University of Leicester

Model-based Development of Web Services

Reiko Heckel

Workshop on Specification and Design Methodology for Adaptive and Embedded Systems

Bangalore, January 2005

Application Scenario: Online Shopping with Max

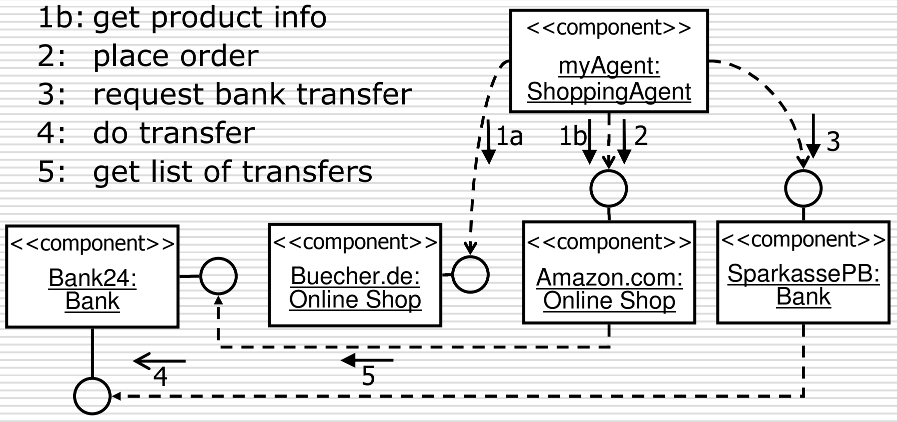
Max, seeking
"Harry Potter, The Order of the Phoenix", employs *Shopping Agent* to

- look for the book
- obtain further product info (prize, availability, ...)
- choose the best offer
- order and pay (via bank transfer)

University of Leicester

Scenario as UML Diagram

1a: get product info
1b: get product info
2: place order
3: request bank transfer
4: do transfer
5: get list of transfers



University of Leicester

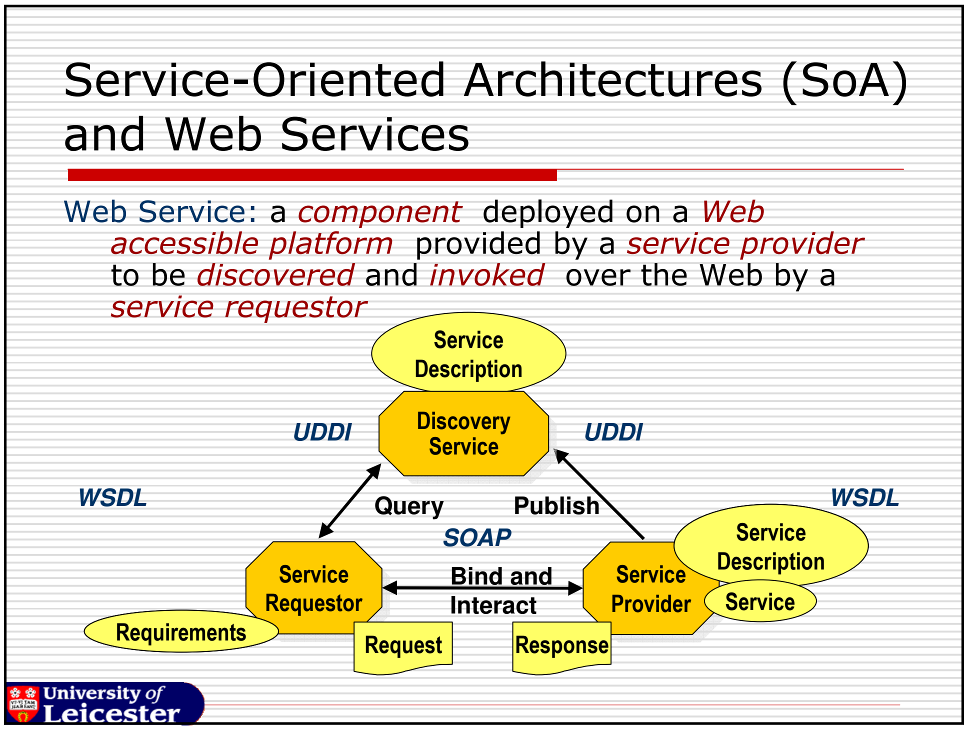
But, …

- How do SparkassePB and Amazon.com know Bank24?
- How does myAgent know Buecher.de and Amazon.com?

static	compiled into component
	loaded from configuration files
	specified by the user
dynamic	retrieved from a public registry

