From Program Execution to Automatic Reasoning
Integrating Ontologies into Programming Languages

Alexander Paar
State of the Art

- Widely used object-oriented programming languages include a built-in static type system
- Conceptual framework that makes it particularly easy to
  - design,
  - understand, and
  - maintain object-oriented systems
- In recent years *schema languages* and *ontology languages* emerged
  - Programming language agnostic data types and content models
  - Automatic reasoning about domain model
- Powerful modeling features readily available
- Impedance mismatch makes programming difficult and error-prone
- Facilitate the development of intelligent applications
The Extensible Markup Language (XML)

- Separate formatting of a document from its content
- *General Markup Language* (GML) proposed by Charles Goldfarb, Edward Mosher, and Raymond Lorie in 1967
- Standard General Markup Language (SGML) became an ISO standard in 1986
- In 1999, the Extensible Markup Language (XML) became a W3C standard
- XML is not a language
XML Schema Definition (XSD)

- Published as a W3C recommendation in 2001
- XSD specification comprises two parts
  - XML Schema Part 1: Structures
  - XML Schema Part 2: Datatypes
- Atomic XSD data types are triples
  - Value space
  - Lexical space
  - Fundamental facets
Built-in XSD Data Types

- 19 primitive built-in data types
- Several isomorphic mappings to programming language data types

<table>
<thead>
<tr>
<th>.NET value type</th>
<th>XSD type</th>
<th>.NET value type</th>
<th>XSD type</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.String</td>
<td>xsd#string</td>
<td>System.Int16</td>
<td>xsd#short</td>
</tr>
<tr>
<td>System.Boolean</td>
<td>xsd#boolean</td>
<td>System.UInt16</td>
<td>xsd#unsignedShort</td>
</tr>
<tr>
<td>System.Single</td>
<td>xsd#float</td>
<td>System.Int32</td>
<td>xsd#int</td>
</tr>
<tr>
<td>System.Double</td>
<td>xsd#double</td>
<td>System.UInt32</td>
<td>xsd#unsignedInt</td>
</tr>
<tr>
<td>System.SByte</td>
<td>xsd#byte</td>
<td>System.Int64</td>
<td>xsd#long</td>
</tr>
<tr>
<td>System.Byte</td>
<td>xsd#unsignedByte</td>
<td>System.UInt64</td>
<td>xsd#unsignedLong</td>
</tr>
</tbody>
</table>

- Why are there no isomorphic mappings in general?
User-defined Type „age“

- A valid age is any integer number greater than or equal to zero and less than 110.

```xml
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  <xsd:simpleType name="age">
    <xsd:restriction base="xsd:int">
      <xsd:minInclusive value="0"/>
      <xsd:maxExclusive value="110"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```
XSD Data Types in C#

- Mapping option 1

```csharp
class Person {
    public uint HasAge;
}
```

- Mapping option 2

```csharp
using System.Diagnostics;
internal class Person {
    private uint _HasAge;
    public uint HasAge {
        get { return _HasAge; }
        set { Trace.Assert(value >= 0 && value < 110);
             _HasAge = value; }
    }
}
```
XSD Data Types in C# cont’d

- Mapping option 3

```csharp
using System.Diagnostics;

internal struct uint110 {
    private uint value;
    public static implicit operator uint110(uint value) {
        Trace.Assert(value < 100);
        return new uint110 {value = value};
    }
    public static implicit operator uint(uint110 value) {
        return value.value;
    }
}
```
XSD Data Types in C# cont’d

- Mapping option 4

