



## Semantic Web Services (SWS) for Activity-Based Computing

E. Michael (Max) Maximilien  
<http://maximilien.org>

With

Thomas P. Moran and Alex Cozzi  
Almaden Research Center (USER group)



## Agenda

- **Motivations**
  - SWS and Activity-Based Computing
  - Preliminary research questions
- **Use case**
- **Ontologies**
- **Architecture**
  - Details and demo
- **Discussion**
- **Future research**

## Motivations

- **SWS**
  - Annotate Web services with semantic information
  - Facilitate usage of services, i.e., automatic (or semi-automatic) discovery, selection, invocation, and composition of Web services
  - In line with the vision of the semantic Web
- **Activity-Based Computing**
  - Representation for human activities or *every day processes*
  - Embeds contexts in activity representation as semantic information
  - Thesis is that human realization of work is loose, malleable, and flexible
    - Closer to activity-based computing
    - Far from typical, rigid, and mechanical workflow representation of work

## Motivations (*cont.*) and initial research questions

- **Standard ontology for activities**
  - Upper ontology for all activities
  - Represents common concepts and relationships
- **Domain-specific ontologies**
  - Capture domain-specific activities and concepts
  - Represent aspects that are unique to domain
  - Basis for type information for activities?
- **Activity as services**
  - API to build activity-based solutions
  - Build on standard Web services stack, i.e., WSDL, SOAP, ...
- **Taking advantage of semantics – “research questions”**
  - How can the semantics of activities be reflected in exposed services?
  - How can activity-based widgets and applications be enhanced using the semantics?

## Use case

- **Reading group activities**
  - Common, shared activity among knowledge workers
  - Participants share reading items
  - Domain ontology is *Reading Documents* and *Reading Group Activity*
  - Implicit goal is to gather knowledge from group that otherwise individual would not realize or discover
- **Integrating discovered properties**
  - Availability of Web services in the domain, e.g., Amazon's E-Commerce Services (ECS) 3.0
  - Adapt Amazon ECS 3.0 with domain ontology – create Amazon SWS
  - Dynamically discover new information to enrich Reading Group Activity applications and widgets

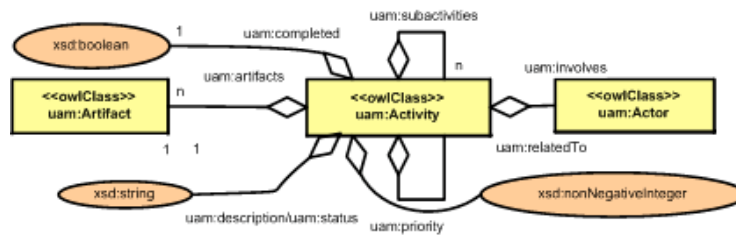
## Ontologies

- **Activity**
  - Unifying concept for things that occur in domain
  - Recursive and fractal in nature
  - Has a collection of *Artifacts* and involves some *Actor(s)*
- **Artifact**
  - Represents all *non-active* elements of an activity
  - Contains references to some resource via the resource's URI
- **Actor**
  - Models the *active* element of an activity
  - Can be both human or software

## Ontologies (cont.)

- **Upper ontology for activities**

- Represented in OWL
- UML diagram showing key concepts and relationships



## Ontologies (cont.)

- **Reading documents ontology**

- Simplified
- Full document ontology is out of scope
- Enough information for interesting widget and SWS

- **Reading Item**

- title, publisher, description, ...

- **Book, BookSection**

- isbn, numberOfPages, ...

- **Article, WebArticle**

- **Author**

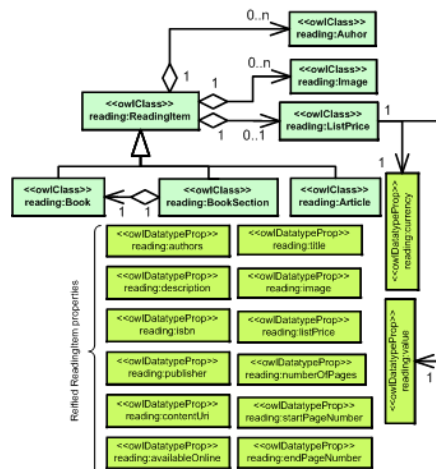
- firstName, lastName

- **Image**

- resourceUri

- **ListPrice**

- currency, value, formattedPrice

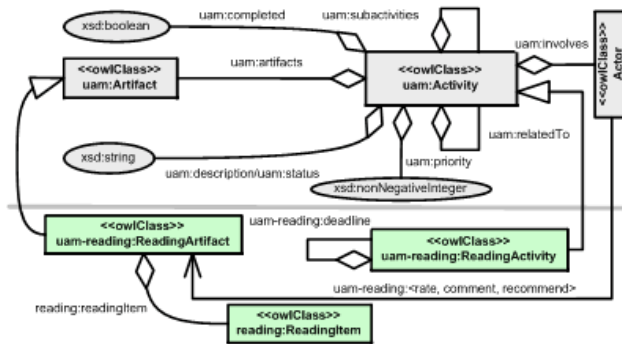


## Ontologies (cont.)

- **Reading group activities**
  - Extends activity ontology
  - Incorporates Reading Document ontology