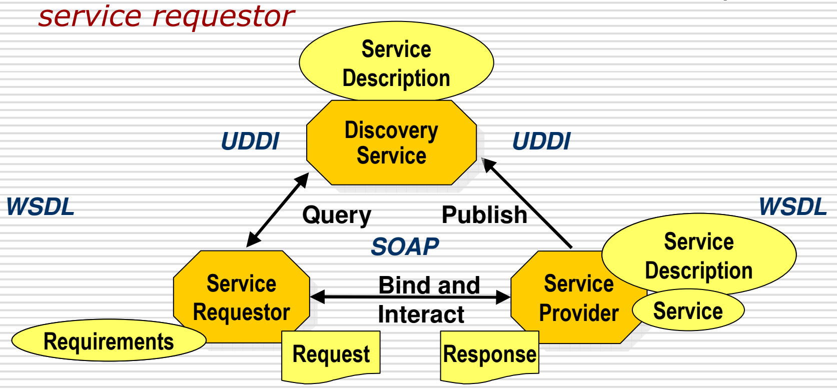
→ *service-oriented architecture*

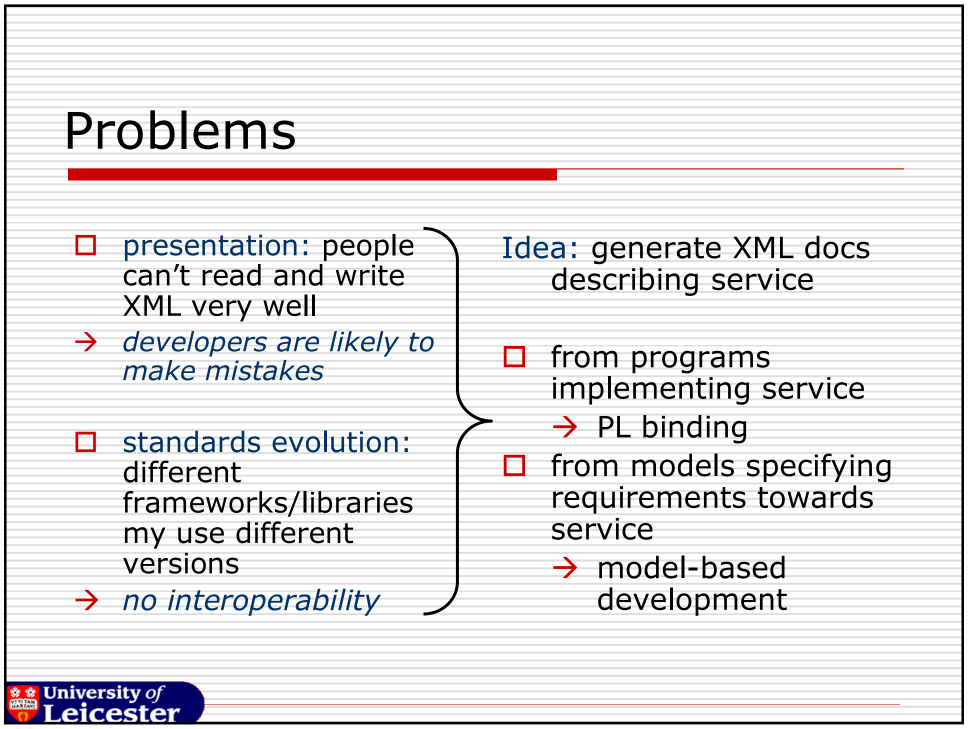
University of Leicester



Service-Oriented Architectures (SoA) and Web Services

Web Service: a *component* deployed on a *Web accessible platform* provided by a *service provider* to be *discovered* and *invoked* over the Web by a *service requestor*





Problems

- presentation: people can't read and write XML very well

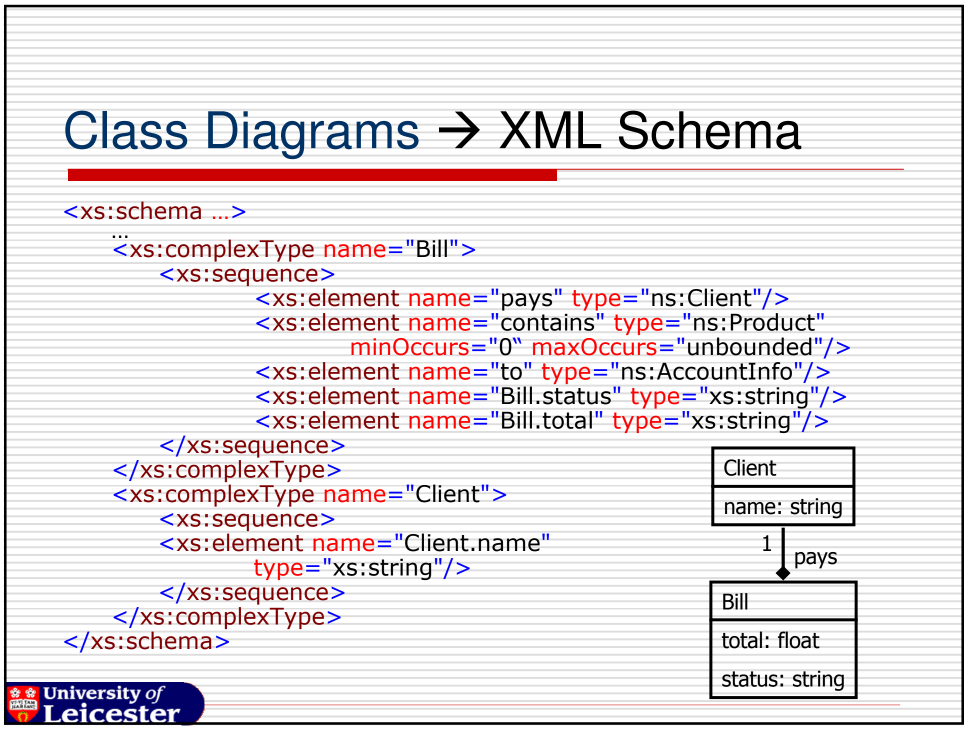
→ *developers are likely to make mistakes*

- standards evolution: different frameworks/libraries my use different versions

→ *no interoperability*

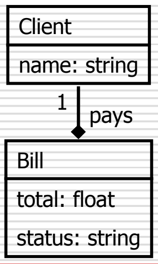
Idea: generate XML docs describing service

- from programs implementing service
  → PL binding
- from models specifying requirements towards service
  → model-based development

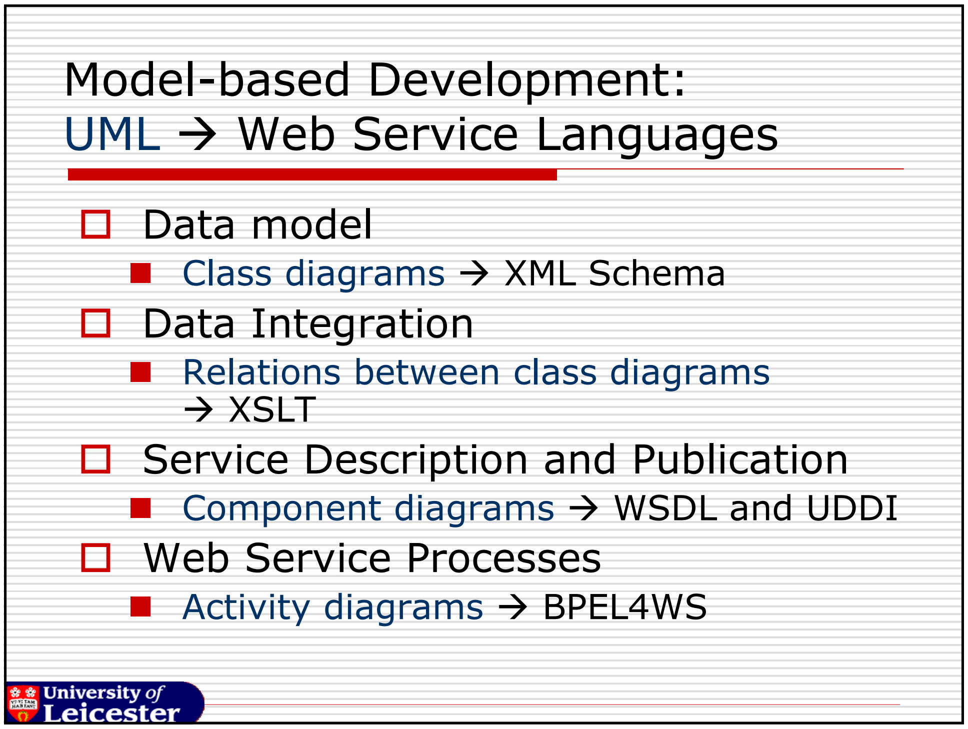


Class Diagrams → XML Schema

```
<xs:schema …>
     …
     <xs:complexType name="Bill">
          <xs:sequence>
                    <xs:element name="pays" type="ns:Client"/>
                    <xs:element name="contains" type="ns:Product"
                              minOccurs="0" maxOccurs="unbounded"/>
                    <xs:element name="to" type="ns:AccountInfo"/>
                    <xs:element name="Bill.status" type="xs:string"/>
                    <xs:element name="Bill.total" type="xs:string"/>
          </xs:sequence>
     </xs:complexType>
     <xs:complexType name="Client">
          <xs:sequence>
          <xs:element name="Client.name"
                    type="xs:string"/>
          </xs:sequence>
     </xs:complexType>
</xs:schema>
```