```csharp
abstract class XsdAnySimpleType {
    protected object value;
}

class XsdUnsignedInt : XsdAnySimpleType {
    public static implicit operator XsdUnsignedInt(uint value) {
        return new XsdUnsignedInt {value = value};
    }

    public static implicit operator uint (XsdUnsignedInt value) {
        return (uint) value.value;
    }
}
```
XSD Constraining Facets

- 12 constraining facets

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>defines the number of units of length</td>
</tr>
<tr>
<td>minLength</td>
<td>defines the minimum number of units of length</td>
</tr>
<tr>
<td>maxLength</td>
<td>defines the maximum number of units of length</td>
</tr>
<tr>
<td>pattern</td>
<td>constrains the lexical space to literals that match a specific pattern</td>
</tr>
<tr>
<td>enumeration</td>
<td>constrains the value space to a specified set of values</td>
</tr>
<tr>
<td>minInclusive</td>
<td>defines the inclusive lower bound of the value space</td>
</tr>
<tr>
<td>minExclusive</td>
<td>defines the exclusive lower bound of the value space</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>defines the exclusive upper bound of the value space</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>defines the inclusive upper bound of the value space</td>
</tr>
<tr>
<td>totalDigits</td>
<td>defines the maximum number of values in the value space</td>
</tr>
<tr>
<td>fractionDigits</td>
<td>defines the minimum difference between values in the value space</td>
</tr>
<tr>
<td>whiteSpace</td>
<td>controls the normalization of string data types (modifying)</td>
</tr>
</tbody>
</table>

- A constraining facet is an optional property that can be applied to an atomic data type to constrain its value space

- A value space $\nu(T)$ is the set of values for a given atomic data type $T$
Constraint Application

- The value space function $\nu(P)$ is defined semantically for each primitive base type $P$ under consideration.
- The value space of a base type $T$ can be restricted by the application of one or more constraints $c_i = \phi(TV) b_i \quad i \in 1..n$.
- A constraint body $b = \{ x | x \in \nu(TV) \} \bigcap_{k \in 1..m} \{ x | x \prec \text{literal}_k \}$ defines the intersection of the value space of $TV$ and those values that satisfy the properties $x \prec \text{literal}_k \quad k \in 1..m$.
- $T_n = T \cdot c_1 \ldots \cdot c_n \equiv \bigcap_{i \in 1..n} c_i$. 
Constraint Subsumption

- **S-CstrVSpace** (non-algorithmic)
  \[
  \nu(c_1\{\{TV \leftarrow T\}\}) \subseteq \nu(c_2\{\{TV \leftarrow T\}\})
  
  c_1 \leq:: c_2
  \]

- **S-CstrWidth**
  \[
  \phi(TV)\{x| x \in \nu(TV)\} \bigcap_{i \in 1..n+k} \{x| x \prec y_i\} \leq:: \phi(TV)\{x| x \in \nu(TV)\} \bigcap_{i \in 1..n} \{x| x \prec y_i\}
  \]

- **S-CstrDepth**
  for each \(i\)
  \[
  \phi(TV)\{x| x \in \nu(TV)\} \bigcap_{i \in 1..n} \{x| x \prec y_i\} \leq:: \phi(TV)\{x| x \in \nu(TV)\} \bigcap_{i \in 1..n} \{x| x \prec z_i\}
  \]
Subtyping of XSD Data Types

- **S-VSPACE** (non-algorithmic)
  \[ \nu(S) \subseteq \nu(T) \]
  \[ S <: T \]

- **S-WIDTH**
  \[ S = \bigcap_{i=1..n+k} c_i \]
  \[ U = \bigcap_{i=1..n} c_i \]
  \[ S <: U \]

- **S-DEPTH**
  For each \( i \), \( c_i <: d_i \)
  \[ S = \bigcap_{i=1..n} c_i \]
  \[ U = \bigcap_{i=1..n} d_i \]
  \[ S <: U \]
An XSD Type System

- Interpretation of XSD data types as computational structures
- Foundation for the integration into programming language type system
- Will it be sound (will there be progress and preservation)?
- Yes, it will because...
