Mag. iur. Dr. techn. Michael Sonntag

XML in E-Business

ebXML, SOAP&WSDL&UDDI, XML for websites

Institute for Information Processing and
Microprocessor Technology (FIM)
Johannes Kepler University Linz, Austria

E-Mail: sonntag@fim.uni-linz.ac.at
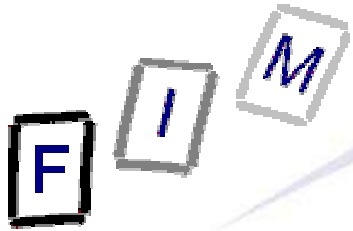http://www.fim.uni-linz.ac.at/staff/sonntag.htm
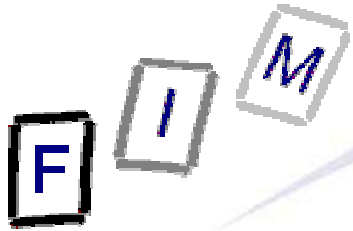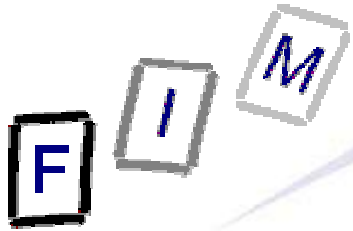
Questions?

Please ask immediately!

- Why XML?
  - → The need for automatic conversions
- We will cover four different groups of standards briefly:
  - → Business data interchange: ebXML
    - » Defining the data to be exchanged between companies
  - → Web services: SOAP, WSDL, UDDI
    - » How to transfer data, provide electronic services, integrate software
- Webservice example: Amazon ECS

- HTML, EDI, ... have been sufficient in the past!
  - → HTML is a presentation language and very hard to parse
    - » Syntax is quite open, different "versions", ...
  - → EDI is extremely complicated and difficult
- XML is like relational databases
  - » RDB = Storage, XML = Transmission
  - → A framework for structured content
  - → But the actual rows/elements must be defined by the user!
- Difficulty: Mapping from one XML schema to another
  - → Syntax is rather easy: XSL for "rewriting"
  - → Semantics is much more difficult!
    - » Often not available in "usable" form; e.g. in a text document
    - » Semantic description possible today; semantic "rewriting" still far away
      - – Some early and easy examples possible through automatic deduction
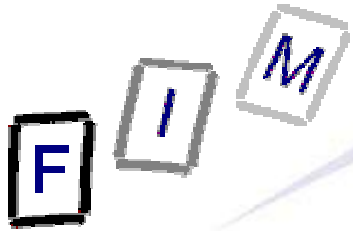
- Each company has its own way of doing business and therefore also its own data schema
  - → Mapping data schemas to each other has a complexity of O(N^2)
  - → This is not feasible for more than very few partners!
    - » But how often must company A map data from B directly to C?
- Solution: Define your own "canonical" schema and convert all data to it (and then back to a different schema if required)
  - → Complexity: O(N), or more exactly 2N (from and to each partner)
- Still a difficulty: Exchanging the internal software, processes, etc.
  - → All transformations would have to be modified
  - → Solution: A single transformation from the canonical data schema to your own internal representation
    - » Practice: Database and schema identical at start; drift apart later on
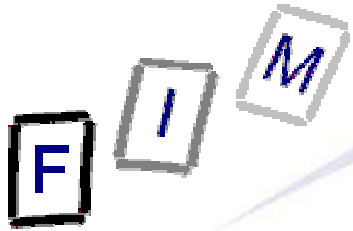   - » Important: Allow separation right from the beginning!

Complete meshing

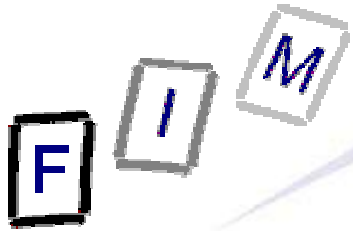Direct mappings

Intermediate data schema
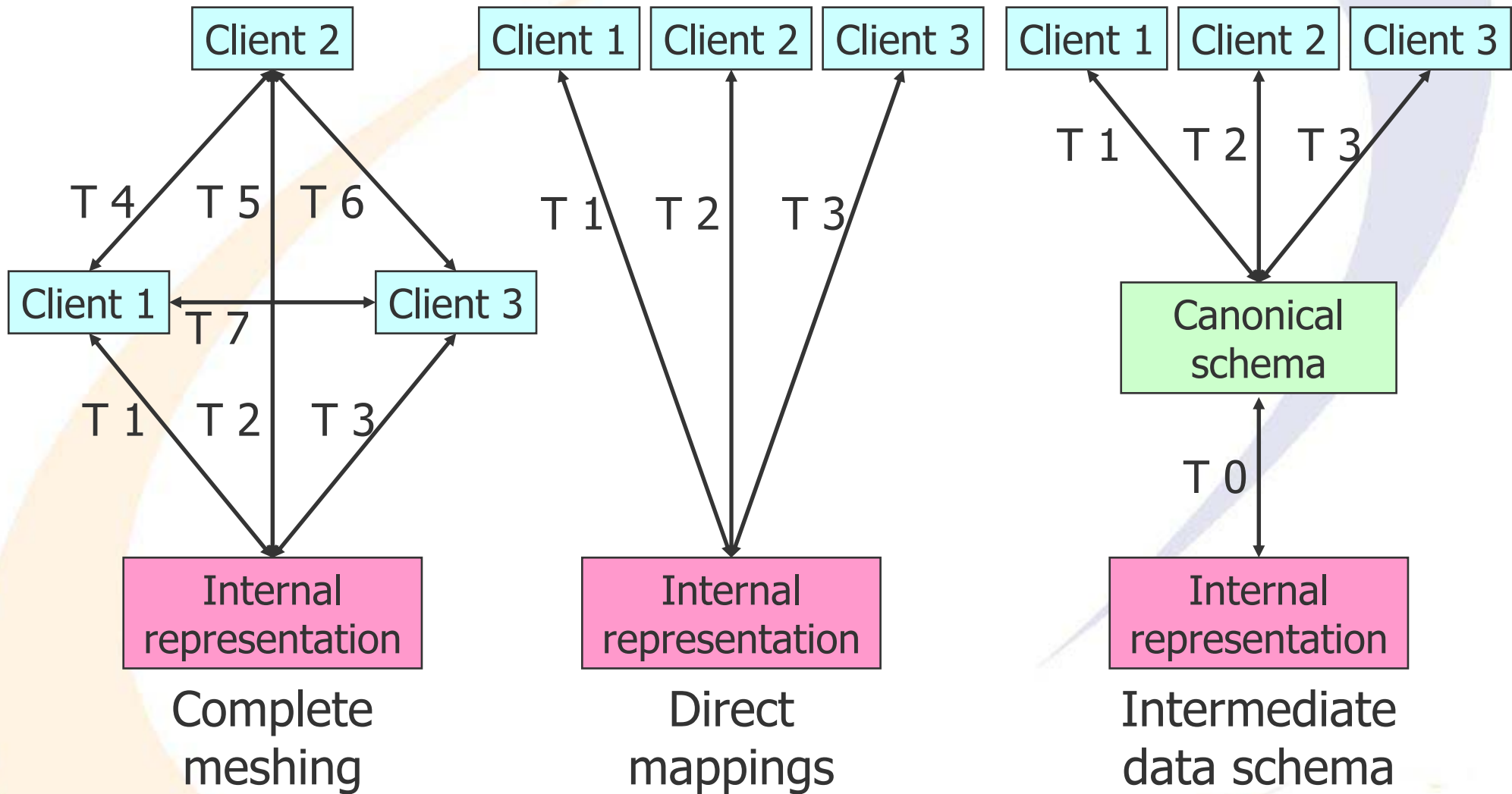
Automatic conversions

