INFORMATION TECHNOLOGY
FOR
DISTRIBUTED ENGINEERING SYSTEMS

Jochen Hanff
Institut für Bauingenieurwesen
FG Theoretische Methoden der Bau- und Verkehrstechnik
TU Berlin
• Distributed environments and applications
• Exchange of data
  o XML
• Remote objects
  o RMI
  o CORBA
• Object structures and algorithms in engineering
  o Dependencies of objects
  o Sequence of modification
• Conclusion
Object oriented method

- An application consists of **structured sets of objects**
- Each object consists of **methods** and **attributes**
  - The methods describe the behaviour of an object
  - The attributes contain the data which is stored in an object
Distributed environments

- Exchange of **data**
  - Remote method invocation
  - Exchange of files
- Using remote **functionality**
  - Remote method invocation
Exchange of Data

Extensible Markup Language (XML)

- XML is a document processing standard proposed by the World Wide Web Consortium (W3C)
- XML is a meta-language that allows you to create your own document markups
- XML documents are similar to HTML documents (in fact, HTML can be defined in XML)
Overview of an XML document

- **XML** Document (required)
- **Document Type Definition** (DTD) (optional)
- **Stylesheet** (XSL, XSLT) (optional)
Example of a simple XML document

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE objects SYSTEM "objects.dtd">
<!-- this is a comment -->
<objects>
  <point id="x1" x="1.0" y="1.0"/>
  <point id="x2" x="3.0" y="1.5"/>
  <point id="x3" x="2.0" y="2.0"/>
  <triangle id="x4">
    <p1>x1</p1>
    <p2>x2</p2>
    <p3>x3</p3>
  </triangle>
</objects>
```
Example of a simple XML document

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE objects SYSTEM "objects.dtd">
<!-- this is a comment -->
<objects>
  <point id="x1" x="1.0" y="1.0"/>
  <point id="x2" x="3.0" y="1.5"/>
  <point id="x3" x="2.0" y="2.0"/>
  <triangle id="x4">
    <p1>x1</p1>
    <p2>x2</p2>
    <p3>x3</p3>
  </triangle>
</objects>
```
Example of a simple XML document

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE objects SYSTEM "objects.dtd">
<!-- this is a comment -->
<objects>
  <point id="x1" x="1.0" y="1.0"/>
  <point id="x2" x="3.0" y="1.5"/>
  <point id="x3" x="2.0" y="2.0"/>
  <triangle id="x4">
    <p1>x1</p1>
    <p2>x2</p2>
    <p3>x3</p3>
  </triangle>
</objects>
```
Example of a simple XML document

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE objects SYSTEM "objects.dtd">
<!-- this is a comment -->
<objects>
  <point id="x1" x="1.0" y="1.0"/>
  <point id="x2" x="3.0" y="1.5"/>
  <point id="x3" x="2.0" y="2.0"/>
  <triangle id="x4">
    <p1>x1</p1>
    <p2>x2</p2>
    <p3>x3</p3>
  </triangle>
</objects>
```
Hierarchy:

objects

point, id = x1, x = 1.0, y = 1.0

point, id = x2, x = 3.0, y = 1.5

point, id = x3, x = 2.0, y = 2.0

triangle, id = x4

x1

x2

x3
Document Type Definition (DTD) : 'objects.dtd'

<!ELEMENT objects (triangle*,point*)>

<!ELEMENT point EMPTY>
<!ATTLIST point id ID #REQUIRED
  x CDATA #REQUIRED
  y CDATA #REQUIRED>

<!ELEMENT triangle (p1,p2,p3)>
<!ATTLIST triangle id ID #REQUIRED>

<!ELEMENT p1 (#PCDATA)>
<!ELEMENT p2 (#PCDATA)>
<!ELEMENT p3 (#PCDATA)>
Extensible Stylesheet Language (XSL)
Extensible Stylesheet Language Transformations (XSLT)

- XSL is used to describe how an XML source document is transformed into another document
- XSL documents are themselves XML documents
South Africa, Johannesburg, 27 mar. 2003
Extensible Stylesheet Language Transformations (XSLT)
Extensible Stylesheet Language (XSL)

Points

<table>
<thead>
<tr>
<th>ID</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>x2</td>
<td>3.0</td>
<td>1.5</td>
</tr>
<tr>
<td>x3</td>
<td>2.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Triangles