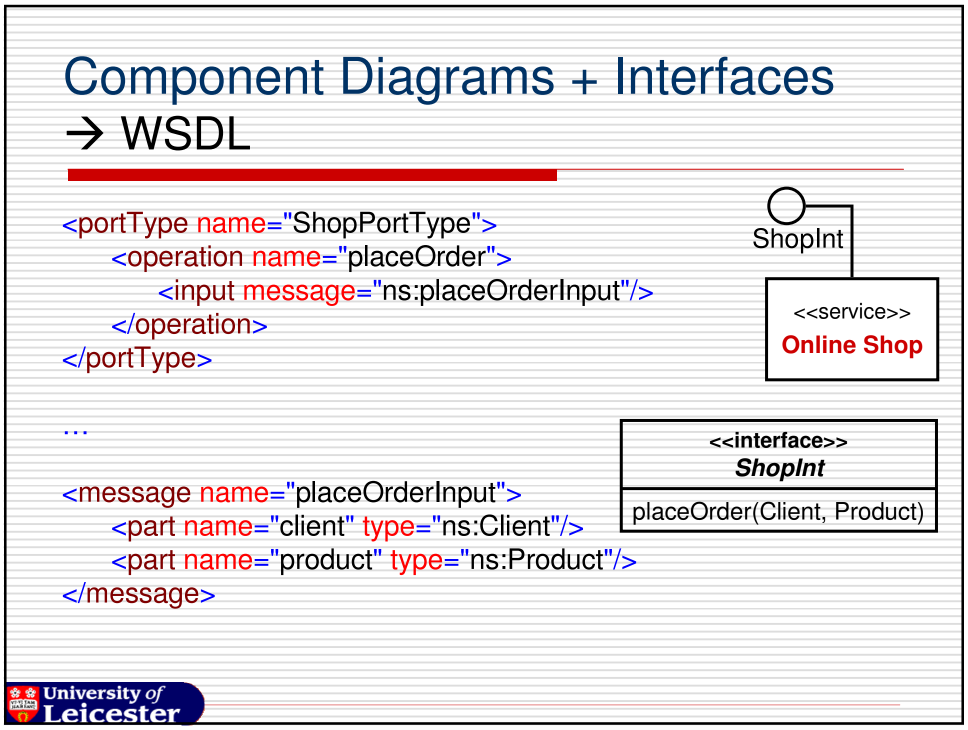
University of Leicester



Model-based Development: UML → Web Service Languages

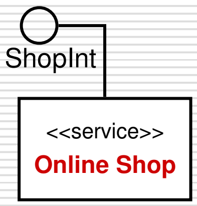
- Data model
  - Class diagrams → XML Schema
- Data Integration
  - Relations between class diagrams → XSLT
- Service Description and Publication
  - Component diagrams → WSDL and UDDI
- Web Service Processes
  - Activity diagrams → BPEL4WS

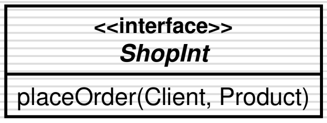
University of Leicester



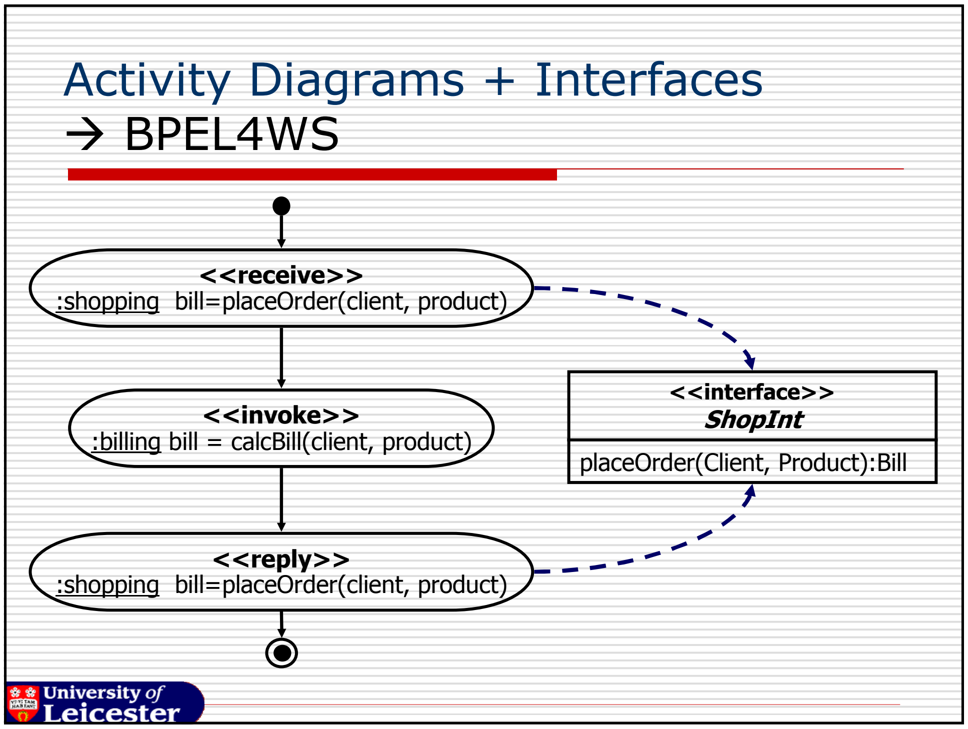
Component Diagrams + Interfaces → WSDL

```
<portType name="ShopPortType">
    <operation name="placeOrder">
        <input message="ns:placeOrderInput"/>
    </operation>
</portType>

…

<message name="placeOrderInput">
    <part name="client" type="ns:Client"/>
    <part name="product" type="ns:Product"/>
</message>
```