  - XSD type construction does not pertain to programming language typing and evaluation rules.
  - Subsumption property (if $S' <: S$ and $S <: T$ then $S' <: T$) can be easily proved.
- Practical implementation in the Zhi# programming language
Initial Knowledge Representation Schemes

- In the 1960s, network-based notations occurred.

![Diagram of a Knowledge Graph]

- Translation of semantic networks into fragments of first-order predicate calculus (1980s).

  Philosophers are men.

  Therefore philosophers’ thoughts are thoughts of men.
Description Logics (DL)

- Decision procedures of DL systems always terminate
- Efficient query answering
- Define the concepts of a domain of discourse
- Use concepts to specify properties of objects
- General knowledge contained in the $TBox$
- Contingent knowledge contained in the $ABox$
- Fundamental inference is subsumption: $C \sqsubseteq D$
The \( \mathcal{AL} \) Description Logic

- **Basic Description Logic introduced 1991**

<table>
<thead>
<tr>
<th>Constructor Name</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic concept</td>
<td>( A )</td>
<td>( A^\mathcal{I} \subseteq \triangle^\mathcal{I} )</td>
</tr>
<tr>
<td>top level concept</td>
<td>( \top )</td>
<td>( \top^\mathcal{I} = \triangle^\mathcal{I} )</td>
</tr>
<tr>
<td>bottom concept</td>
<td>( \bot )</td>
<td>( \bot^\mathcal{I} = {} )</td>
</tr>
<tr>
<td>atomic negation</td>
<td>( \neg A )</td>
<td>( (\neg A)^\mathcal{I} = \triangle^\mathcal{I} \setminus A^\mathcal{I} )</td>
</tr>
<tr>
<td>intersection</td>
<td>( C_1 \cap C_2 )</td>
<td>( (C_1 \cap C_2)^\mathcal{I} = C_1^\mathcal{I} \cap C_2^\mathcal{I} )</td>
</tr>
<tr>
<td>value restriction</td>
<td>( \forall R. C )</td>
<td>( (\forall R. C)^\mathcal{I} = {x</td>
</tr>
<tr>
<td>ltd. ex. quantification</td>
<td>( \exists R. \top )</td>
<td>( (\exists R. \top)^\mathcal{I} = {x</td>
</tr>
</tbody>
</table>

- Set theoretic interpretation \( \mathcal{I} = (\triangle^\mathcal{I}, \cdot^\mathcal{I}) \)
A Business Meeting Scenario

- Atomic concepts: *Person*, *Meeting*
- General concept inclusion: *Employee* ⊑ *Person*
- Ltd. ex. quantification:
  \[ \text{MeetingParticipant} \equiv \text{Person} \sqcap \exists \text{attendsMeeting}. \top \]
- Value restriction:
  \[ \text{MarketingMeeting} \equiv \text{Meeting} \sqcap \forall \text{hasTopic}. \text{MarketingTopic} \]
- Negation: *Guest* ≡ *Person* ⋈ ¬*Employee*
Reasoning With The $SHOIN$ DL

- $SHOIN$ extends $AL$ with additional modelling features: negation ($C$), qualified number restrictions ($Q$), role hierarchies ($H$), and inverse ($I$) and transitive roles ($R^+$)
- $SHOIN(D)$ includes a concrete domain: data types
Reasoning With The SHOIN DL

- \textit{SHOIN} extends \textit{AL} with additional modelling features: negation (\(\mathcal{C}\)), qualified number restrictions (\(\mathcal{Q}\)), role hierarchies (\(\mathcal{H}\)), and inverse (\(\mathcal{I}\)) and transitive roles (\(\mathcal{R}^+\))
- \textit{SHOIN(D)} includes a concrete domain: data types
Reasoning With The {$\text{SHOIN}$} DL

- {$\text{SHOIN}$} extends {$\mathcal{AL}$} with additional modelling features: negation ({$\mathcal{C}$}), qualified number restrictions ({$\mathcal{Q}$}), role hierarchies ({$\mathcal{H}$}), and inverse ({$\mathcal{I}$}) and transitive roles ({$\mathcal{R}^+$})
- {$\text{SHOIN}(\mathcal{D})$} includes a concrete domain: data types
API-based Ontology Management

- No programming language integration
- Insufficient compiler/IDE support
- Jena Semantic Web Framework widely used