<table>
<thead>
<tr>
<th>ID</th>
<th>Point 1</th>
<th>Point 2</th>
<th>Point 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>x4</td>
<td>x1</td>
<td>x2</td>
<td>x3</td>
</tr>
</tbody>
</table>
Advantages of XML

- structured data
- standardized description of structure
- tools available
  (parser, xslt processor, xml validator ... )

Disadvantages of XML

- File size
- Increased processing time
Using remote functionality

double multiply(double a, double b) {
  double ret = a * b;
  return ret;
}
Client/server model

client

request

network

server

reply
Using remote functionality

JAVA Remote method invocation (RMI)

- Available since JAVA 1.1
- RMI is a mechanism that enables an object on one Java virtual machine to invoke methods on an object in another Java virtual machine
JAVA Remote method invocation (RMI)

1. Define a JAVA interface which inherits `java.rmi.Remote`

3. Define a server class which inherits `java.rmi.server.UnicastRemoteObject` and which implements your interface

4. Generate stubs/skeletons using `rmic`

5. Bind object to JAVA registry
Class hierarchy for remote objects

```
java.rmi.server.RemoteObject
    ↓
java.rmi.server.RemoteServer
    ↓
java.rmi.server.UnicastRemoteObject
    ↓
Server
```

```
"interface"
java.rmi.Remote
    ↓
"interface"
ServerInterface
```

`Server` extends `java.rmi.server.UnicastRemoteObject` and implements `ServerInterface`.
Client
Remote
method()
Stub

Registry
bind
lookup

Server
RemoteObject
method()
Skeleton

South Africa, Johannesburg, 27 mar. 2003
Advantages of JAVA RMI

- Easy to use
- No additional interface description language

Disadvantages of JAVA RMI

- Limited to JAVA platform
Common Object Request Broker Architecture (CORBA)

- Specified by the Object Management Group (OMG)
- Platform for distributed object oriented applications in heterogeneous environments
- Language bindings for C++, JAVA, Smalltalk ...
- Commercial and free implementations
Parts of CORBA

- Interface Definition Language (IDL)
- ORB Core
- ORB Interface
- Static interfaces (stubs, skeletons)
- Dynamic Interfaces (Dynamic Skeleton Interface, Dynamic Invocation Interface)
- Object Adapters (Basic Object Adapter, Portable Object Adapter)
- Inter-ORB protocols
Advantages of CORBA

- Platform independent
- Language bindings for several languages
- Standardized services (notification, life cycle, trading ...)

Disadvantages of CORBA

- Increased overhead
- Performance (?)
Dependencies of objects

\[ r.\text{area} = r.\text{height} \cdot r.\text{width} \]
\[ r.\text{height} = p_2.y - p_1.y \]
\[ r.\text{width} = p_2.x - p_1.x \]
Object structures and algorithms in engineering

- High complexity of objects
- Relations to other objects are manifold
- Complex algorithms with high computational effort

This leads to:

- Delayed updates
- Correct sequence of modification of attributes has to be determined

⇒ Graph theory
\[ B := \{ (a, b) \in A \times A \mid a \text{ binds } b \} \]

\[ B = \{ (a, d), (a, c), (d, c), (b, d), (e, b) \} \]
\[
\begin{align*}
H := \{ (a, b) \in A \times A \mid b \text{ depends on } a \} \\
H &= B \cup B^2 \cup \ldots \cup B^{n-1}
\end{align*}
\]
Sequence of modification

Topological sorting

Attribute \(c\) has to be updated.
Correct sequence of modification: \(<e, a, b, d, c>\)
System

1. Elements \( E : \text{Set of elements} \)
2. Properties \( F : \text{Set of properties} \)
3. Algorithms \( (f_1, \cdots, f_m) \longrightarrow A \longrightarrow (h_1, \cdots, h_n) \)
   \( A : \text{Set of algorithms} \)
4. Complete binding relation \( B := (A, F, R_e, R_a) \)
   \( R_e \subseteq F \times A \quad \text{Input parameters, algorithms} \)
   \( R_a \subseteq A \times F \quad \text{Algorithms, output parameters} \)
5. System \( S := (E, A, F; B) \)
Update of a system

