How SW Rules + Ontologies Connect to Procedural Aspects of SW Services

Presentation for
Semantic Web Services Language committee of
Semantic Web Services Coalition (a.k.a. Initiative)
April 11, 2003, at face-to-face, Miami, FL, USA

Prof. Benjamin Grosof

MIT Sloan School of Management

bgrosof@mit.edu http://www.mit.edu/~bgrosof/

Overview: {SW rules+ontologies} and the procedural aspect of SWS

Quickly review: rule-based SWS cf. the 3/20 SWSL telecon presentation and 4/9 DAML PI Mtg Services Breakout

- Describing post-conditions and pre-conditions, esp. contingent behavior
- Let's do more use cases and application scenarios

- Situated logic programs (SLP) [the largest emphasis of this presentation]
  - very simple workflow, viewable as timeless and stateless
  - abstraction of event-condition-action rules and OPS5 production rules
  - supported in RuleML and (basically too in) Jess.
  - actions (invoke external procedures) triggered by inferring of conclusions
  - queries (invoke external procedures) during testing of rule antecedent conditions
- Built-ins used in rules and ontologies, e.g.,
  - arithmetic and comparison operators/functions
- Exception handling in workflows and service agreements/contracts
- Representing service post-conditions and state transitions, incl. contracts, persistence defaults [-- next presentation could usefully have more on this]

Rule-based Semantic Web Services

- Rules/LP in appropriate combination with DL as KR, for RSWS
  - DL good for categorizing: a service overall, its inputs, its outputs
- Rules to describe service process models
  - rules good for representing:
    - preconditions and postconditions, their contingent relationships
    - contingent behavior/features of the service more generally,
      - e.g., exceptions/problems
  - familiarity and naturalness of rules to software/knowledge engineers
- Rules to specify deals about services: cf. e-contracting.

Rule-based Semantic Web Services

- Rules often good to executably specify service process models
  - e.g., business process automation using procedural attachments to perform side-effectful/state-changing actions ("effectors" triggered by drawing of conclusions)
  - e.g., rules obtain info via procedural attachments ("sensors" test rule conditions)
  - e.g., rules for knowledge translation or inferencing
  - e.g., info services exposing relational DBs
- Infrastructural: rule system functionality as services:
  - e.g., inferencing, translation

Application Scenarios for Rule-based Semantic Web Services

- SweetDeal [Grosof & Poon 2002] configurable reusable e-contracts:
  - LP rules about agent contracts with exception handling
  - … on top of DL ontologies about business processes;
  - *a scenario motivating DLP*
- Other:
  - Trust management / authorization (Delegation Logic) [Li, Grosof, & Feigenbaum 2000]
  - Financial knowledge integration (ECOIN) [Firat, Madnick, & Grosof 2002]
    - Rule-based translation among contexts / ontologies
    - Equational ontologies
  - Business policies, more generally, e.g., privacy (P3P)

Flavors of Rules Commercially Most Important today in E-Business

- E.g., in OO app's, DB's, workflows.

- Relational databases, SQL: Views, queries, facts are all rules.
  - SQL99 even has recursive rules.
- Production rules (OPS5 heritage): e.g.,
  - Jess, Blaze, ILOG, Haley: rule-based Java/C++ objects.
- Event-Condition-Action rules (loose family), cf.:
  - business process automation / workflow tools.
  - active databases; publish-subscribe.
- Prolog, e.g., XSB: *"logic programs" as a full programming language.*
- *(Lesser: other knowledge-based systems.)*

Heavy Reliance on Procedural Attachments in Currently Commercially Important Rule Families

- E.g., in OO app's, DB's, workflows.

- Relational databases, SQL: Built-in sensors, e.g., for arithmetic, comparisons, aggregations. Sometimes effectors: active rules / triggers.

- Production rules (OPS5 heritage): e.g., Jess
  - Pluggable (and built-in) sensors and effectors.
- Event-Condition-Action rules:
  - Pluggable (and built-in) sensors and effectors.

- Prolog: e.g., XSB.
  - Built-in sensors and effectors. More recent systems: more pluggability of the built-in attached procedures.