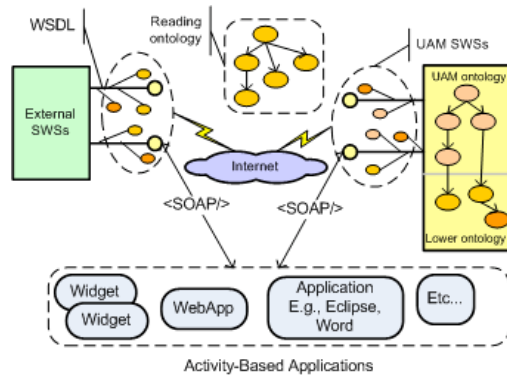
- **Key concepts**

- *ReadingActivity*
  - *readingArtifacts*
- *ReadingArtifact*
  - *readingItem*

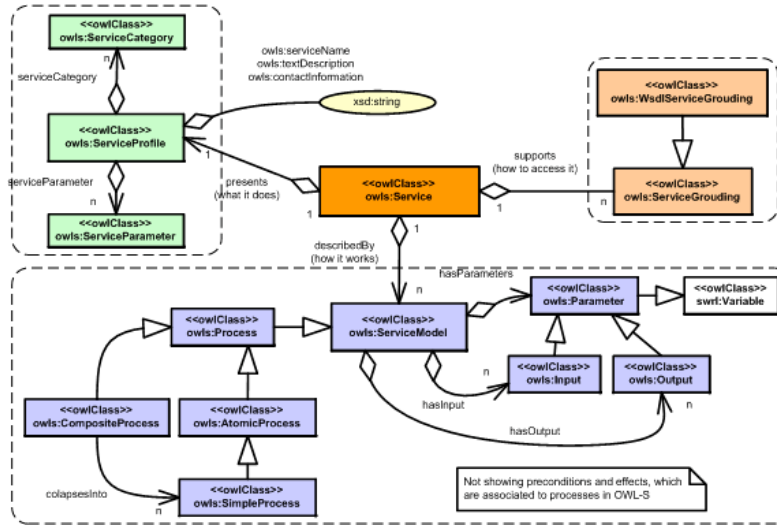


## Architecture – Overview, standards, and implementation

- **Web services standards**
  - WSDL
  - SOAP
  - JAX-RPC
- **Protégé OWL**
- **SWS – OWL-S**
- **Implementation**
  - Java and Python
  - Jena for OWL parsing
  - Axis WS engine
  - Tomcat or WAS
  - ANT

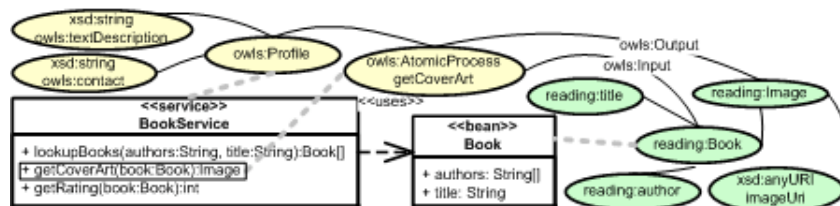


## Architecture – OWL-S service ontology overview



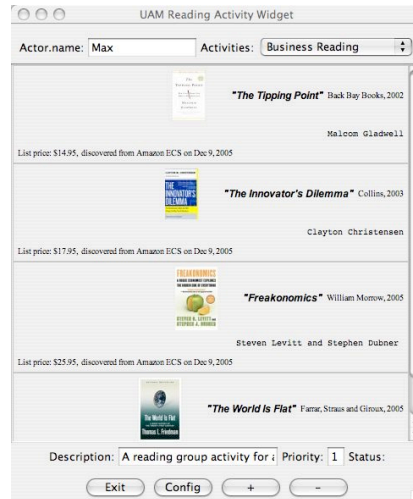
## Architecture – Semantic Web Services (SWS)

- **Hailmary BookService**
  - Proxies Amazon's ECS Web service and adds semantics annotation
- **SWS annotation details**
  - OWL-S individuals for *Service*, *ServiceProfile*, and *ServiceGrounding* (grounded to a WSDL Web service)
  - WSDL operations annotated with *AtomicProcess* with correct *Input* and *Output*
  - *Input* and *Output* annotated with domain ontology concepts



## Architecture – Reading Group widget example

- **User interface**
  - Simple *ReadingActivity* list
  - Configure *Actor* and SWS server
  - Ability to add and remove *ReadingArtifacts*
  - Discovery of properties
- **Implementation**
  - Jython (Python in Java)
  - Leverages Python's scripting prowess
  - Leverages Python's meta programming facilities, OO support, and general *coolness* 😊