```java
public class Program {
    public static void main(String[] args) {
        String eval = "http://www.zhimantic.com/eval#";

        OntModel m = ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC);
        m.read(new FileInputStream("Evaluation.owl"), "");

        Individual a = m.getOntClass("http://www.w3.org/2002/07/owl#Thing").createIndividual(eval + "a");

        Individual o = m.getOntClass("http://www.w3.org/2002/07/owl#Thing").createIndividual(eval + "o");

        a.addProperty(m.getObjectProperty(eval + "R"), o);

        for (Iterator it = m.getOntClass(eval + "B").listInstances(); it.hasNext();)
            System.out.println(((Individual) it.next()).getURI());

        o.addProperty(m.getDataTypeProperty(eval + "T"),
                m.createTypedLiteral("23", "http://www.w3.org/2001/XMLSchema#positiveInteger"));
    }
}```
Ontologies vs. Object-Orientation

- In Java/C# properties are class members
- In DL ontologies...
  - ...properties form a separate hierarchy (i.e. property centric modeling)
  - ...property domain and range restrictions are used for ontological reasoning
- Description Logics make the *open world assumption*
- Description Logics do not make the *unique name assumption*
The Zhi# („Semantic C#“) Approach

- OWL Compiler Plug-In
- CHIL OWL API
- OWL DL
- Zhi# Compiler Framework
- C#
- .NET
- XSD Compiler Plug-In
- λC-Calculus
- XSD
XSD Data Types in Zhi# Programs

• Precise static typing and type inference for XSD data types

```csharp
using System;
import XML xsd = http://www.w3.org/2001/XMLSchema;
import XML app = http://www.example.com/datatypes;
class C {
    public void f() {
        int i = new Random().Next(); // i : int
        if (i < 110)
            #app#age a = i; // a : int{>= 0}{< 110}, i : int{< 110} Error!
    }
}
```

• Implement OCL expressions with XSD-like data types

<table>
<thead>
<tr>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td>-name : String</td>
</tr>
<tr>
<td>-age : Integer</td>
</tr>
<tr>
<td>-eMail : String</td>
</tr>
</tbody>
</table>

**context Person**

```plaintext```
inv age >= 0 && age < 110
```

```plaintext```
inv name.size() < 40
```
Ontologies in Zhi# Programs

- Combination of static typing and dynamic checking
- Full support for datatype properties

```csharp
using System;
import XML xsd = http://www.w3.org/2001/XMLSchema;
import OWL ont = http://www.zhimantic.com/ont;

class C {
    public DateTime f (#ont#Meeting m, #xsd#dateTime xdt) {
        m.#ont#beginsAt = xdt;
        #xsd#duration xd = "P0Y0M0DT1H30M0S";
        m.#ont#endsAt = xdt + xd;

        #ont#Person alice = new #ont#Person("Alice");
        #ont#Person bob = new #ont#Person("Bob");
        #ont#Person charlie = new #ont#Person("Charlie");

        m.#ont#hasParticipant = bob;
        charlie.#ont#participatesIn = m;
        alice.#ont#moderates = m;

        foreach (#ont#Participant p in m.#ont#hasParticipant)
            Console.WriteLine(p + " participates in " + m);

        if (m is #ont#ImportantMeeting)
            Console.WriteLine(m + " is an important meeting");

        return (DateTime) m.#ont#endsAt;
    }
}
```
Conclusion

- Schema and ontology languages can be integrated in object-oriented programming language
- Object-oriented notation sufficient
- Separation of concerns without impedance mismatch
- Applications become intelligent
- Pay as you go
- Programming languages need to become aware of information spaces
- First Workshop on Programming the Semantic Web
  www.inf.puc-rio.br/~psw12/
Contact

Alexander Paar
alexpaar@acm.org
www.alexpaar.de

The Zhi# Approach
zhisharp.sourceforge.net

Acknowledgements
I kindly acknowledge the support of my employer TWT GmbH Science and Innovation in making my participation possible.
www.twt-gmbh.de