK Advantages
  – Note: SILK syntax is (mildly) simplified for presentation

• Demonstration – Policy Modeling

• Questions and Discussion
SILK Introduction

• 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

- "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?

• Flexible
  – supremely flexible

• Accessible
  – opaque to non-developers

• Ontological
  – code yes, world no

• Business logic
  – none

• Agile
  – far too low-level
Why SILK?

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?

OWL+RIF

- **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

DisjointClasses(
  unionOf(a:animal
  restriction(a:part_of
  someValuesFrom (a:animal)))
unionOf(a:plant
restriction(a:part_of
someValuesFrom (a:plant))))
Why SILK?

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?

<table>
<thead>
<tr>
<th>Feature</th>
<th>low-level languages (Java, .NET, XML, protocols)</th>
<th>business processes (BPMN, BPEL)</th>
<th>OWL+RIF</th>
<th>SILK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexible</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Accessible</td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Ontological</td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Business Logic</td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Yellow" /></td>
</tr>
<tr>
<td>Agile</td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Yellow" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
</tbody>
</table>

Mix other approaches, or just use SILK?
Representing variations and exceptions

- 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

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

• …where each child is a subtle variation of its parent
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

• …where each child is a subtle variation of its parent
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

- ...where each child is a subtle variation of its parent
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

- …where each child is a subtle variation of its parent
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

• …where each child is a subtle variation of its parent
Representing variations and exceptions

SILK is the preferred method for developing ontologies/varieties of complex phenomena - like processes!

• …where each child is a subtle variation of its parent
Representing time, events and 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
• …where we only need to enter each child’s differences from its parent
SILK Expressiveness

逻辑（OWL）

描述逻辑（OWL）

霍恩逻辑程序（RIF-BLD）

逻辑程序（SILK）

（异常情况 & 重写）

（程序性附件）

图象由Vulcan Inc., Benjamin Grosof, Mike Dean, and Michael Kifer提供。
SILK Builds Upon Standards

- **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

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

\[
\text{toOrder}(\text{?Item}) \ :- \ \text{approved}(\text{?Item}) \ \text{and} \ \text{not} \ \text{ordered}(\text{?Item}) ; \\
\hline
\text{approved(Pens43)} ; \quad \text{ordered(Pens43)} ;
\]

\[\downarrow \text{toOrder(Pens43)} ;\]

- SILK elaborates upon this paradigm
SILK’s Override Mechanism

- 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

```prolog
@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

- This mechanism can also automatically detect conflicts/bugs

```prolog
@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

- SILK has special syntax to express exceptions within ontologies

- Define Item and its subclass

  Pens43 # Item ;
  DBSoftware11 # Expensiveltem ;
  Expensiveltem ## Item ;

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

  Item[approvedBy → Mgr] ;
  Expensiveltem[approvedBy → VP] ;
Object-oriented/ontologies, but with exceptions

- SILK has special syntax to express exceptions within ontologies

- Define Item and its subclass
  
  Pens43 instance-of Item ;
  DBSoftware11 instance-of Expensiveltem ;
  Expensiveltem subclass-of Item ;

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

  Item[approvedBy → Mgr] ;
  Expensiveltem[approvedBy → VP] ;
Object-oriented/ontologies, but with exceptions

• SILK has special syntax to express exceptions within ontologies

• Define Item and its subclass

Pens43 instance-of Item ;
DBSoftware11 instance-of Expensiveltem ;
Expensiveltem subclass-of Item ;

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

Item[approvedBy → Mgr] ;
Expensiveltem[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
Representing time, events and processes

• 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”
Representing time, events and processes

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\[object \rightarrow \text{?Item},\]
  \[pre \rightarrow \text{receivedRequest(?Item)},\]
  \[post \rightarrow \text{isApproved(buy(?Item))},\]
  \[leadTime \rightarrow \text{3 days}]\]
"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]
Representing time, events and processes

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]
Representing time, events and processes

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]
Representing time, events and processes

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.”

\[
\text{ApproveSensitive}\{\text{object} \rightarrow ?\text{Item}, \text{pre} \rightarrow \text{receivedRequestX3(?Item)}, \text{post} \rightarrow \text{isApproved(buy(?Item)), leadTime} \rightarrow 3 \text{ days}\}
\]
Representing time, events and processes

We can use this approach to develop ontologies of processes/policies
Representing time, events and processes
We can use this approach to develop ontologies of processes/policies

• …where each child is a subtle variation of its parent
Representing time, events and processes

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

```java
// Item, ExpensiveItem, and SpecialItem
ExpensiveItem ## Item;
SpecialItem ## ExpensiveItem;
Item[approvedBy*->Mgr];
ExpensiveItem[approvedBy*->VP];
SpecialItem[approvedBy*->CFO];

Pens43 # Item;
DBSoftware11 # ExpensiveItem;
Payroll # SpecialItem;

useBigAccount(?Item) :- ?Item[approvedBy->CFO];
```

SILK Command Shell
```
org.semwebcentral.silk.engine.impl.v2.EngineImpl $Revision: 3512 $ using FLORA-2 0.98.2 and XSB 3.3
silk> ?- useBigAccount(Payroll);
?- useBigAccount(Payroll);
1: TRUE
silk>
```
Additional Information

- **http://silk.semwebcentral.org**
  - Public SILK site
  - Presentations and conference papers
  - RIF SILK dialect

  - Grosof/Dean/Kifer Web rules tutorial
  - 219 slides with theoretical and application details

Thanks! Questions?