## Discussion – Limitations

- **Domain ontology**
  - Exists and in use
  - This is a big assumption
- **Scalability**
  - How to discover multiple SWS in one domain?
  - How to display many discovered properties?
  - Ranking discovered properties?
- **Complete dynamic SWS generation**
  - Domain ontology and operator ontology (or partial OWL-S *ServiceProfile* definition)
  - Generate complete code and descriptions for SWS
- **Back-end of Hailmary is currently a flatfile**
  - Use a RDF DB-based back end with support for transactions and so on
- **Support for query language, i.e., XQuery, in find<Xyz>(…) services**

## Future research

- **WSDL-S**
  - Simpler and more pragmatic SWS description, uses WSDL extensions
  - Good tool support (Eclipse plug-in)
  - Ontology language agnostic (supports OWL and UML)
  - Missing standard service ontology?
- **Other use cases and examples**
  - Implement other use cases
  - Web application example
  - Other applications or widgets
- **Matching algorithm**
  - *AtomicProcess* chaining to generate some output?
  - *CompositeProcess* or some BPEL-like composition language?
- **Is there a hybrid, ontology and folksonomy (i.e., <http://del.icio.us>, Yahoo! Flykr), approach that may work best?**
  - Address limitation of existing and agreed ontology
  - Enables some level of automation

## References

Maximilien, E. M., et al. “Semantic Web Services for Activity-Based Computing”, In Proceedings of ICSOC 2005

Moran, T.P., et al. “Unified Activity Management”, Communications of the ACM, Dec. 2005

Sycara, K., et al. “Automated Discovery, Interaction, and Composition of Semantic Web Services”, Journal on Web Semantics, Sept. 2003

Berners-Lee, T. et al. “The Semantic Web”, Scientific American, May 2001

OWL-S 1.0, <http://www.daml.org/services/owl-s/1.0/>

WSDL-S, <http://www.w3.org/Submission/WSDL-S/>

Amazon ECommerce Services, <http://www.amazon.com/aws>

# Acknowledgements



<http://python.org>



<http://protege.stanford.edu>



धन्यवाद  
Hindi

多謝  
Traditional Chinese

ขอบพระคุณ  
Thai

Спасибо  
Russian

Gracias  
Spanish

شكراً  
Arabic

Thank You  
English

Obrigado  
Brazilian Portuguese

Grazie  
Italian

Danke  
German

Merci  
French

多谢  
Simplified Chinese

நன்றி  
Tamil

ありがとうございました  
Japanese

감사합니다  
Korean

## [Backup] Architecture – Matching algorithm sketch

```
[...]  
hServices = ... # list of domain Hailmary services  
services = ... # get from registry of SWS (currently fixed)  
for s in services:  
    if s.owlS.profile.sParameter.uri == self.domainOnto:  
        props = self.matchProperties( s, hServices )  
        self.displayProps( props ) # displays DiscoveredProp  
[...]  
  
def matchProperties( self, service, hServices ):  
    props = {}  
    for s in hServices:  
        self.findMatchingProperties( service, s, props )  
    return props
```

## [Backup] Architecture – Matching algorithm sketch (cont.)

```
def findMatchingProperties( self, service, hService, props ):  
    for atomicProcess in service.atomicProcesses:  
        matched = 0  
        inputs = atomicProcess.inputs  
        params = []  
        for output in hService.profile.outputs: # decomposed output  
            if self.matchInsOut( inputs, output ):  
                inputs.remove( output )  
                params.append( output )  
        if len( inputs ) == 0:  
            dProp = DiscoveredProp(atomicProcess, params, hService)  
            props[ atomicProcess.name ] = dProp
```

## [Backup] Architecture – Matching algorithm sketch (cont.)

```
def matchInsOut( self, ins, out ):  
    for i in ins:  
        if out.class == i.class or i.class.subsumes( out.class ):  
            return 1  
    return 0
```

### ▪ Example Matching

- BookService SWS
  - BookService.getCoverArt( Book.isbn ) : Image
  - BookService.getListPrice( Book.isbn ) : ListPrice
- Hailmary SWS outputs
  - FindReadingWs.findBook(...) : Book
  - CreateReadingActivity.createBook( ... ) : Book

## [Backup] Discussion – Other use cases and domains

### ▪ Asset-based IT services

- Service engineers build service solutions using assets
- Asset ontology includes: *HardwareAsset*, *SoftwareAsset*, *PeopleAsset*, *DocumentAsset*, and so on
- Services to match
  - Software asset service, e.g., [sourceforge.net](http://sourceforge.net)
  - Technical document repository service, e.g., [freshmeat.org](http://freshmeat.org), [oreilly.com](http://oreilly.com), and so on
  - Social network sites for developers, e.g., [slashdot.org](http://slashdot.org)

### ▪ Other use cases

- Activities of professional knowledge workers
- Insurance, sales, healthcare, and so on

## Acknowledgements and note on project name

### ▪ Acknowledgements

- UAM Teams (Cambridge and Almaden)
  - Thanks for welcoming me to Research
  - Thanks for the engaging discussions
  - Hope to continue involvement in future
- Tom Moran
- Alex Cozzi, Steve Farrell, and John Tang

### ▪ “Hailmary” name

- A prayer in Catholic church to The Virgin Mary
- In American football, a last minute pass attempt from quarterback to entire receiving team in hope of scoring game winning or tying touchdown... ☺

