# Avoiding WSDL Bad Practices in Code-First Web Services

J. L. Ordiales Coscia C. Mateos M. Crasso A. Zunino

ISISTAN
Instituto de Sistemas Tandil

UNICEN
Universidad Nacional del Centro de la Provincia de Buenos Aires

CONICET

JAIIO - Jornadas Argentinas de Informática e Investigación Operativa
ASSE - Simposio Argentino de Ingeniería de Software

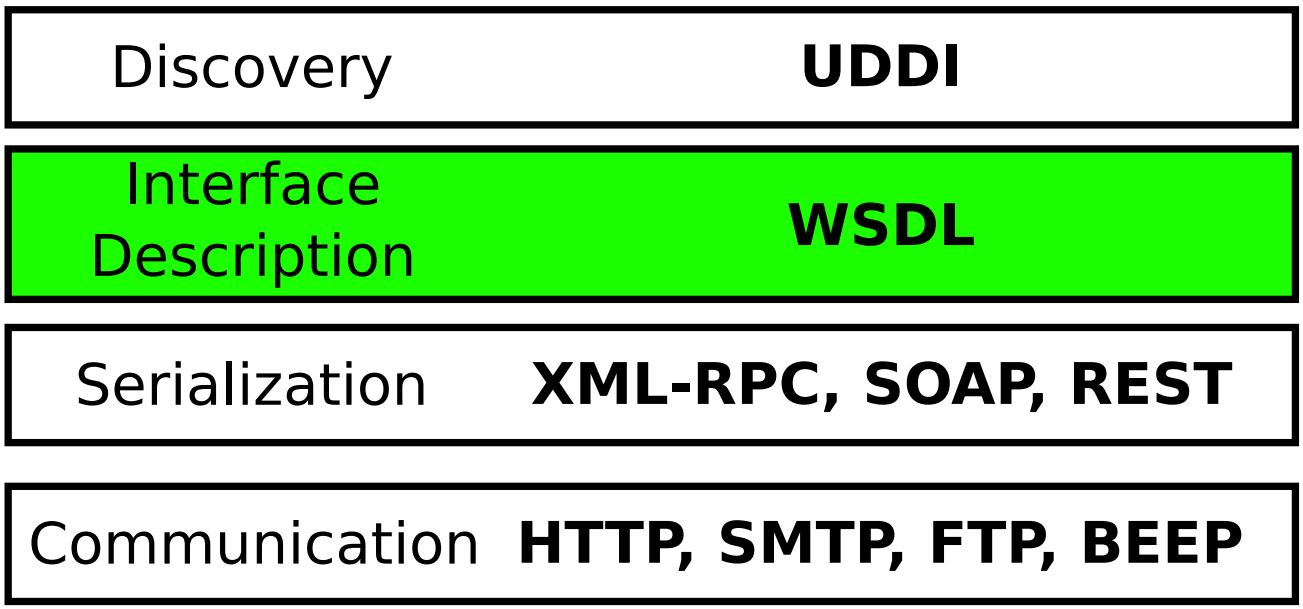
# Outline

# Services: the building block

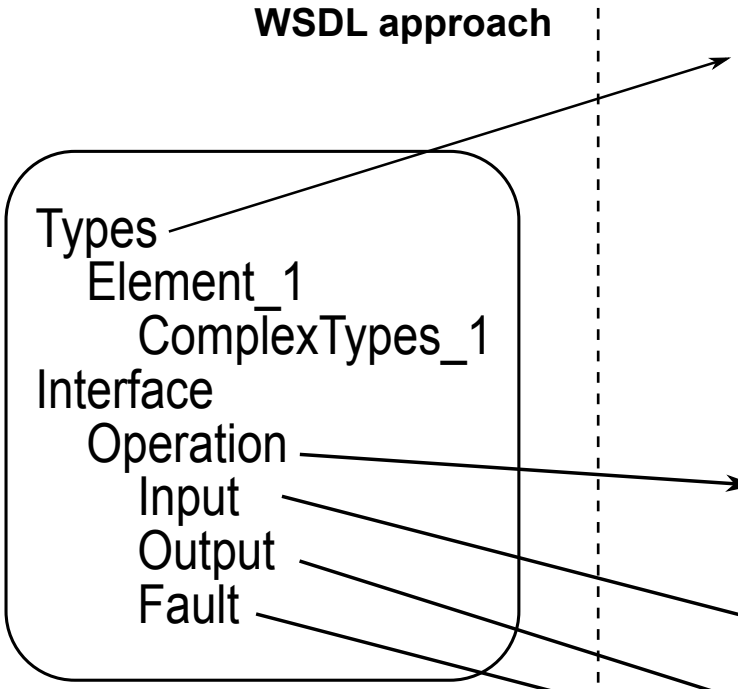
- SOC is a relative new paradigm for distributed computing
- Specific software functionality is organized as a *service*
- A service must have a network-addressable interface
- Applications are build by combining different services even services owned by independent providers

# Implementing SOC over the Internet

- The term Web Services refers to a stack of technologies
- Proposed by W3C consortium