- The conversions from before should be made automatically
- This requires:
  - → Exact syntax definition (incl. charset, size of data elements, ...)
  - → Exact semantics definition
    - » Is "delivery date" the date of sending or the expected date of receipt?
  - → All definitions must be explicitly clear and unambiguous
  - → Platform independence: All software/hardware/platforms supported
- What would be nice in addition:
  - → Automatically verifying input and output whether it is syntactically correct and "makes sense"; as extensive as possible
  - → Easily extensible: Adding new data, messages, protocols
  - → Strict definition: If there exists a way, there should be only one way
  - → Simplicity: Learning curve should be low and flat
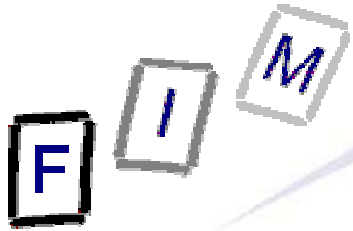 - → Possibility for industry standards

- XML provides all of the issues above except the semantics
  - » There exist semantic standards based on XML; but they are not (yet?) in widespread use!
  - → XML Schema (or others, e.g. Relax NG) provide syntax definition
    - » This is itself in XML and therefore fully automatic and unambiguous
    - » Validation completely automatic; depends only on degree of completeness/exactness of the definition
  - → Complete platform independence: Unicode, available for all programming languages and systems
  - → Extensibility: XML Schema supports a kind of "object orientation" with subclassing; "procedural" extensions also possible
  - → XML is strictly defined and simple
    - » 6 years without changes except error corrections and clarifications
  - → Industry standards available, e.g. ebXML or BizTalk

XML Basics

- XML is similar to HTML: Elements and attributes
  - → Difference: Always end tag; empty tags; attributes always "quoted" and always have a value; case sensitive
  - → No "interweaving": XML is always a tree (not a wood!)
- Namespaces are an important part to distinguish elements from different areas (here: different companies)
- Special handling of whitespaces and linebreaks
  - → Whitespace handling can be customized (e.g. if important)

```
<Order>
   <OrderNr>139</OrderNr><OrderingDate>11.11.2003</OrderingDate>
   <CustomerName>Michael Sonntag</CustomerName>
   <Items>
      <Item count="1" Nr="4711">Firewire harddisk (External)</Item>
   </Items>
   <DeliveryDate>2.1.2004, 10:30</DeliveryDate>
</Order>
```
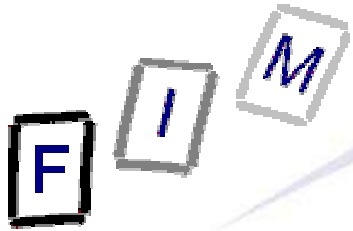
Schema Basics

- XML Schema describes both the structure and the content
  → Structure: Which elements are available and their nesting
  → Content: Datatypes (e.g. string patterns, date, numerical types)
- Drawback: Quite complicated
  → Even a relatively simple XML file has a long schema
    » One reason is, that it is explicitly specified to the last possible item!
  → Schemas are not self-explanatory: Understanding a schema is difficult (but tool support is growing)

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema ... >
   <xs:element name="message"><xs:complexType>
    <xs:sequence>
     <xs:element name="from" type="xs:string"/>
     <xs:element name="to" type="xs:string"/>
    </xs:sequence>
   </xs:complexType></xs:element>
</xs:schema>
```

```
<message ...>
   <from>Michael Sonntag</from>
   <to>Rudolf Hoermanseder</to>
</message>
```
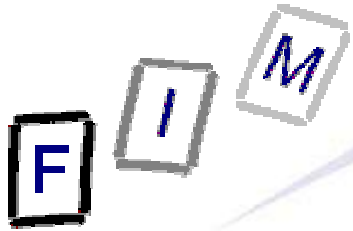
- XML manipulation is possible in many ways:
  - → Manually (i.e. string manipulation): Please avoid it!
  - → DOM/SAX: Programmatically (large changes or inserting new data)
  - → Stylesheets: These are itself XML and describe how to transform XML input into some (XML, HTML, text, ...) output
- Stylesheets = XSL (eXtensible Stylesheet Language)
  - = Transformation rules (XSLT)
  - + expressions language for targeting document parts (XPath)
- This is especially useful for mapping one document to another
  - → Order from company A is transformed to the canoncial representation of an order for company B
  - → Extensible through custom functions (programs)
    - » This allows arbirary content modifications
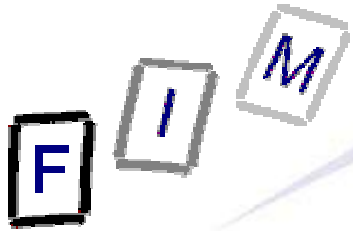- Limitations: Input & output must be known exactly (→XML Schema!)
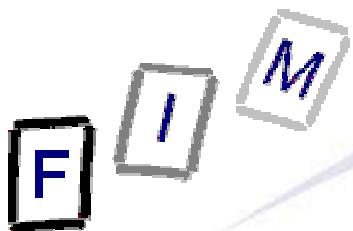
- Transformations possible in two ways
  - → Template rules (functional style): "Everything matching is handled exactly in this way"
  - → Programmatic rules (procedural style): "Now insert this part, then copy this if it matches, and finally loop over all such elements"
- Quite easy to learn, but difficult to master
- Also suitable for presentation
  - → Conversion to HTML (direct presentation in the WWW)
  - → Conversion to PDF (additional software req.) (XSL-FO)
- Integration with software also possible (variables, functions)