Choosing properties which have to be updated:

\[ Z := \{ a \in F \mid a \text{ is choosen for an update} \} \]

Domain of properties:

\[ D_F := \{ a \in F \mid a \text{ has to be updated} \} \]

Domain of algorithms:

\[ D_A := \{ A \in A \mid \text{Algorithm } A \text{ must be invoked for updating an element in } D_F \} \]

Domain of update:

\[ D = D_A \cup D_F \]
Reduced binding relation

\[ \overline{B} = \Gamma^T B \Gamma \]

\[ \overline{B} := \{(a, b) \in D \times D \mid (a, b) \in B\} \]
Removing virtual properties

\[\bar{B} = \{(a, A), (A, b), (A, c), (b, C), (b, B), (c, B), (C, e), (B, d)\}\]

\[D = \{a, A, b, c, C, B, e, d\}\]
Removing virtual properties

\[ \bar{B}_v = \bar{B} \setminus \{(A, b), (b, C'), (c, B), (b, B)\} \cup \{(a, C'), (a, B)\} \]
\[ = \{(a, A), (a, C'), (a, B), (A, c), (c, B), (C, e), (B, d)\} \]

\[ D_v = \{a, A, c, C, B, e, d\} \]
Removing methods without successors

\[ \bar{B} = \{(a, A), (A, b), (b, B), (B, c)\} \]

\[ D = \{a, A, b, B, c\} \]
Removing methods without successors

\[ \bar{B}_v = \{(a, A), (a, B), (B, c)\} \]

\[ D_v = \{a, A, B, c\} \]
Removing methods without successors

\[ \bar{B}_{v,m} = \{(a, B), (B, c)\} \]
\[ D_{v,m} = \{a, B, c\} \]
Sequence of modification

Calculation of $\bar{\mathcal{B}}_A$

$$\bar{\mathcal{B}}_A = \Gamma^T_A \bar{\mathcal{B}}^2_v, m \Gamma_A$$

Topological sorting

$\bar{\mathcal{B}}_A \rightarrow a$

Invoking methods

$$a = \langle B, D, A, ... \rangle$$
Example

- **Rectangle**
  - \( h : \text{double} \)
  - \( a : \text{double} \)
  - get_width() : double
  - compute_height() : void
  - compute_area() : void

- **Point**
  - \( x : \text{double} \)
  - \( y : \text{double} \)
Objects \[ E = \{ r, p_1, p_2 \} \]

Properties \[ F = \{ p_1.x, p_1.y, p_2.x, p_2.y, \]

\[ r.pointA, r.pointB, r.width, r.h, r.a \} \]

Algorithms \[ A = \{ r.get\_width(), r.compute\_height(), r.compute\_area() \} \]

\[(r.pointA.x, r.pointB.x) \rightarrow \text{r.get\_width()} \rightarrow (r.width)\]

\[(r.pointA.y, r.pointB.y) \rightarrow \text{r.compute\_height()} \rightarrow (r.h)\]

\[(r.h, r.width) \rightarrow \text{r.compute\_area()} \rightarrow (r.a)\]
<table>
<thead>
<tr>
<th></th>
<th>p1.x</th>
<th>p1.y</th>
<th>p2.x</th>
<th>p2.y</th>
<th>r.pointA</th>
<th>r.pointB</th>
<th>r.h</th>
<th>r.a</th>
<th>r.width</th>
<th>r.get_width()</th>
<th>r.compute_area()</th>
<th>r.compute_height()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>p1.x</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>p1.y</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>p2.x</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>p2.y</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.pointA</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.pointB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.h</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.a</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.width</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.get_width()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.compute_area()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>r.compute_height()</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ Z = \{r.a\} \]
\[ D_F = \{r.a, r.breite, r.h, r.punktA, r.punktB, p1.x, p1.y, p2.x, p2.y\} \]
\[ D_A = \{r.berechne_flaeche(), r.breite(), r.berechne_hoehe()\} \]
\[ D = D_F \cup D_A \]
\[ = \{r.a, r.breite, r.h, r.berechne_flaeche(), r.breite(), r.berechne_hoehe(), r.punktA, r.punktB, p1.x, p1.y, p2.x, p2.y\} \]
\( \bar{B}_A = \Gamma^T_A \bar{B}^2_{v,m} \Gamma_A \) \hspace{1cm} (1)

