Semantically Extensible Schemas for Web Service Evolution

Erik Wilde
ETH Zürich

http://dret.net/netdret/publications#wil04j

Abstract

Web Services are designed for loosely coupled systems, which means that in many cases it is not possible to synchronously upgrade all peers of a Web Service scenario. Instead, Web Service peers should be able to coexist in different versions. Additionally, older software versions often could benefit from upgrades to the service if they were able to understand it. This paper presents a framework for semantically extensible schemas for Web Service evolution. The core idea of is to use declarative semantics to describe extensions to a service's vocabulary. These declarative semantics can be used by older software versions to understand the semantics of extensions, thus enabling older software to dynamically adapt to newer versions of the service. As long as declarative semantics are sufficient, older software can benefit from the service's extension.

Outline

- The Web Web Service
  - what is semantic extensibility?
- Real-life Example
  - why is semantic extensibility useful?
- Semantics Distribution
  - how are extensions distributed among applications?
- Challenges
  - what is required to implement it?

The Web Web Service

- the Web implements a "Web Service"
  - Request: GET/POST URI, HTTP Headers
  - Response: Resource (in most cases HTML)
- open, extensible and very robust
  - supports diversity among client and server versions
    - because people are more robust than machines
    - because most Web pages are "well-designed"
      - counter-examples: Flash, JavaScript/ActiveX required, ...
- even HTML Forms are robust
  - new elements such as `fieldset` and `legend`
  - new functionality such as Web Forms 2.0 (Opera Flavor)

HTML Example

- HTML pages may contain custom formatting
- different implementations are possible:
  1. every element contains CSS instructions (`style="..."`)
     - hard to maintain, large volume of redundant code
  - grouping elements of a common style (`class="..."`)
    2. including the CSS for the `class` (`<style>`)
       - reuses CSS code for one HTML page
       - redundancy across pages (repeated content of `<style>`)
    3. linking an external CSS (`<link>`)
       - reuses CSS code across HTML pages
       - creates a separate resource that describes the custom formatting
- CSS is just one example for declarative semantics

Example (Semantics)

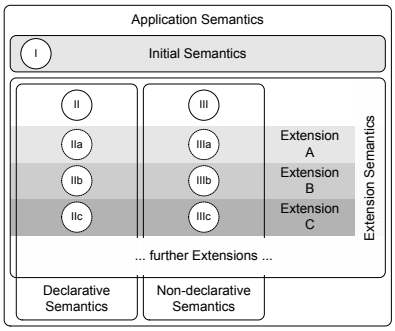
- HTML Extension (Version 1):
```
<div style=" [[ CSS ]] "> ... </div>
```
- HTML Extension (Version 2):
```
<style type="text/css">.textbox { [[ CSS ]] }</style>
    ...
<div class="textbox"> ... </div>
```
- Borderline HTML Extension (depending on textbox.css origin):
```
<link rel="stylesheet" type="text/css" href="textbox.css" />
    ...
<div class="textbox"> ... </div>
```
- perfect HTML Extension:
```
<textbox> ... </textbox>
```


©2004 Erik Wilde

Declarative vs. Non-declarative

- HTML Extension for Square Root Element:
  `Square root rendering looks like <sqrt>2</sqrt>`
- Declarative Formatting Semantics (CSS):
  ```
  sqrt:before { content : "sqrt(" }
  sqrt:after  { content : ")" }
  ```
- Formatting of HTML Extension:
  - Declarative Semantics:
    Square root rendering looks like sqrt(2)
  - Non-declarative Semantics:
    Square root rendering looks like $\sqrt{2}$

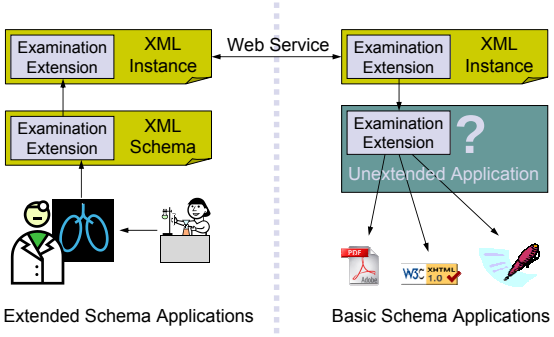
Extension Semantics



Real-life Scenario

- Workflow support for cardiologists
  - a number of examination types
    - 12 different types have been identified
  - a number of participants in the scenario
    - examiner, hospital staff, general practitioner
  - implementation must be open and extensible
    - new examination types may be identified
    - they should be handled by older implementations
      - as good as possible → sometimes they may be ignored
- simple input and output methods are required
  - Web-based input, HTML/PDF report generation