```
<xsl:template match="product">
<table><xsl:apply-templates select="sales/domestic"/></table>
<xsl:value-of select="@given-name"/><xsl:text> </xsl:text><xsl:value-of select="@family-name"/>
</xsl:template>
```
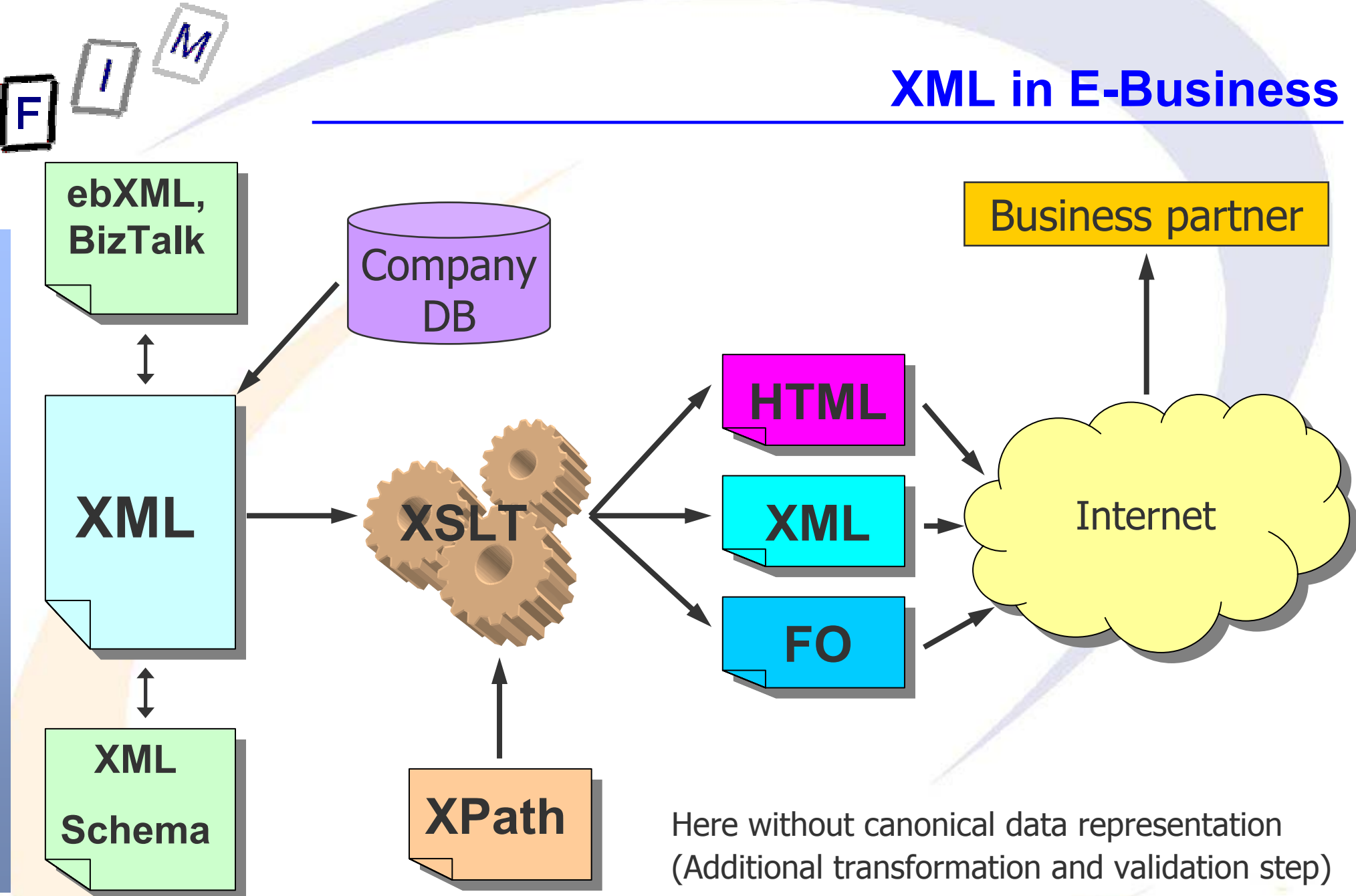
- ebXML = Electronic Business XML
  - → Should be a successor for EDI
    - » EDI problems: ex(t/p)ensive software and hardware, difficult to implement, integration into existing software
  - → Is a comprehensive global framework
  - → For the exchange of business data
- Components:
  - → Business process models
    - » Includes e.g. parties and message sequences
  - → Message formats
  - → Repositories/Directories
- Main advantage: Simpler and cheaper
  - → More suitable for SMEs
   How to exchange a sequence of messages

- Each company describes its business process
  - → Manually or from a standard definition
  - → E.g. "Buying something" = Send request, wait for ack., wait for acceptance, send invoice, send goods, ...
- Process and company data is stored in a registry
  - → Allows finding out what they do and how they do it
  - → Practice: Probably not public!
- Describing a business arrangement
  - → What processes were actually agreed upon to be performed
- Defined service for message exchange
  - → How to deliver/receive messages with non-repudiation, encryption, signatures, ...
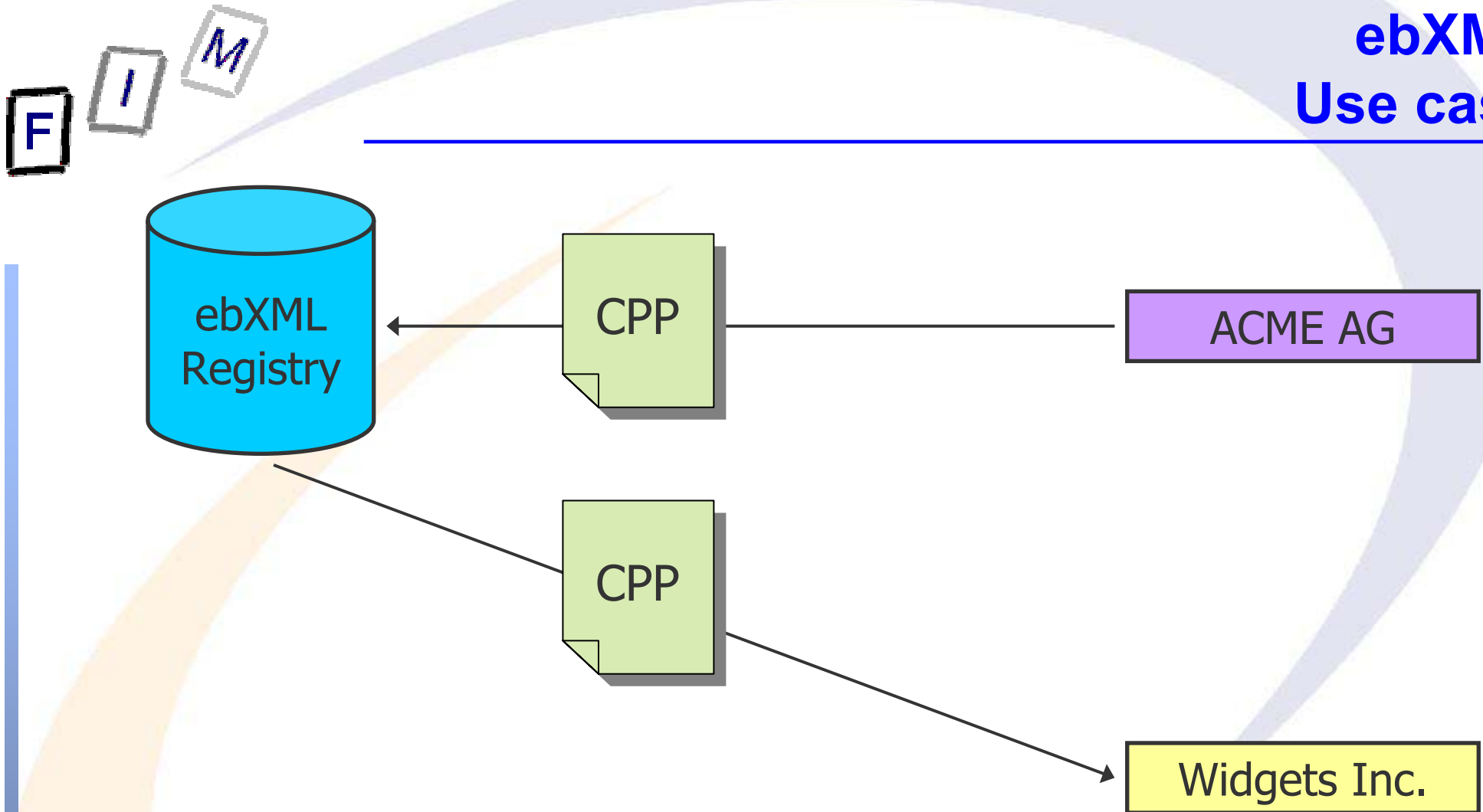
ebXML, BizTalk

Company DB

Business partner

XML

XSLT

HTML

XML

FO

Internet

XML Schema

XPath

Here without canonical data representation (Additional transformation and validation step)
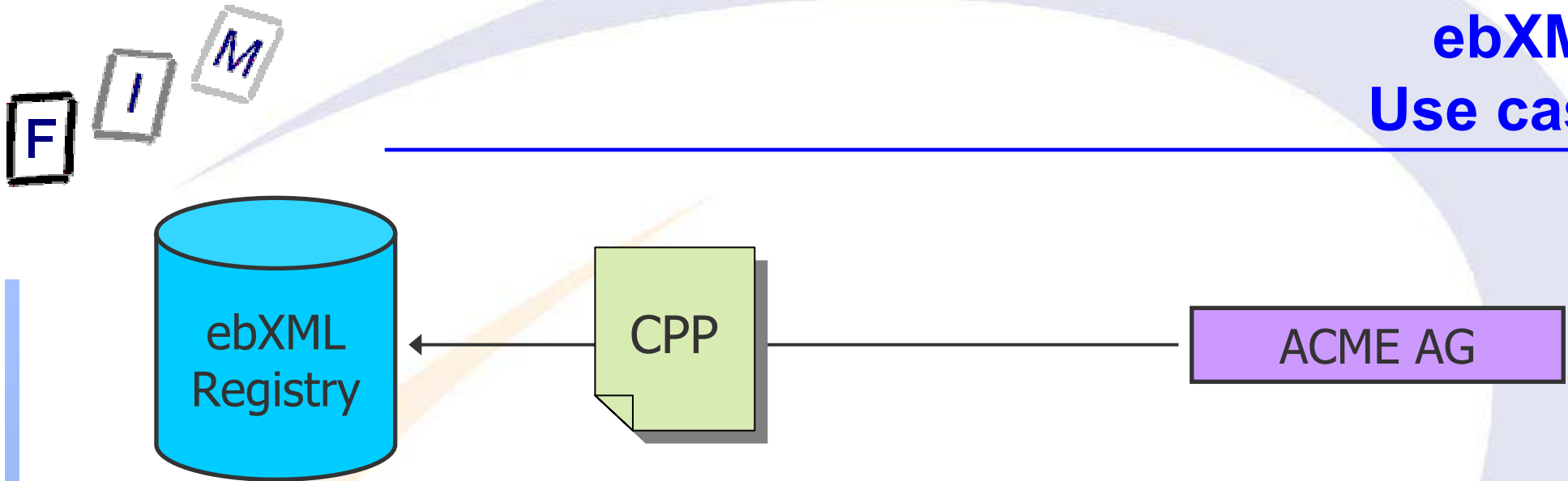
- The first company (ACME AG) provides information about the processes it can perform in a registry
 → This is a "CPP" (Collaboration Party Profile)

- The second company (Widgets Inc.) queries the registry and finds a suitable trading partner (=ACME)
  - → It retrieved the CPP and compares it with its desires

F I M

ebXML
Registry

CPP

ACME AG
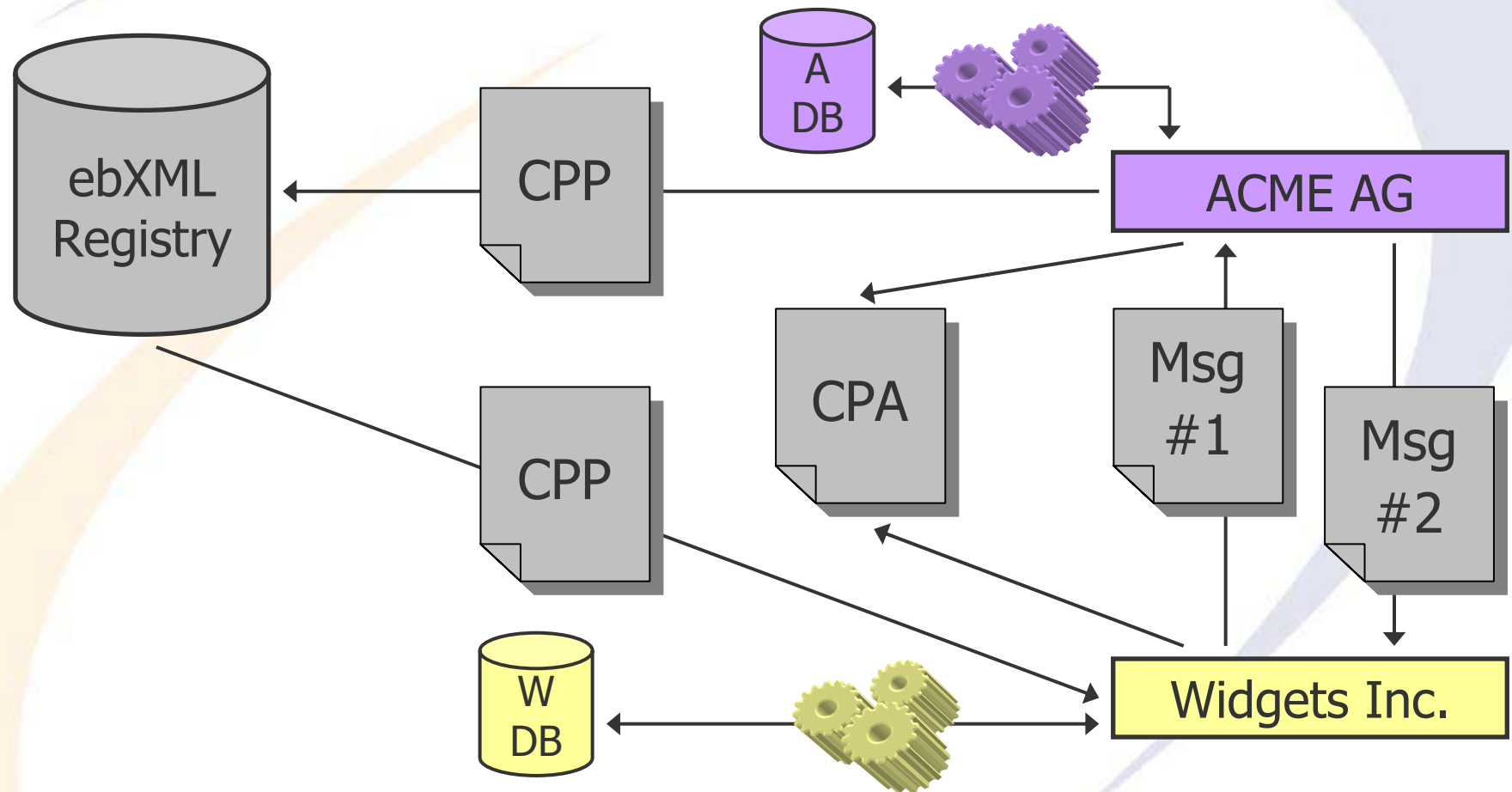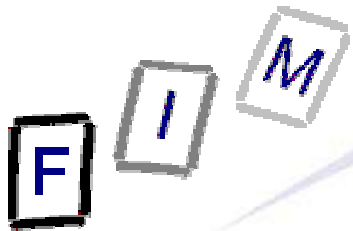
CPP

CPA

Widgets Inc.
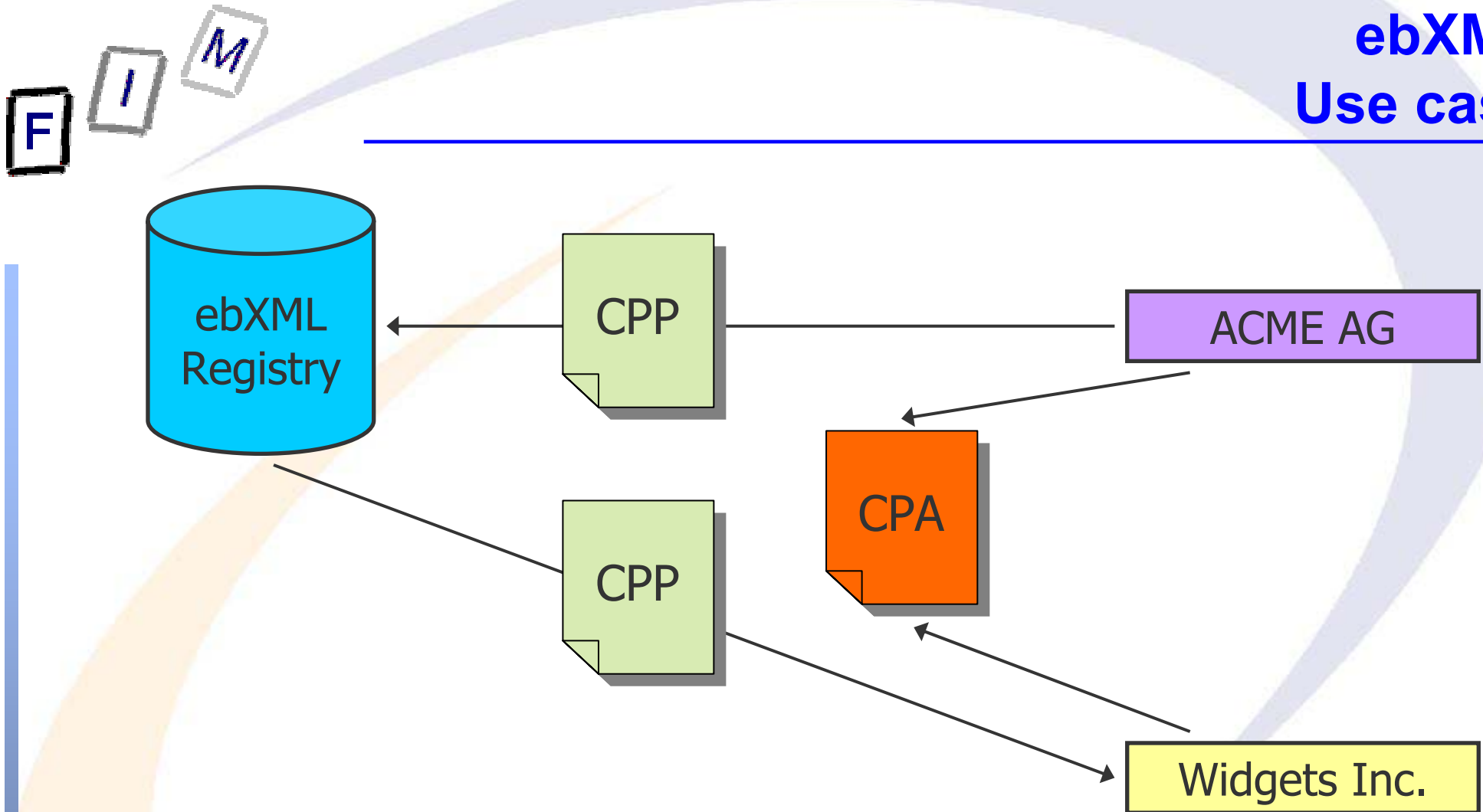
- Both agree on certain processes to be performed
  - → The result is a "CPA" (=Collaboration Protocol Agreement)
    - » This could be derived automatically if desired and both provide CPP's!
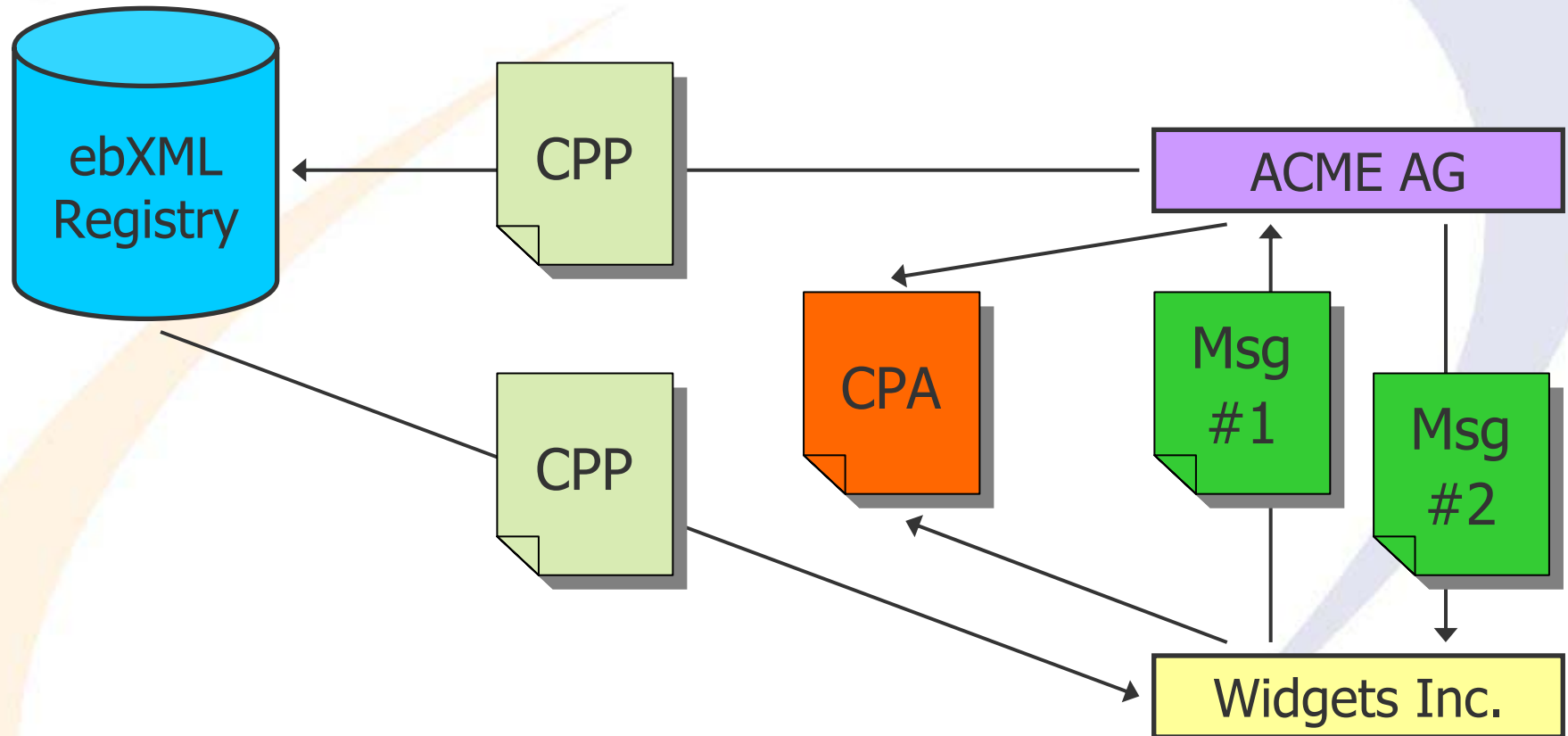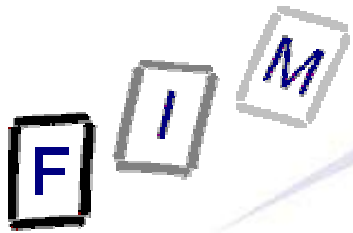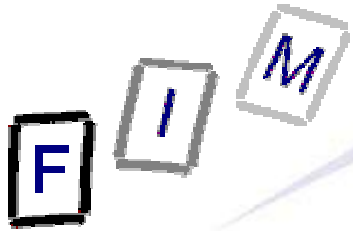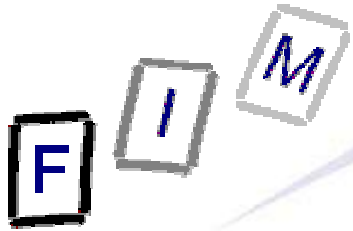