Situated LP's: Overview

- Point of departure: LP's are pure-belief representation, but most practical rule systems want to invoke external procedures.
- Situated LP 's feature a semantically-**clean** kind of **procedural attachments**. I.e., they hook beliefs to drive procedural API's outside the rule engine.
- Procedural attachments for **sensing** (queries) when testing an antecedent condition or for **effecting** (actions) upon concluding a consequent condition. Attached procedure is invoked when testing or concluding in inferencing.
- Sensor or effector **link** statement specifies an association from a predicate to a procedural call pattern, e.g., a method. A link is specified as part of the representation. I.e., a SLP is a conduct set that includes links as well as rules.

Situated LP's: Overview (cont.'d)

- phoneNumberOfPredicate   ::s::   BoeingBluePagesClass.getPhoneMethod . *ex. sensor link*
- shouldSendPagePredicate   ::e::   ATTPagerClass.goPageMethod . *ex. effector link*
- Sensor procedure may require some arguments to be ground, i.e., bound; in general it has a specified binding-signature.
- Enable dynamic or remote invocation/loading of the attached procedures (exploit Java goodness).

- Overall: cleanly separate out the procedural semantics as a declarative extension of the pure-belief declarative semantics. Easily separate chaining from action.

SweetJess: Translating an Effector Statement

```
<damlRuleML:effe>
  <damlRuleML:_opr>
   <damlRuleML:rel>giveDiscount</damlRuleML:rel>
  </damlRuleML:_opr>
  <damlRuleML:_aproc>
   <damlRuleML:jproc>
     <damlRuleML:meth>setCustomerDiscount</damlRuleML:meth>
     <damlRuleML:clas>orderMgmt.dynamicPricing</damlRuleML:clas>
     <damlRuleML:path>com.widgetsRUs.orderMgmt
       </damlRuleML:path>
   </damlRuleML:jproc>
  </damlRuleML:_aproc>

</damlRuleML:effe>
```

Associates with predicate P : an attached procedure A that is side-effectful.

- Drawing a conclusion about P triggers an action performed by A.

jproc = Java attached procedure.

meth, clas, path = its methodname, classname, pathname.

Equivalent in JESS: key portion is:

```
(defrule effect_giveDiscount_1
  (giveDiscount ?percentage ?customer)
  =>
  (effector setCustomerDiscount orderMgmt.dynamicPricing
                (create$ ?percentage  ?customer) ) )
```

Overview: Semantics of Situated Logic Programs

- Definitional: complete inferencing+action occurs during an "episode" – intuitively, run all the rules (including invoking effectors and sensors as go), then done.
- Effectors can be viewed as all operating/invoked after complete inferencing has been performed.
  - Independent of inferencing control.
    - But often intuitively less appropriate if only doing backward inferencing.
  - Separates pure-belief conclusion from action.

Overview: Semantics of Situated LP, continued

- Sensors can be viewed as accessing a virtual knowledge base (of facts). Their results simply augment the local set of facts. These can be saved (i.e., cached) during the episode.
  - Independent of inferencing control.
- The sensor attached procedure could be a remote powerful DB or KB system, a web service, or simply some humble procedure.
- Likewise, an effector attached procedure could be a remote web service, or some humble procedure. An interesting case for SW is when it performs updating of a DB or KB, e.g., "delivers an event".

Overview of Semantics of Situated LP, continued

- Conditions:
  - Effectors have only *side* effects: they do not affect operation of the (episode's) inferencing+action engine itself, nor change the (episode's) knowledge base.
  - Sensors are purely informational: they do not have side effects (i.e., any such can be ignored).
  - Timelessness of sensor and effector calls: their results are not dependent on when they are invoked, during a given inferencing episode.
  - "Sensor-safeness": Each rule ensures sufficient (variable) bindings are available to satisfy the binding signature of each sensor associated with any of its body literals – such bindings come from the other, non-sensor literals in the rule body. During overall "testing" of a rule body, sensors needing such bindings can be viewed as invoked after the other literals have been "tested".