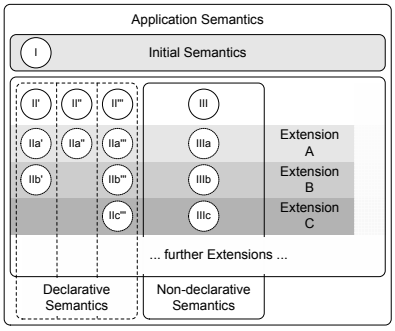
Real-life Problem



Example

- patient records may contain extensions
- extensions should be formatted
  - by using extension semantics for formatting
  - XSLT templates for generating HTML and XSL-FO
- extensions should be modified
  - through extension semantics for input forms
  - XSLT templates for generating Web Forms and XForms
- 4 declarative semantics for extensions
  - every extension must provide this information
  - read-only systems only need the 2 formatting semantics

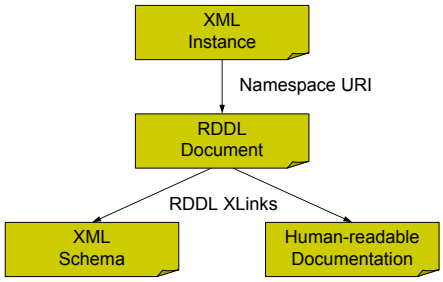
Multiple Declarative Semantics



Extensional vs. Intensional

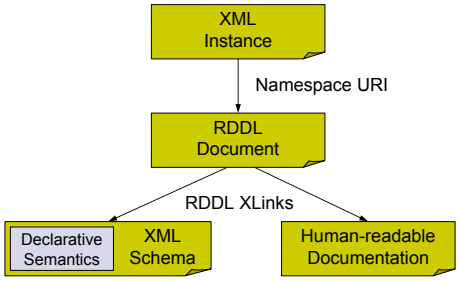
- how are extension semantics communicated?
- Extensional Semantics
  - part of the instance
  - can be interpreted with the instance only
  - SOAP's `mustUnderstand`
- Intensional Semantics
  - part of the extension definition (e.g., the XML Schema)
  - interpretation of an instance requires the schema
  - makes the instances smaller
    - but they are no longer semantically self-describing

Schemas, Namespaces and RDDL



Extensional Semantics

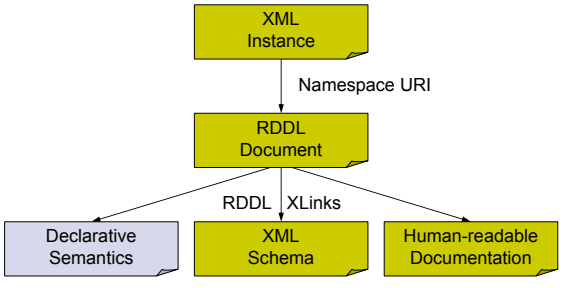
Intensional Semantics



Semantics Distribution

- declarative semantics are part of the XML Schema
  - defined in `xs:annotation/xs:appinfo` elements
  - should be validated by application-specific check
- they must be extracted from the schema
  - error-prone process
  - repeated by every application accessing the schema
- more efficient to publish them separately
  - extracted and published through automated tools
  - simply another resource described by RDDL
  - can be easily accessed and interpreted by applications

Intensional Semantics



The Challenges

- the theory looks easy, what about the real world?

1. identifying the requirements may be hard
   - but it may be worth the effort!
2. declarative semantics may be hard to define
   - but a small subset (`mustUnderstand`) is always possible!
3. the scenario with RDDL looks complicated
   - it is easy to handle and easy to implement!
   - schema versioning will become important anyway!

Thank You! Questions?

- URI for this presentation:
  - `http://dret.net/netdret/publications#wil04j-talk`
- URI for the paper:
  - `http://dret.net/netdret/publications#wil04j`
- my homepage
  - `http://dret.net/netdret/`

©2004 Erik Wilde