Topological sorting: \( \bar{B}_A \rightarrow < r.\text{compute}_\text{height}(), r.\text{compute}_\text{area}() > \)
Metacompiler

ClassA.h
_tm_ClassSchema_ (
 )

ClassA.cpp

tmreflectc

ClassA_reflect.cpp
class Point : public tm_core::tmObject
{
public:

_tm_ClassSchema_
( <class name="Point">

  <superclass>tm_core::tmObject</superclass>

  <primitive name="X">
   <datatype>double</datatype>
   <read>get_x</read>
   <write>set_x</write>
   <activate>activate_x</activate>
   <info>"Coordinate x."</info>
  </primitive>

  <primitive name="Y">
   <datatype>double</datatype>
   <read>get_y</read>
   <write>set_y</write>
   <activate>activate_y</activate>
   <info>"Coordinate y."</info>
  </primitive>

   </class> );

Point(double x, double y, tm_core::tmRepository* repo = 0);
Point(tm_core::tmRepository* repo = 0);
double get_x() const { return _x; }
bool set_x(double value);
double get_y() const { return _y; }
bool set_y(double value);
void set_xy(double x, double y);

protected:

/* -- this is needed for persistence ------------------------------ */
TM_OBJECT
Point(const QString& id, tm_core::tmRepositoryP* repo = 0)
 : tm_core::tmObject(id,repo) {
   init();
 }
void activate_x(double x) { _x = x; }
void activate_y(double y) { _y = y; }
/* ------------------------------------------------------------------ */

private:

   void init();
   double _x, _y;

};
class Rectangle : public tm_core::tmObject
{
public:

_tm_ClassSchema_
( <class name="Rectangle">
  <superclass>tm_core::tmObject</superclass>

  <primitive name="Width" virtual="true">
    <datatype>double</datatype>
    <read>get_width</read>
    <info>"Width of rectangle."</info>
  </primitive>

  <primitive name="Height">
    <datatype>double</datatype>
    <read>get_height</read>
    <activate>activate_height</activate>
    <info>"Height of rectangle."</info>
  </primitive>

  <primitive name="Area">
    <datatype>double</datatype>
    <read>get_area</read>
    <activate>activate_area</activate>
    <info>"Area of rectangle."</info>
  </primitive>

}
<reference name="point_A">
   <datatype>Point</datatype>
   <read>get_point_A</read>
   <activate>activate_Point_A</activate>
</reference>

<reference name="point_B">
   <datatype>Point</datatype>
   <read>get_point_B</read>
   <activate>activate_Point_B</activate>
</reference>

<method name="compute_height">
   <returntype>void</returntype>
   <input>point_A.Y</input>
   <input>point_B.Y</input>
   <output>Height</output>
</method>

<method name="compute_area">
   <returntype>void</returntype>
   <input>Height</input>
   <input>Width</input>
   <output>Area</output>
</method>

<method name="get_width">
   <returntype>double</returntype>

South Africa, Johannesburg, 27 mar. 2003
<input>point_A.X</input>
<input>point_B.X</input>
<output>Width</output>
</method>

</class>);

Rectangle(Point* p1, Point* p2, tm_core::tmRepository* repo = 0);
double get_width() { return _point_B->get_x() - _point_A->get_x(); }
double get_height() { return _height; }
double get_area() { return _area; }
void compute_height() { _height = _point_B->get_y() - _point_A->get_y(); }
void compute_area() { _area = _height * get_width(); }
QString get_point_A() { return _point_A->get_ID(); }
QString get_point_B() { return _point_B->get_ID(); }

private:

void init();

double _height, _area;

Point* _point_A;
Point* _point_B;

};
• Besides direct dependencies of attributes, indirect dependencies are of special interest.

• The sequence in which the attributes are updated is vital.

• Due to the continuous change of source and target data by the users of the related applications, the links between the attributes are not static.

• In a distributed environment, it is unlikely that all objects are known at a central location.

• The number of attributes is high.
Conclusion

- Standards for exchanging data and remote method invocation are available

- On their own, these standards do not solve the problems of engineering in distributed environments

- The development of methods for distributed engineering in computer networks require mathematical rigour, i.e. application of graph theory