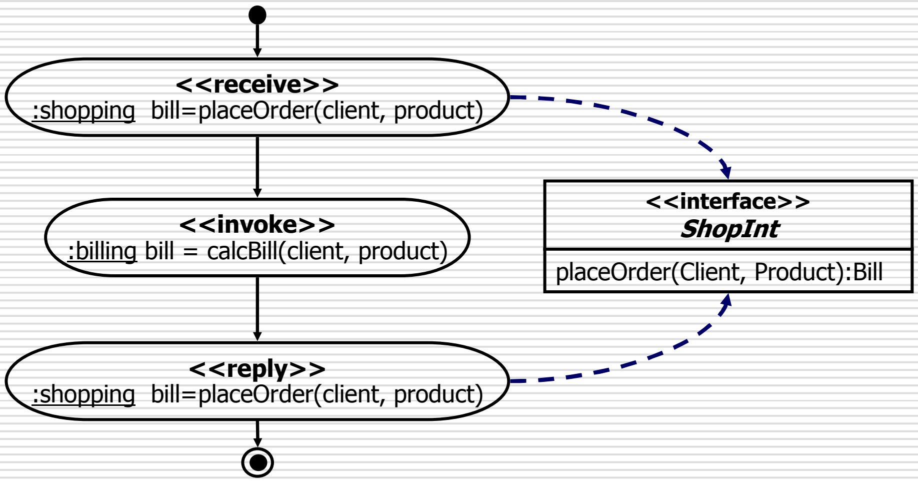




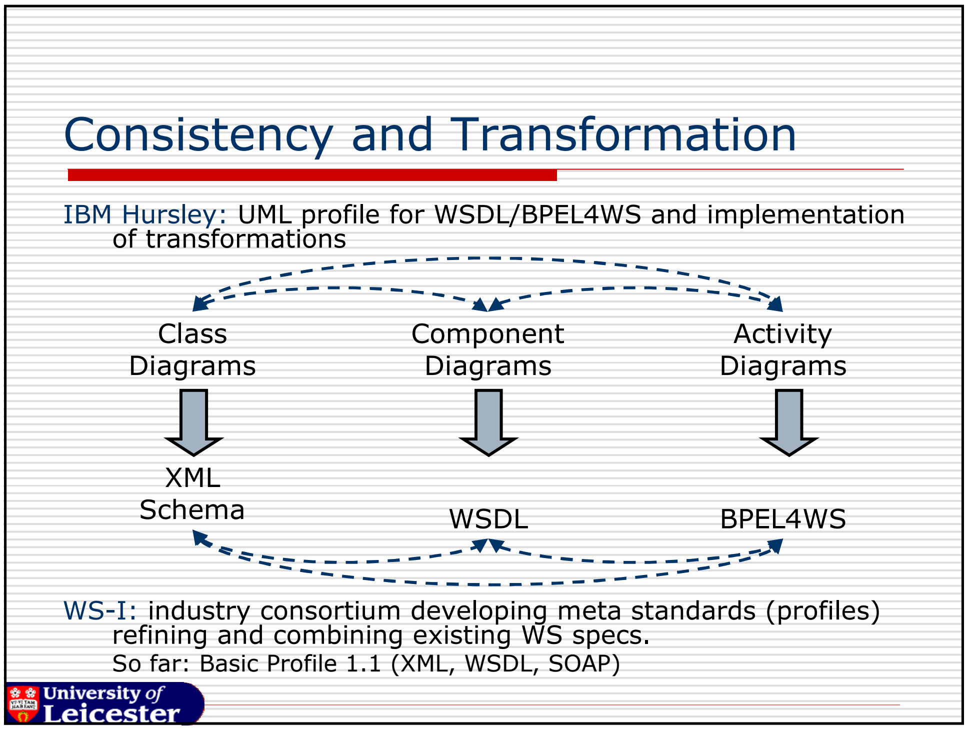
University of Leicester



Activity Diagrams + Interfaces → BPEL4WS


<<receive>>
:shopping bill=placeOrder(client, product)
<<invoke>>
:billing bill = calcBill(client, product)
<<reply>>
:shopping bill=placeOrder(client, product)
<<interface>>
ShopInt
placeOrder(Client, Product):Bill

University of Leicester



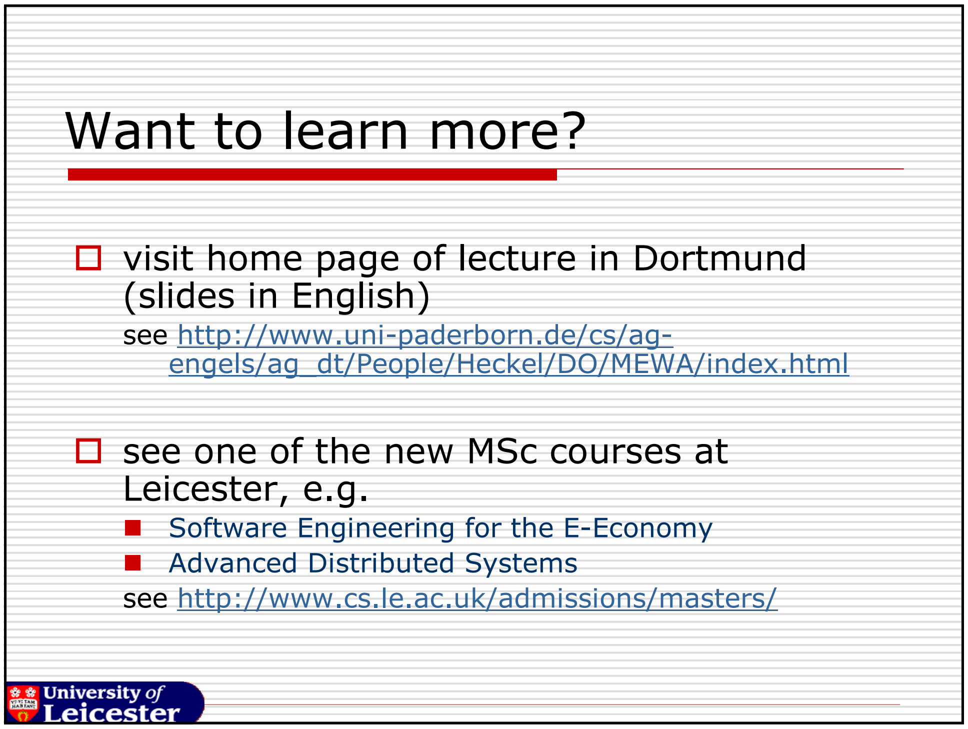
Consistency and Transformation

IBM Hursley: UML profile for WSDL/BPEL4WS and implementation of transformations

Class Diagrams → XML Schema
Component Diagrams → WSDL
Activity Diagrams → BPEL4WS

WS-I: industry consortium developing meta standards (profiles) refining and combining existing WS specs.
So far: Basic Profile 1.1 (XML, WSDL, SOAP)

