Enhancing Semantic Web Services with Inheritance

Simon Ferndriger[1] Abraham Bernstein[1] Jin Song Dong[2] Yuzhang Feng[2] Yuan-Fang Li[3] Jane Hunter[3]

[1]Department of Informatics
University of Zurich, Switzerland

[2]School of Computing
National University of Singapore, Singapore

[3]School of ITEE
University of Queensland, Australia

ISWC, October 2008

Outline

Outline

Motivation

- Currently proposed Semantic Web services allow for semantic annotation of web services
- Facilitates service discovery, invocation, composition, monitoring and more
- OWL-S and other frameworks can support only independent development
  - Expensive to find a suitable alternative or substitute service
  - No concrete, self-contained mechanism of establishing inheritance relationships

Motivating Tasks
Semantic Service Annotation

- Semantic Web Service needs to reach a critical mass for wider acceptance and adoption
- Creation of annotation is an important first task
- Currently, majority is created from scratch
- What if we can selectively reuse components of existing services?
  - Faster and more systematic

Motivating Tasks
Semantic Service Discovery

- Automatic service discovery: Major motivation for Semantic Web services
- Depends heavily on potentially large service registry
  - Sometimes inefficient and unreliable
- What if we can make use of the relationships among different services to find service substitutes?
  - Faster service discovery and composition

Outline

OWL-S

- Developed to enrich Web services with semantics
- Enables automated discovery, invocation, composition, interoperation and monitoring
- Defines an essential set of vocabularies to describe three components of a service
  - Profile: What does the service do?
  - Model: How is the service used?
  - Grounding: How can the service be accessed?

Semantic Web Services Framework (SWSF)

- Includes two components
  - Semantic Web Services Language
    - A generic language to formally specify web services concepts and descriptions
    - Includes two sub-languages: SWSL-FOL and SWSL-Rules
  - Semantic Web Services Ontology
    - Provides semantic specifications of web services
    - Serves essentially the same purpose as OWL-S

Molecules

- A language construct in the Frames layer of SWSL-Rules
- A molecule can be viewed as an atomic term
  - Value molecules: `t[m -> v]`
  - Boolean molecules: `t[p]`
  - Some other forms omitted
  - Complex molecules `t[m -> v and p]`

Outline

Perspectives of Inheritance

Strict IR vs Normal IR and Single IR vs Multiple IR

- Strict IR
  - Related to complete inheritance
  - Inherited information can neither be altered nor extended
- Normal IR
  - Related to default inheritance
  - Inherited information can be modified and extended afterwards
  - New features or functionalities are allowed to be added
  - More flexible compared with the OWL `imports`
- Single IR
  - Allowing a service to inherit from only one service
- Multiple IR
  - Allowing a service to inherit from several super services
  - Added complexity: inconsistency

Syntax Extension

- IRs are modelled in additional, independent service inheritance profiles, a subclass of the OWL-S class `ServiceProfile`
- IRs can be modelled in two directions
  - If specified in both, the inheritance is “endorsed” by the super service

Definition

$$\mathit{InheritanceProfile} \sqsubseteq \mathit{ServiceProfile}$$

$$\mathit{Relationship} = \mathit{SuperService} \sqcup \mathit{SubService}$$

$$\geq 1\ \mathit{contains} \sqsubseteq \mathit{InheritanceProfile}$$

$$\top \sqsubseteq \forall \mathit{contains}.\mathit{Relationship}$$

$$\mathit{SuperService} \sqcap \mathit{SubService} = \bot$$

Modification of Inheritance

Type	IR type	Modification
Customization	strict	process & IDs
Extension	normal	process
Manipulation	normal & strict	IOPEs

Modification of Inheritance

- Customization

  $\texttt{Inherit[AdoptServiceModel(PID}_{\texttt{INHERITED}}\texttt{)]}$
  $\texttt{Renaming[ID}_{\texttt{INHERITED}}\texttt{ *-> ID}_{\texttt{REPLACEMENT}}\texttt{]}$
  $\texttt{ProcessReplacement[PID}_{\texttt{INHERITED}}\texttt{ *-> PID}_{\texttt{REPLACEMENT}}\texttt{]}$

- Extension

  $\texttt{ProcessInsertion[\{after/before\} PPID}_{\texttt{INHERITED}}\texttt{ *-> CCID}_{\texttt{NEW}}\texttt{]}$
  $\texttt{ProcessDeletion[PPID}_{\texttt{INHERITED}}\texttt{]}$

- Manipulation

  $\texttt{ExpressionReplacement[\{}$
  $\texttt{replaceCondition(CID}_1\texttt{) *-> CID}_2\texttt{ or}$
  $\texttt{replaceResult(CID}_1\texttt{) *-> CID}_2$
  $\texttt{\}]}$
  $\texttt{AddInputsAndOutput[addIO(PID, OID) ->* NID]}$
  $\texttt{DeleteInputsAndOutput[deleteIO(PID, OID) ->* NID]}$

Satisfaction Conditions

- Conditions to be satisfied by service modification
- Generally need to be checked by agents using the IRs
- Example
  - If an arbitrary inheritance profile *contains* two super services, and adopts the service model of each of these two services, then the two services are actually the same and so are the two service models
  - When input types of the replacement process are OWL classes, the input types must either be from the same OWL class or from an OWL sub class

Outline

Prototype and Examples

- Prototype developed to implement the IR framework
  - Through a web interface
  - Four main features: service annotation creation, service visualization, service discovery and inheritance validation
- Example: Congo book store extended with IR
  - Both normal and strict IR are used

Examples
Normal IR

- `FullCongoBuy`: An example for buying service for physical books
- Create a service for buying digital books by reusing `FullCongoBuy`

```
Inherit[AdoptServiceModel(FullCongoBuy), Processes].
ProcessesReplacement[LocateBook*->Locate_eBook].
ProcessDeletion[SpecifyDeliveryDetailsPerform].
ProcessInsertion[
after(BuySequence)*->SpecifyDownloadDetailsPerform,
after(SpecifyDownloadDetails)*->ProvideDownloadOptionsPerform]
Renaming[FullCongoBuy*->Full_eBookBuy].
```

Examples
Strict IR

- `ExpressCongoBuy`: book buying service with standard delivery setting
- Create a service `EconomyCongoBuy` with a slower delivery

```
Inherit[AdoptServiceModel(ExpressCongoBuy), Processes ].
Renaming[ExpressCongoBuy *-> EconomyCongoBuy ].
ExpressionReplacement[
Effect(ExpressCongoBuy, ExpressCongoOrderShippedEffect) *->
Effect(EconomyCongoBuy, EconomyCongoOrderShippedEffect)
].
```

Outline

Summary

- Current OWL-S specification lacks a systematic way of creating and discovering services
- Proposed an inheritance relationship between services
  - Modelled in inheritance profiles
  - Language constructs to link services
  - Service customization, extension and manipulation for service modifications
  - Satisfaction conditions for service modifications
  - Prototype implementation

Future Work

- Frame problem
  - Unwanted effects of a procecure/operation resulted from under specification of pre/post conditions
- Validation of satisfaction conditions
  - Computationally intensive
  - re-validation sometimes neccssary
- Prototype improvement
  - Protégé plugin
  - Directly communicates with OWL-S plugin
- Transfer of IR into other service ontologies and languages (SWSO, WSMO and WSDL-S)

THANK YOU