Overview: Semantics of Situated LP, Continued

- Generalizations possible:
  - permit multiple sensors or effectors per predicate.
  - sense functions (or terms) not just predicates.
  - permit sensor priority – i.e, specify the prioritization of the facts that result from a particular sensor .
  - associate sensing with atoms/literals (or terms), but this is reducible to sensing predicates (or functions) – by rewriting of the rules.
- Challenge: error handling info returned from attached procedures

Example: Notifying a Customer when their Order is Modified

- See extended version of B. Grosof WITS-2001 paper
  - "Representing E-Business Rules on the Semantic Web: Situated Courteous Logic Programs in RuleML"
  - In file wits01-report-r2.pdf
  - Also at http://ebusiness.mit.edu/bgrosof

SweetDeal OPTIONAL SLIDES FOLLOW

Example Contract Proposal with Exception Handling
Represented using RuleML & DAML+OIL, Process Descriptions

Using concise text syntax

(SCLP textfile format)

for concise human reading

```
buyer(co123,acme);
seller(co123,plastics_etc);
product(co123,plastic425);
price(co123,50);
quantity(co123,100);
http://xmlcontracting.org/sd.daml#Contract(co123);
http://xmlcontracting.org/sd.daml#specFor(co123,co123_process);
http://xmlcontracting.org/sd.daml#BuyWithBilateralNegotiation(co123_process);
http://xmlcontracting.org/sd.daml#result(co123,co123_res);
shippingDate(co123,3); // i.e. 3 days after order placed
// base payment = price * quantity
payment(?R,base,?Payment) <-
  http://xmlcontracting.org/sd.daml#result(co123,?R) AND
  price(co123,?P) AND quantity(co123,?Q) AND
  multiply(?P,?Q,?Payment) ;
```

SCLP TextFile Format for (Daml)RuleML

```
payment(?R,base,?Payment) <-
http://xmlcontracting.org/sd.daml#result(co123,?R) AND
price(co123,?P) AND quantity(co123,?Q) AND
multiply(?P,?Q,?Payment) ;
```

```
<drm:imp>
  <drm:_head> <drm:atom>
     <drm:_opr><drm:rel>payment</drm:_opr></drm:rel>    <drm:tup>
     <drm:var>R</drm:var> <drm:ind>base</drm:ind> <drm:var>Payment</drm:var>
  </drm:tup></drm:atom> </drm:_head>
  <drm:_body>
   <drm:andb>
     <drm:atom> <drm:_opr>
       <drm:rel href= "http://xmlcontracting.org/sd.daml#result"/>
         </drm:_opr> <drm:tup>
           <drm:ind>co123</drm:ind> <drm:var>Cust</drm:var>
          </drm:tup> </drm:atom>

... </drm:andb> </drm:_body>  </drm:imp>
```

drm = namespace for damlRuleML

Example Contract Proposal, Continued: lateDeliveryPenalty exception handler module

```
lateDeliveryPenalty_module {
// lateDeliveryPenalty is an instance of PenalizeForContingency
//   (and thus of AvoidException, ExceptionHandler, and Process)
http://xmlcontracting.org/pr.daml#PenalizeForContingency( lateDeliveryPenalty) ;
// lateDeliveryPenalty is intended to avoid exceptions of class
// LateDelivery.
http://xmlcontracting.org/sd.daml#avoidsException(lateDeliveryPenalty,
  http://xmlcontracting.org/pr.daml#LateDelivery);
// penalty = - overdueDays * 200 ; (negative payment by buyer)
<lateDeliveryPenalty_def> payment(?R, contingentPenalty, ?Penalty) <-
  http://xmlcontracting.org/sd.daml#specFor(?CO,?PI) AND
  http://xmlcontracting.org/pr.daml#hasException(?PI,?EI) AND
  http://xmlcontracting.org/pr.daml#isHandledBy(?EI,lateDeliveryPenalty) AND
  http://xmlcontracting.org/sd.daml#result(?CO,?R) AND
  http://xmlcontracting.org/sd.daml#exceptionOccurred(?R,?EI) AND
  shippingDate(?CO,?CODate) AND shippingDate(?R,?RDate) AND
  subtract(?RDate,?CODate,?OverdueDays) AND
  multiply(?OverdueDays, 200, ?Res1) AND multiply(?Res1, -1, ?Penalty) ;
}
<lateDeliveryPenaltyHandlesIt(e1)> // specify lateDeliveryPenalty as a handler for e1
 http://xmlcontracting.org/pr.daml#isHandledBy(e1,lateDeliveryPenalty);
```

END of SweetDeal OPTIONAL SLIDES

ALSO RELEVANT ARE SLIDES from 3/20/03 SWSL telecon "Overview of Semantic Web Services" by Benjamin Grosof