- ACME and Widget exchange messages and perform the processes/protocols they agreed upon
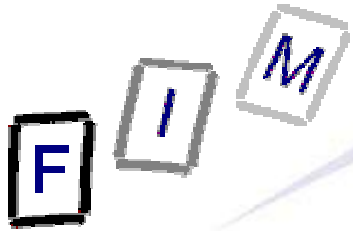
- Documents shown are clearly defined, verifiable, secure, ...
- Software on both sides is standard and unmodified
  - → Both must provide a data mapping to their local systems only!

- Specifies what a party can do:
  - → Business Transactions: Models the processes and the roles
    - » Separate specification
  - → Delivery Channel: Message sending/receiving characteristics
  - → Document Exchange Layer: Encryption, signature, reliability
  - → Transport Layer: Transport protocol, endpoint address
- Contains:
  - → Contact information: company can consist of several "parties"
    - » E.g. office supply and production supply departments
  - → Security details: what certificates/where to find, public keys, ...
  - → Message structure: what messages consist of
  - → Comments

- Specifies what the parties will do:
  - → Describes the visible (i.e. external and enforceable) interactions between the parties
  - → Independent of the internal processes of all parties
    - » Mapping should be simple because everything exchanged is XML!
- Intersection of the CPP's of the trading partners
  - → Must be mutually agreed
  - → Automatically derived or manually negotiated
- Contains:
  - → Additional status value (proposed, signed, ...)
  - → Lifetime: Consent may be time-limited
  - → Constraints on conversations: E.g. at most ? concurrent, ? total maximum, …
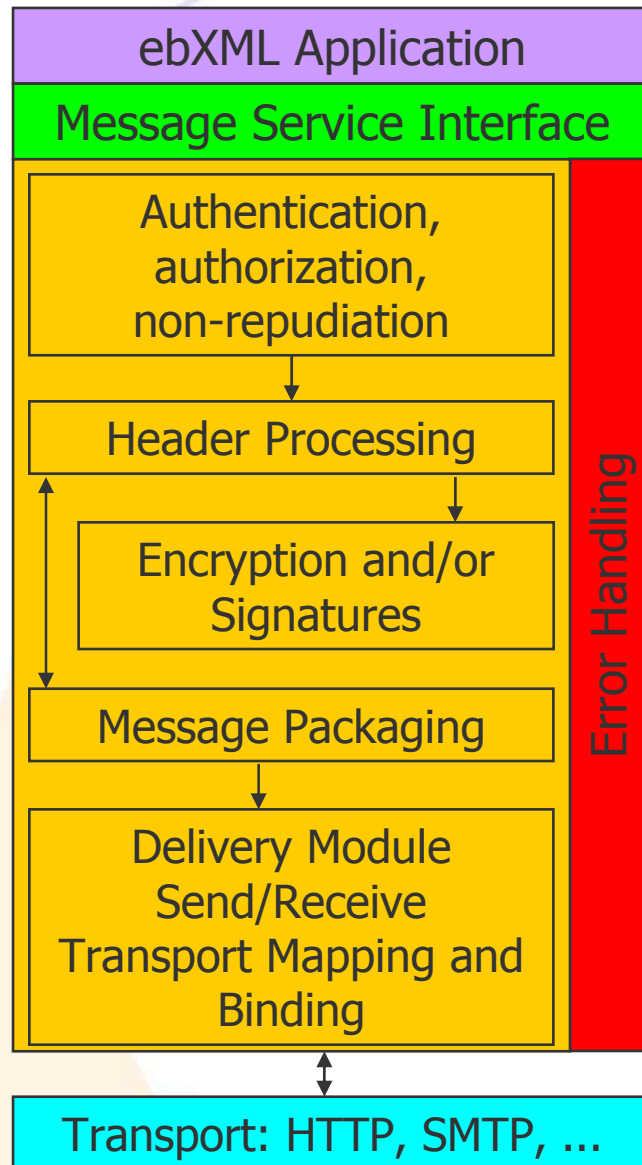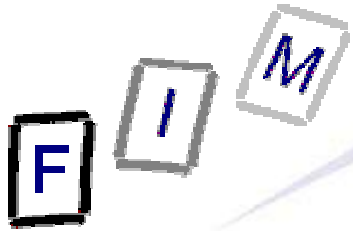
- Secure and reliable exchange of messages between parties
  → Message structure for packaging the actual content
    » "Just another" kind of packaging
  → Behavior of message service handler (MSH)
    » This is something "new"!
  → Doesn't define a spec. interface (data format + semantics, not API)!
    » Can be implemented in Java or C, with different names, etc.!
- Extension to SOAP and "SOAP Messages with Attachments"
- Optional: Reliable messaging
  → Ensuring messages are delivered e.g. exactly once
  → Ensuring information survives any single failure
- Optional: Message ordering
  → Ensuring messages arrive in specific order (at application!)
- Optional: Multi-hop messaging (using intermediaries)

Message transfer schematic

| ebXML Application |
| Message Service Interface |

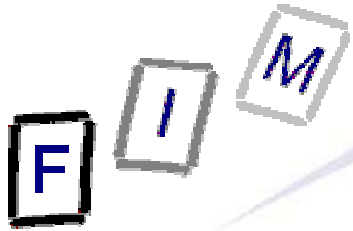| Authentication, authorization, non-repudiation |
| Header Processing |
| Encryption and/or Signatures |