# Describing Web Services interfaces I




**Overview of a WSDL document**

```
<types>
  <xsd:element name="GetRate">
    <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="srcCurrency" type="xsd:string"/>
       <xsd:element name="destCurrency" type="xsd:string"/>
     </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</types>
<message name="GetRateSoapIn">
  <part name="parameters" element="s0:GetRate" />
</message>
<portType name="CurrencywsSoap">
  <operation name="GetRate">
    <documentation>
        This method returns the currency
        conversion ratio between two countries
    </documentation>
    <input message="s0:GetRateSoapIn">
        <documentation>The codes of two countries</documentation>
    </input>
    <output message="s0:GetRateSoapOut" />
    <fault name="nmtoken" message="s0:GetRateFault"/>
  </operation>
</portType>
```

# Describing Web Services interfaces II

- Developers write service interfaces using their preferred programming languages (**Code-First**)
- Language-dependent tools map from C#, Java, C++, Python, etc. to WSDL

$$T : C \rightarrow W$$

$$C = \{M(I_0, R_0), .., M_N(I_N, R_N)\}$$
$$W = \{O_0(I_0, R_0), .., O_N(I_N, R_N)\}$$

*C*: service class
*W*:service WSDL document
*M*: class method; *M(I)*:method input type parameters; *M(R)*: method return type parameters
*O*:operation element; *O(I)*:input message element; *O(R)*: output message element

# Revising WSDL documents: Why?

- WSDL documents that are supposed to describe the API services offer
- Web Services have little value if other cannot discover, access, and make sense of them [1]
- WSDL documents are crucial in enabling third-parties to use services [4]

# Revising WSDL documents: How?

*Best practices for describing Web Services tell us:*

*"Avoid WSDL discoverability anti-patterns"*

- In [4] the authors describe a catalog of usual WSDL document coding practices that affect services descriptiveness, along with corrective courses of actions
- A requirement inherent to remedy anti-patterns is to **have the control of** your WSDL document
- Could you apply "anti-patterns medicine" when lacking of WSDL experts?

# Revising WSDL documents: How?

*Best practices for describing Web Services tell us:*

*"Avoid WSDL discoverability anti-patterns"*

- In [4] the authors describe a catalog of usual WSDL document coding practices that affect services descriptiveness, along with corrective courses of actions
- A requirement inherent to remedy anti-patterns is to **have the control of** your WSDL document
- Could you apply "anti-patterns medicine" when lacking of WSDL experts?

# Could [some|all] WSDL anti-patterns be prevented?

- We study whether well-known Object-Oriented (OO) metrics taken in the code implementing services could be somehow correlated to WSDL anti-patterns occurrences
- We investigate whether employing OO refactorings on service implementation prevent WSDL documents from anti-patterns

# Methodology

1. Get a publicly available data-set of Code-First Web Services
   - 90 services from Merobase, Exemplar, and Google Code, using AXIS Java2WSDL Code-First tool
2. Take OO metrics over service classes
   - Adapted *ckjm* tool for automatically computing the Chidamber-Kemerer's metrics suite among others
3. Count anti-patterns occurrences in WSDL documents from service classes
   - APDetector [3] automatically detects WSDL anti-patterns
4. Analyze correlation between OO metrics and anti-patterns occurrences
   - Spearman's rank correlation coefficient, with OO metrics as independent variables and the anti-patterns as dependent ones

# Correlation Analysis Results

| Anti-pattern/Metric | WMC | CBO | ATC | EPM |
|---|---|---|---|---|
| Ambiguous names | **0.86** | 0.42 | 0.25 | 0.33 |
| Empty messages | 0.54 | 0.20 | 0.19 | **0.99** |
| Enclosed data model | 0.41 | **0.98** | 0.12 | 0.16 |
| Low cohesive operations in the same port-type | **0.61** | 0.38 | 0.12 | 0.39 |
| Redundant data models | **0.79** | 0.33 | 0.15 | 0.31 |
| Whatever types | 0.50 | 0.35 | **0.60** | 0.32 |

- Results in bold are statistically significant at the 5% level, i.e. p-value $< 0.05$

# Refactoring I

- Replacing arguments declared as Object (EPM metric) with a concrete data-type whenever possible and parameters declared as Vector, List, etc., decided to replace the former with array structures; by adapting the Fowler et al.'s Introduce Type Parameter refactoring [2]

## Effects on anti-patterns occurrences

- *Redundant data models* reduced by (↓) 52.73%
- *Whatever types* ↓ 21.09%

# Refactoring II

- Reducing WMC by employing the Move Method refactoring [2], so that on average the metric in the refactored services represented a 70% of the original value

## Effects on anti-patterns occurrences

- *Redundant data models* ↓ 86.66%
- *Low cohesive operations in the same port-type* ↓ 47.26%