University of Leicester

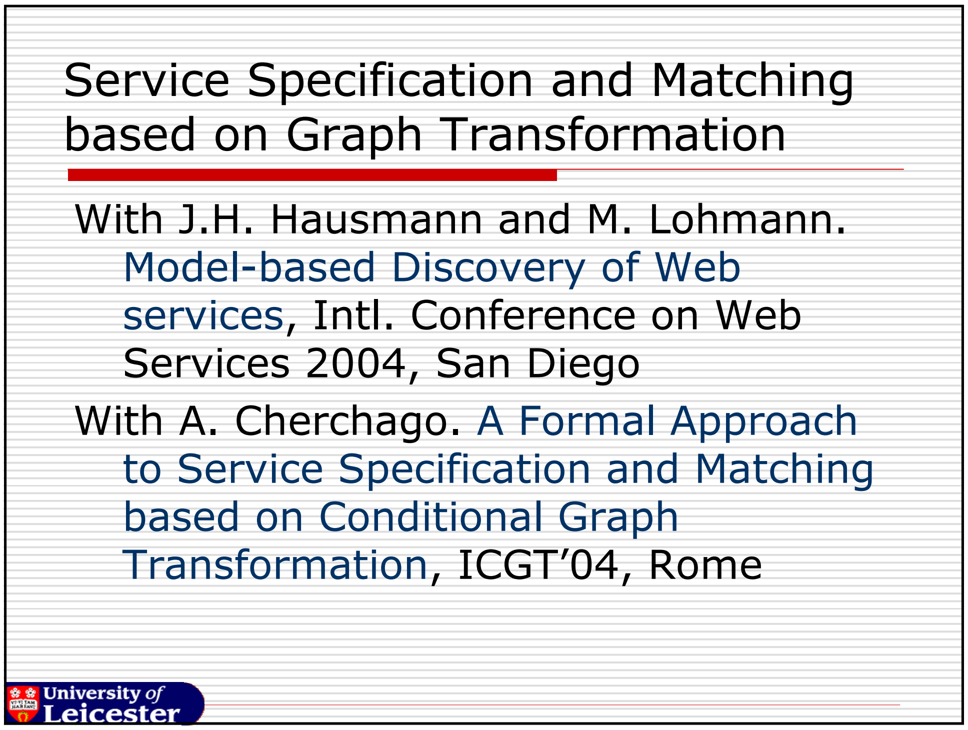


Want to learn more?

- visit home page of lecture in Dortmund (slides in English)
  see http://www.uni-paderborn.de/cs/ag-engels/ag_dt/People/Heckel/DO/MEWA/index.html
- see one of the new MSc courses at Leicester, e.g.
  - Software Engineering for the E-Economy
  - Advanced Distributed Systems

  see http://www.cs.le.ac.uk/admissions/masters/

University of Leicester

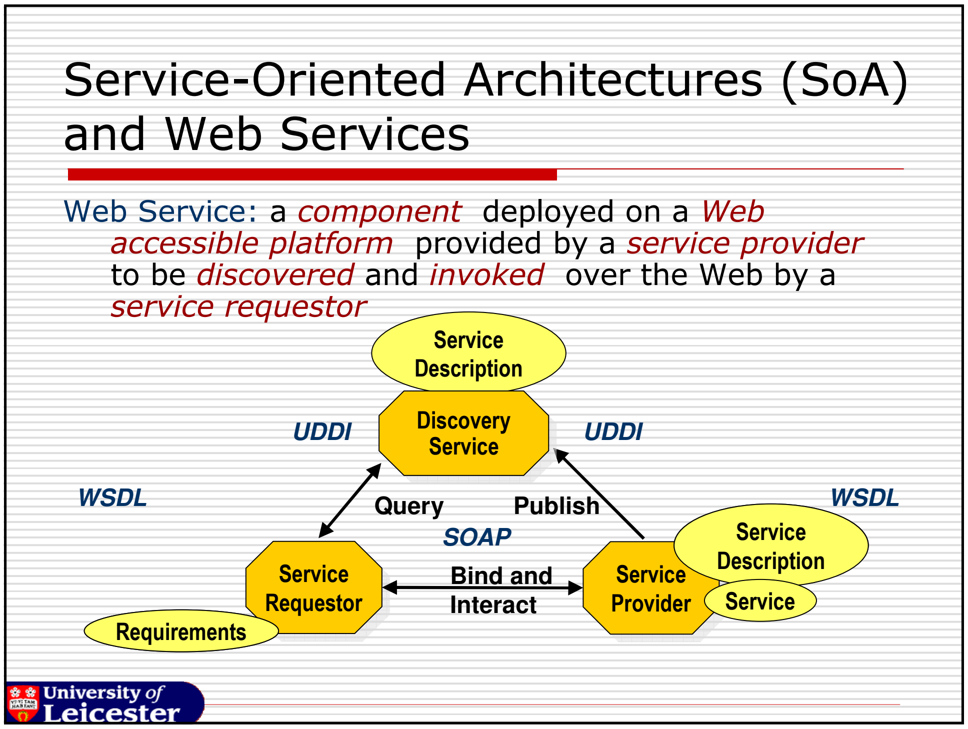


Service Specification and Matching based on Graph Transformation

With J.H. Hausmann and M. Lohmann. Model-based Discovery of Web services, Intl. Conference on Web Services 2004, San Diego

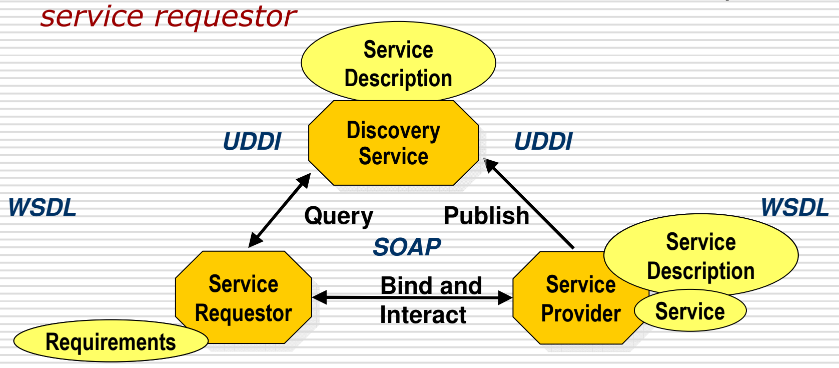
With A. Cherchago. A Formal Approach to Service Specification and Matching based on Conditional Graph Transformation, ICGT'04, Rome