| Message Packaging |
| Delivery Module Send/Receive Transport Mapping and Binding |

Error Handling

Transport: HTTP, SMTP, ...

→ Message Service Interface: Defined by the actual ebXML implementation
  » Independent from the application!

→ Header Processing: Creation/Parsing of ebXML headers
  » Uses the CPA

→ Message Packaging: Final packaging of ebXML into SOAPwA (Enveloping, ...)
  » Headers + Payload (= business content)

→ Delivery Module: Actual connection using certain protocols, incl. acknowledgments
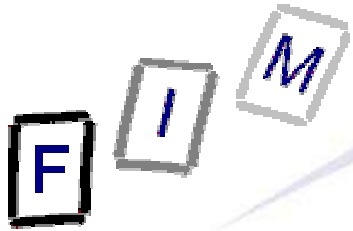 » Persistence, retries, error notifications, ...

- Some implementations (and applications) exist
  → Mainly from large companies and for enterprise systems
  → Case studies seem to be not that frequent currently
- Free software also partially available
- JAXR: Java API for XML Registries
  → Can access XML registries, also supports ebXML and UDDI

- Still nothing for the "small company" out of the box
  → But selected applications can be created probably rather easily

- Problem: Better than EDI, but still expensive to setup
  → Integration, specification of documents, administration, …

- UDDI: Universal Discovery, Description & Information Protocol
  - → For finding the location/available procedures to call
    - » "Yellow pages" of web services providing metadata about them
- WSDL: Web Service Description Language
  - → Describing the procedures available
- SOAP: Simple Object Access Protocol
  - → RPC (Remote Procedure Call) protocol over the Web
   - » Other styles also supported, e.g. free messaging

For all three "services" alternatives exist, but these seem to be those most widely used and the best candidates for the future!

- Communication protocol for message exchange between applications via the Internet
  - → Usually sent via HTTP, but other transports possible (e.g. SMTP)
    - » Good/Bad: Can get around firewalls by this!
  - → Stateless: Each message stands alone
  - → One-way: There is no response to a message
    - » The other side may, however, decide to send a new message, which might be semantically in reply to this one!
  - → Routing, reliability, etc.: Not addressed!
- Consists of up to three parts:
  - → Envelope: Marks this XML document as a SOAP message
    - » Header (optional): Context info, custom extensions
    - » Body (required): The actual call/response
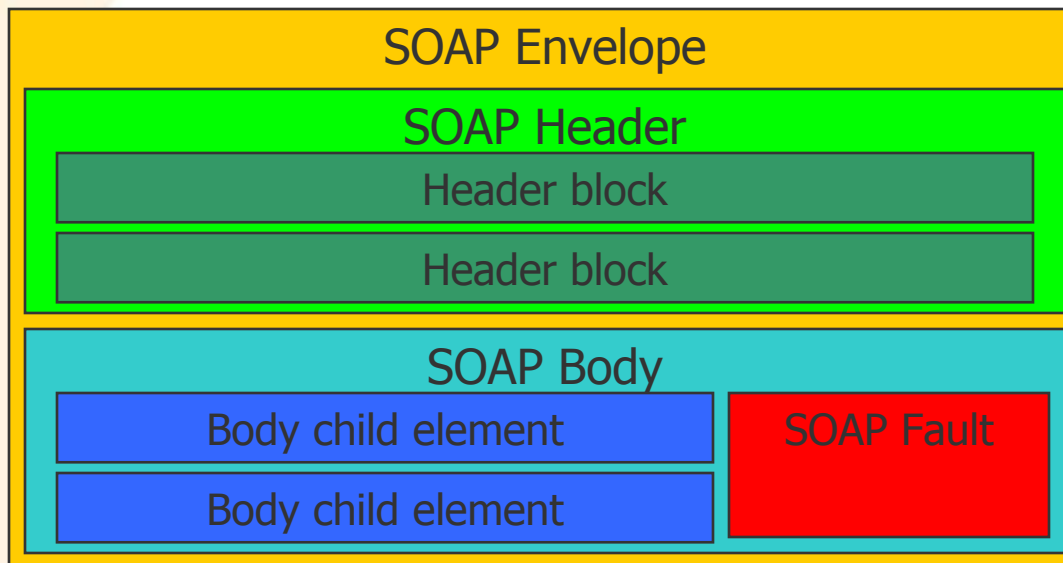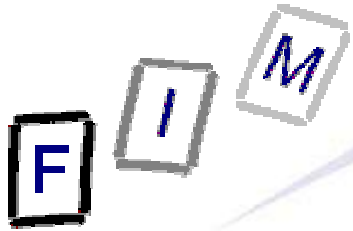   - » Fault (optional): Exceptions, errors, … that occurred during processing