# Findings

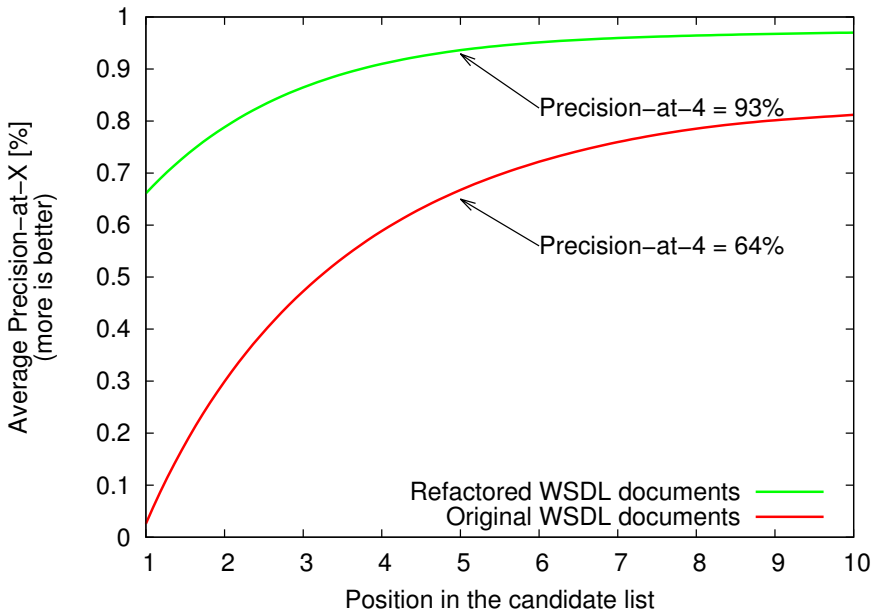
- Some anti-patterns can be prevented to some extent by performing well-known refactorings
  - Most IDEs automatically support employed refactorings
- Two anti-patterns are associated with the employed Code-First tool and cannot be removed by refactoring services classes
  - The *Empty messages* and *Enclosed data model* anti-patterns, which have comparatively little negative impact on service discoverability [4]
- Results are data-set specific and cannot be generalized

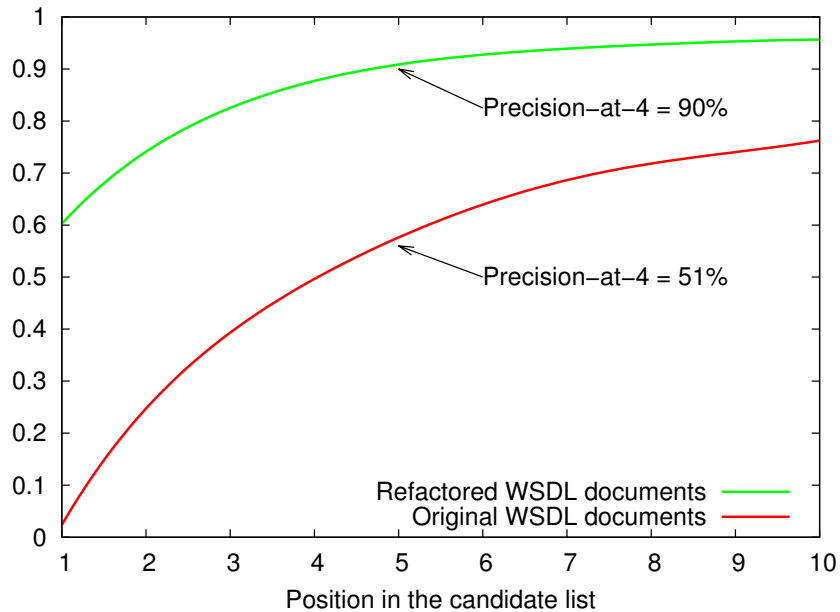
# Future Work *(actually ongoing work)* I

- Eng. José Luis Ordiales Coscia's Magister thesis
  - Focused on methods for facilitating back-end SOC systems development (advisor Phd. C. Mateos, co Phd. M. Crasso)
- We have added more Web Services to the data-set
  - Web Services from a commercial GIS-based logistics system
- We have employed 3 more Code-First tools
  - JavaWS, WSProvide, EasyWSDL

# Future Work *(actually ongoing work)* II

- We have assessed the impact of applying proposed refactorings on service discovery



# References I



Ian Foster.
Service-Oriented Science.
*Science*, 308(5723):814–817, 2005.


Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts.
*Refactoring: Improving the Design of Existing Code*.
Addison-Wesley Professional, 1 edition, July 1999.


Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo.
Automatically detecting opportunities for Web Service descriptions improvement.

# References II

In *Software Services for e-World*, volume 341 of *IFIP Advances in Information and Communication Technology*, pages 139–150, 2010.

Juan Manuel Rodriguez, Marco Crasso, Alejandro Zunino, and Marcelo Campo.
Improving Web Service descriptions for effective service discovery.
*Science of Computer Programming*, 75(11):1001–1021, 2010.

- The EasySOC Project Home Page (Articles, Software prototypes, and more)
  `http://sites.google.com/site/easysoc/`

# Questions...

*Thank you!*