University of Leicester

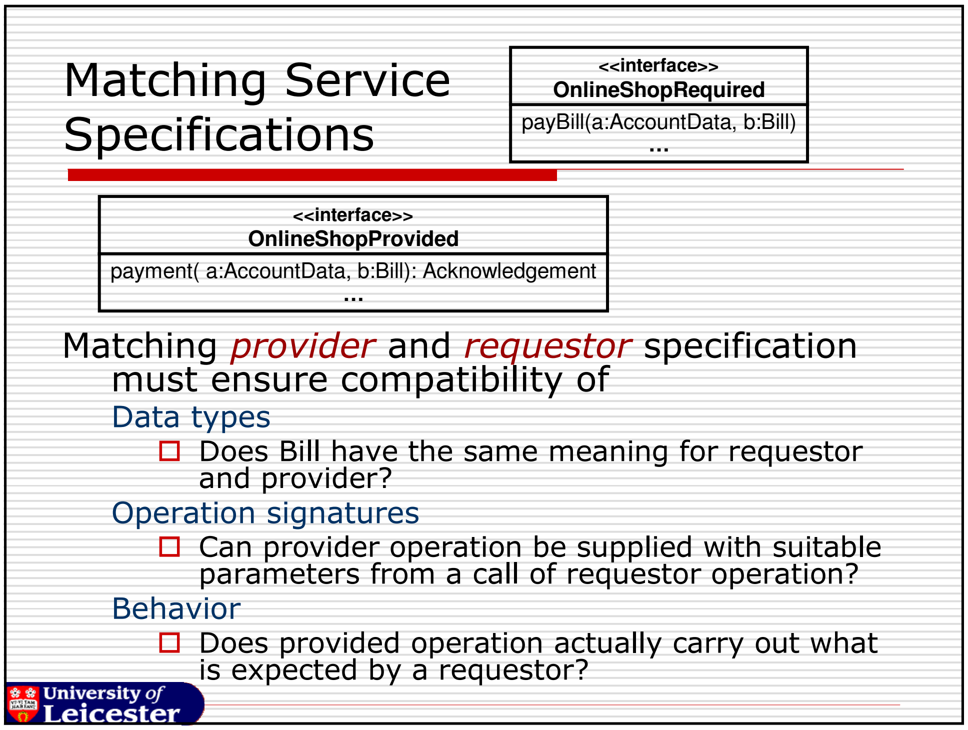


Service-Oriented Architectures (SoA) and Web Services

Web Service: a *component* deployed on a *Web accessible platform* provided by a *service provider* to be *discovered* and *invoked* over the Web by a *service requestor*



University of Leicester



Matching Service Specifications

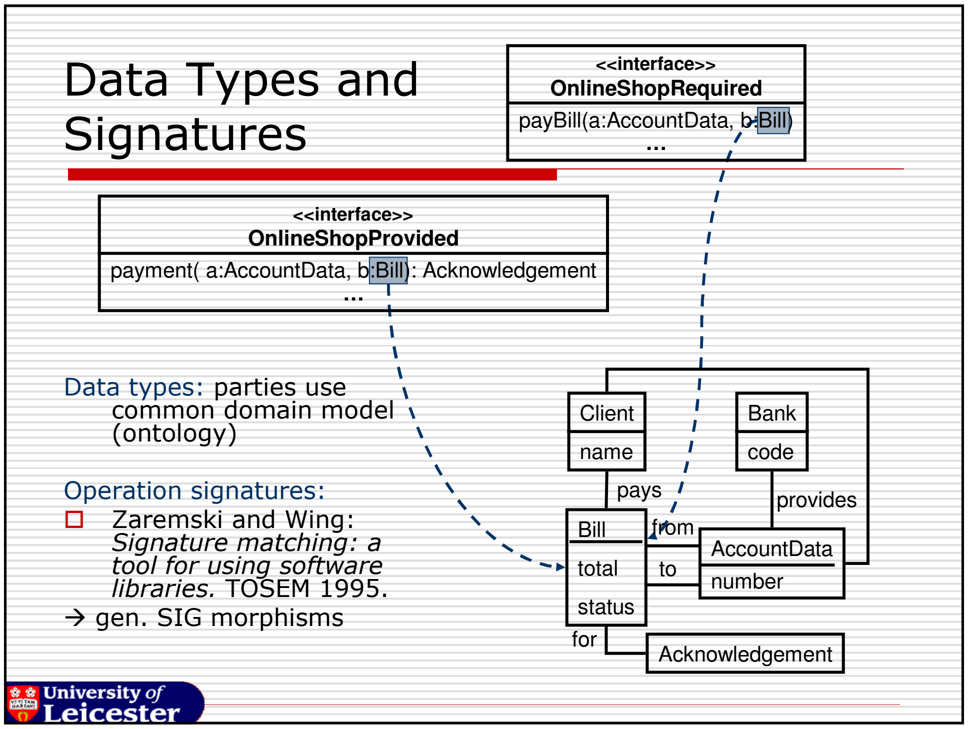
<<interface>>
OnlineShopRequired
payBill(a:AccountData, b:Bill)
...

<<interface>>
OnlineShopProvided
payment(a:AccountData, b:Bill): Acknowledgement
...

Matching *provider* and *requestor* specification must ensure compatibility of

- Data types
  - Does Bill have the same meaning for requestor and provider?
- Operation signatures
  - Can provider operation be supplied with suitable parameters from a call of requestor operation?
- Behavior
  - Does provided operation actually carry out what is expected by a requestor?

University *of* Leicester



Data Types and Signatures

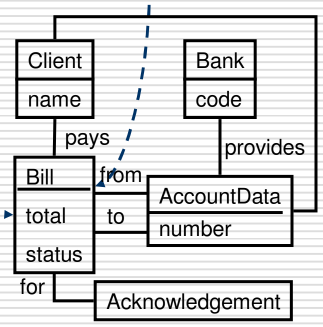
<<interface>>
OnlineShopRequired
payBill(a:AccountData, b:Bill)
...

<<interface>>
OnlineShopProvided
payment(a:AccountData, b:Bill): Acknowledgement
...