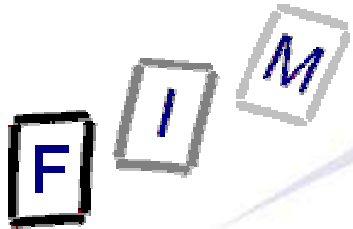
- `<?xml version="1.0"?>`
  `<soap:Envelope` xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding"`>`
     `<soap:Header> ... ... </soap:Header>`
     `<soap:Body> ... <soap:Fault> ... </soap:Fault></soap:Body>`
  `</soap:Envelope>`

Either:

→One or more child elements
→Or a single fault element

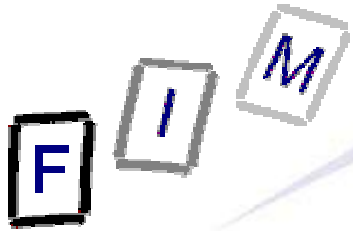| SOAP Envelope	
SOAP Header	
Header block	
Header block	
SOAP Body	
Body child element	SOAP Fault
Body child element	

- SOAP messages may be sent over several "hops"
  - → These are called "intermediaries"
  - → The last recipient is the "ultimate receiver"
- SOAP does NOT specify how these are selected!
  - → No routing; e.g. a header element could contain a route, but each intermediary would have to understand it and act accordingly!
- Processing of header blocks depends on the role of the "node"
  - → Role "next": Header block is intended for the next intermediary
    - » Must be examined, but not necessarily processed!
  - → Role "ultimateReceiver": Intermediaries must ignore this block
  - → Role "none": May not be processed
    - » But might be referenced from a different header block!
  - → Which role a node plays is defined by the application!
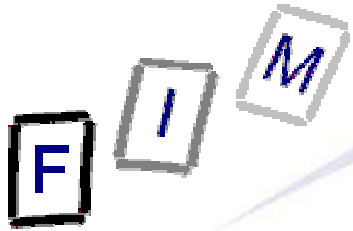   - » Additional roles can be defined by the application

- Special header attributes:
  - → A header block marked as *mustUnderstand* MUST be processed
    - » If it is not understood, a failure message must be created!
    - » The message may not be processed any further
    - » This is intended for important information about block handling
      - – E.g. requiring that it be encrypted before sending it onwards
  - → A header block marked as *relay* MUST be relayed if it is NOT processed by this intermediary
    - » If it is processed, it MUST always be removed
      - – Another header block could however reinsert it!
    - » If it is not processed, it must be passed on to the next recipient
      - – Default would be to remove it, if it is targeted at this intermediaries role!
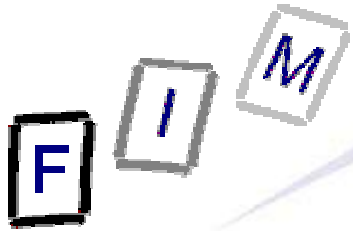
- Describes error during processing of the header or the body
  - → Must contain code and reason
    - » Code may contain subcodes
    - » Reason is a textual description, perhaps in several languages
  - → Can contain node, role (of the node) and detail
    - » Node: The node that generated the fault (important for intermediaries)
    - » Detail: Application defined information; can be any XML content
- The only child element within a SOAP body
- Some faults are predefined
  - → VersionMismatch, MustUnderstand, DataEncodingUnknown
  - → Sender: Incorrect message; do not resend
  - → Receiver: Recipient problem; may be resent identically

- Theoretically SOAP need not be represented as XML
  - → It just hast to adhere to the "XML Infoset", which is more or less the "content" and "style" of XML (is abstract data set)
- Similarly, SOAP need not be sent by HTTP (e.g. SMTP possible)
  - → This is the main protocol, however, and the only one standardized!
  - → GET: Send a simple request and expect a SOAP reply
    - » No SOAP request possible, only a simple URL
    - » The "object" identified by the URI should stay the same (read-only)!
  - → POST: Sending a SOAP request and expecting a SOAP reply
    - » The "object" identified by the URI can also be modified (read-write)!
  - → In both cases the SOAP reply (=complete XML document) is just sent instead of the HTTP content (after the HTTP headers)
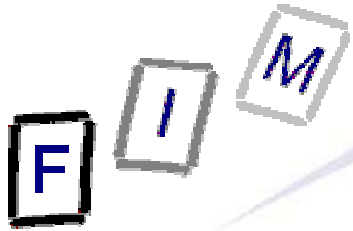   - » This also applies to the request for HTTP POST

- Connection test message ("Ping") from a real system
  → Described using WSDL
  → Transmitted by SOAP over HTTP
- PingPong.wsdl: Description of this call
- Ping.txt: Sending a RPC command
  → SOAPAction: Specific additional SOAP header; describes action
  → POST command: Wants to send additional details
    » In this case: Procedure parameter "message" of type string
      – Value: "Hello"
- Pong.txt: Reply to the command
  → New SOAP message with the parameter "result" of type string
    – Value: "Test.ping.Ping2ResponderAgent responds to Hello"
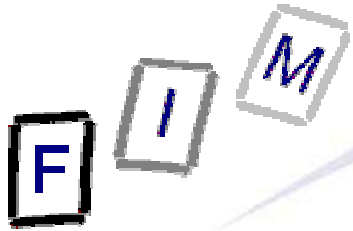
PingPong.wsdl, Ping.txt, Pong.txt

- For describing the syntax of web services and their locations
- Consists of the following elements:
  - → Definitions: Root node for defining several services
  - → Service: Specifies the ports which can be used for bindings
    - » Connecting ports and bindings
    - » Port: Single endpoint address for a binding (e.g. URL)
  - → Binding: How an operation is invoked
    - » Protocol and data formats for the operations and messages
    - » How to encode the parameters; RPC or messaging
  - → PortType/Operation: Description of the action(s) ("Procedure name")
    - » Specifies input and output message
  - → Message: Definition of the data communicated