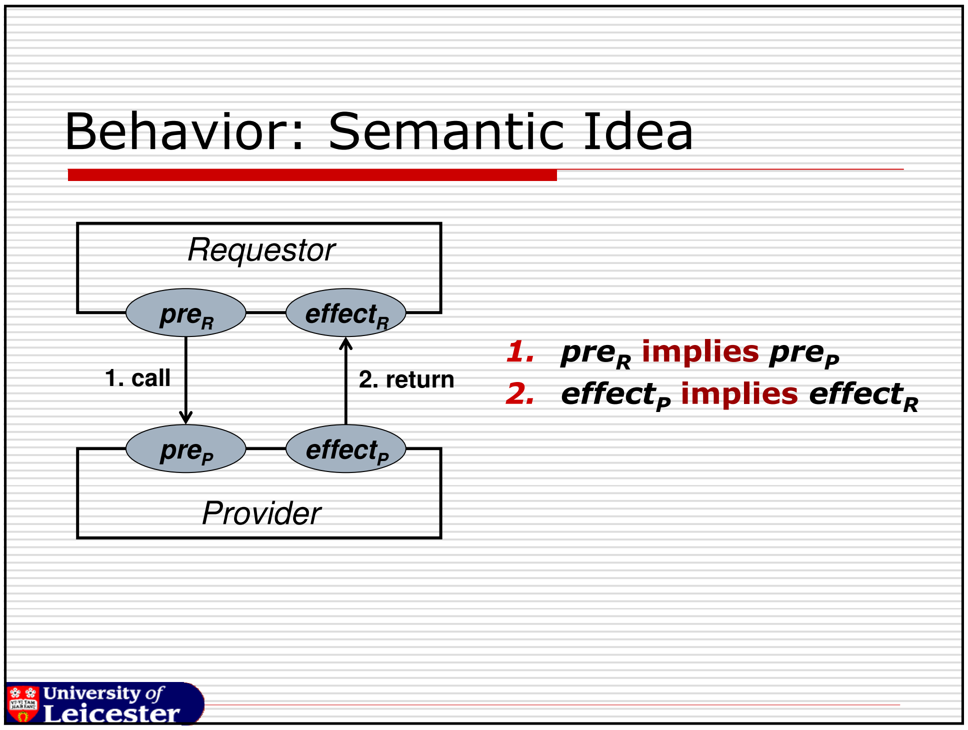
Data types: parties use common domain model (ontology)

Operation signatures:
- Zaremski and Wing: *Signature matching: a tool for using software libraries.* TOSEM 1995.

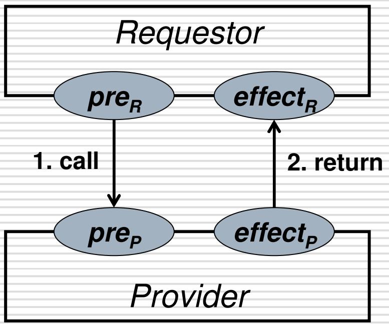
→ gen. SIG morphisms



University *of* Leicester

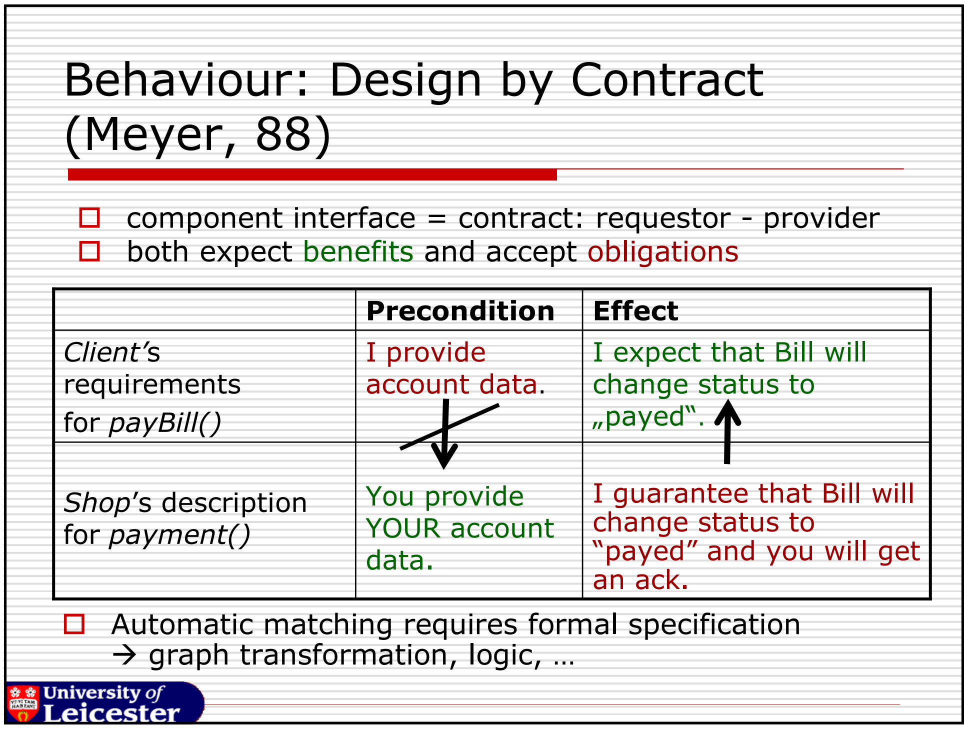


Behavior: Semantic Idea



1. ***pre_R* implies *pre_P***
2. ***$effect_P$* implies *$effect_R$***

University *of* Leicester



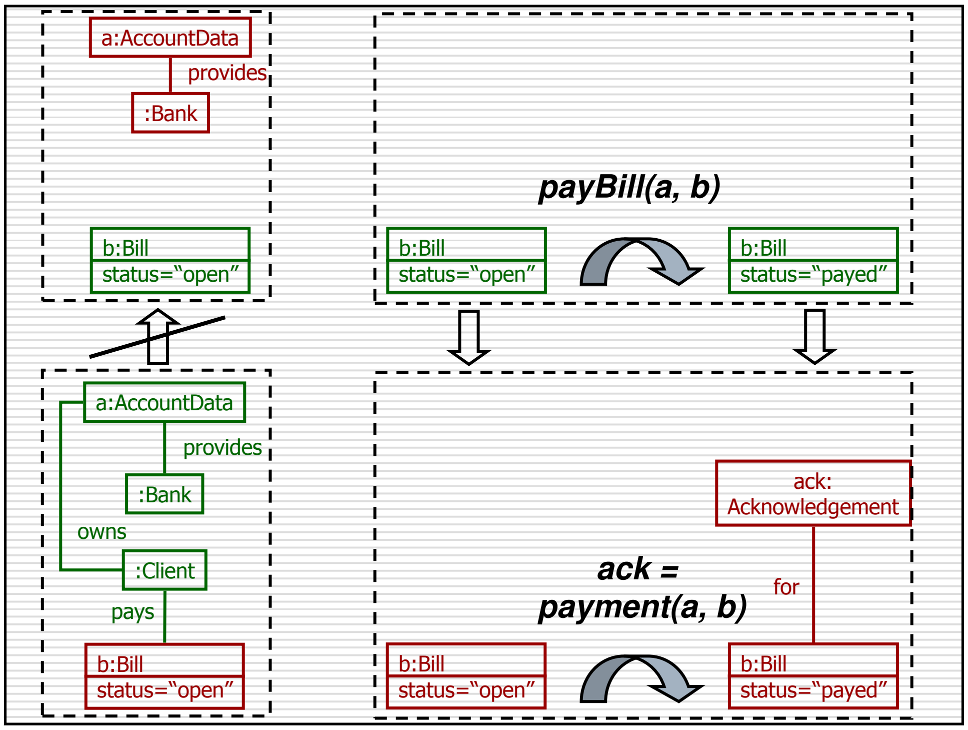
Behaviour: Design by Contract (Meyer, 88)

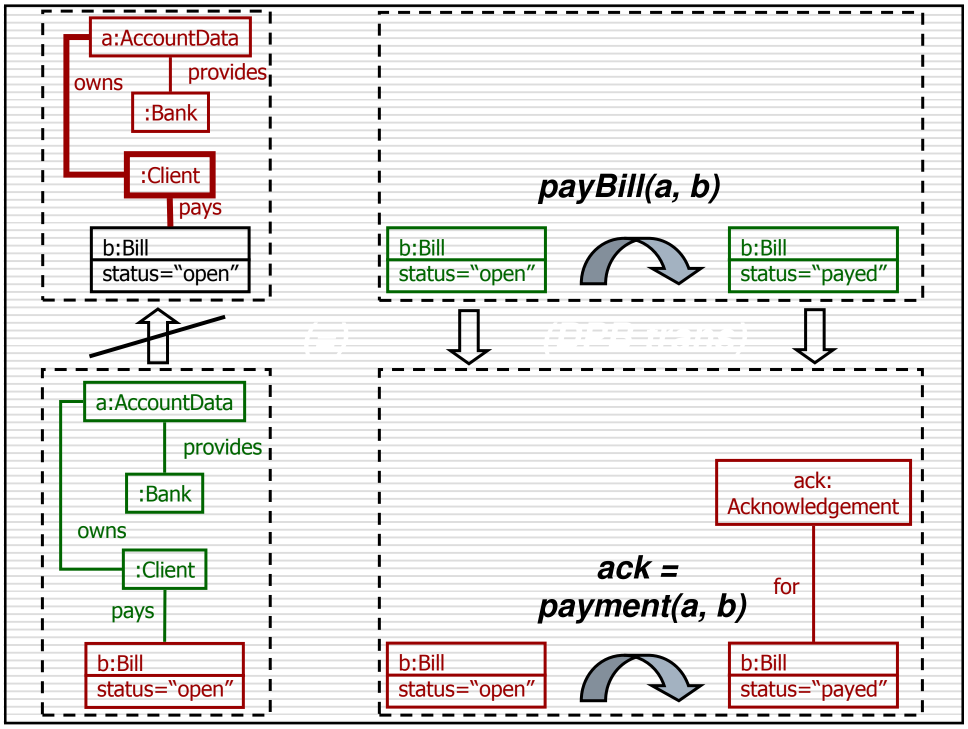
- component interface = contract: requestor - provider
- both expect benefits and accept obligations

	Precondition	**Effect**
Client's requirements for *payBill()*	I provide account data.	I expect that Bill will change status to „payed".
Shop's description for *payment()*	You provide YOUR account data.	I guarantee that Bill will change status to "payed" and you will get an ack.

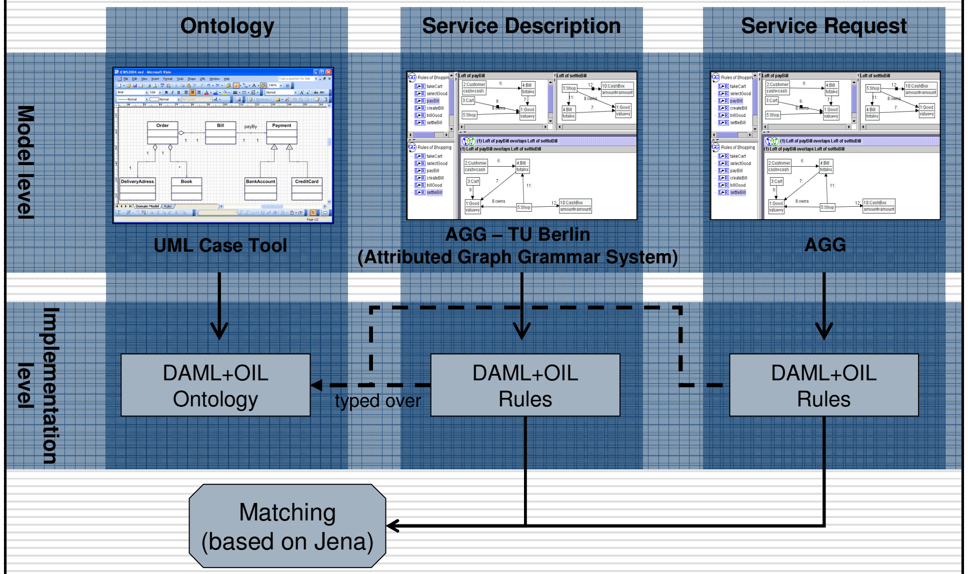
- Automatic matching requires formal specification
  → graph transformation, logic, …

University *of* Leicester

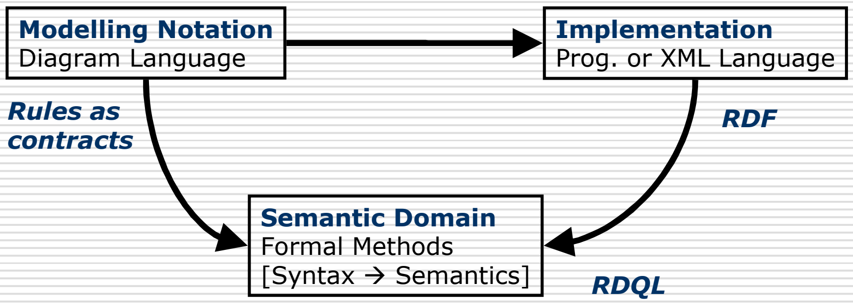

a:AccountData
provides
:Bank
b:Bill
status="open"
payBill(a, b)
b:Bill
status="open"
b:Bill
status="payed"
a:AccountData
provides
:Bank
owns
:Client
pays
b:Bill
status="open"
ack:
Acknowledgement
ack =
payment(a, b)
for
b:Bill
status="open"
b:Bill
status="payed"


a:AccountData
owns
provides
:Bank
:Client
pays
b:Bill
status="open"
payBill(a, b)
b:Bill
status="open"
b:Bill
status="payed"
a:AccountData
provides
:Bank
owns
:Client
pays
b:Bill
status="open"
ack:
Acknowledgement
ack =
payment(a, b)
for
b:Bill
status="open"
b:Bill
status="payed"

Implementation (Prototype)



Generally: Formal Methods for Semantic Consistency



Like in compilers, data base systems, … formal methods are there, but should be hidden.

University of Leicester