    - » What will be sent over the network as a unit; data types of content
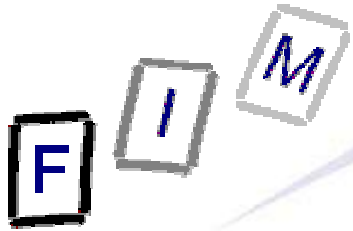 - → Types: Defines complex data types used (e.g. in messages)
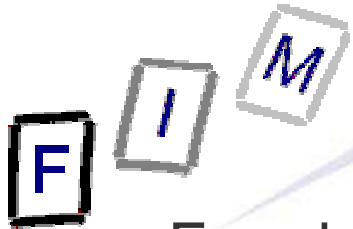
- Four kinds of transmission
  - → One-way: Send a message; no response received
  - → Request-response: Send and wait for result
  - → Notification, Solicit-response: Reverse op.; not completely specified
- Bindings defined for SOAP, HTTP and MIME
  - → HTTP: Pure; different from SOAP transported by HTTP!
  - → MIME: E.g. sending it directly in E-Mail attachments
- From WSDL necessary classes can be generated automatically!
  - → Stub (Proxy): On the client side; called by the application
    - » Packs everything into XML (from the native programming language)
  - → Skeleton: On the server side; called by the SOAP implementation
    - » Calls the actual implementation (or is replaced by it)
  - → Helper classes: For any datatypes specified, locating endpoints, …
- This makes RPCs over SOAP transparent for the application!
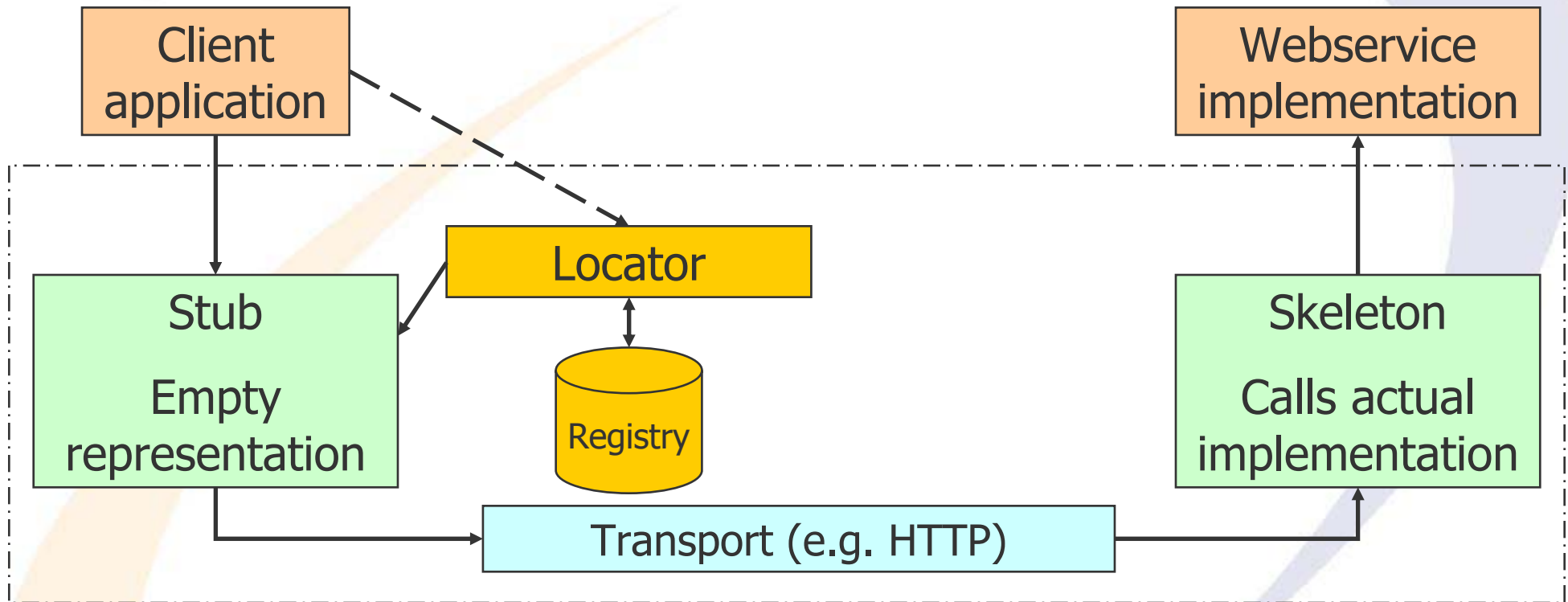 - → With some restrictions: error handling, acquiring the endpoint
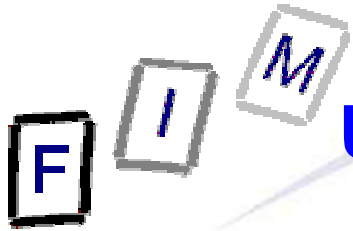
WSDL and programming

- AXIS: One Java XML framework for web services
  - → Supports WSDL and SOAP (with attachments)
    - » And some other goodies!
  - → From WSDL description, Java code is generated automatically
    - » Only the actual "servicing code" must be filled in
- In general the following parts are required (unless optimized!)
  - → Stub: Client side representation
    - » Packages everything into a SOAP message and unpacks the result
  - → Skeleton: Server side representation
    - » Unpackages the request, calls the actual implementation and packages the result (or the error) into a SOAP message
  - → Locator
    - » For locating the server to call (explicitly, UDDI, manually, etc.)
  - → Additional classes (e.g. custom types, interfaces)

Client application

Webservice implementation

Locator

Stub

Empty representation

Registry
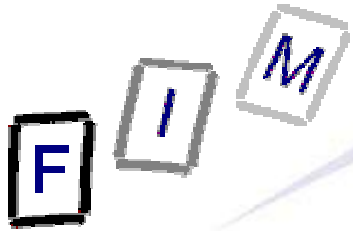
Skeleton

Calls actual implementation

Transport (e.g. HTTP)

- Stub must look exactly like the actual implementation for the client
- Skeleton calls actual implementation with exactly the same parameters and accepts all results (incl. all exceptions, errors, ...)
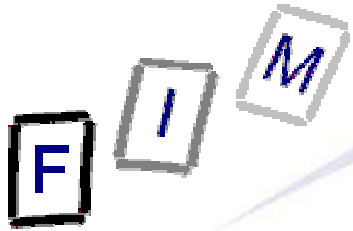- The whole subsystem must be invisible to client and application

- Inquiry and discovery
  - → Four core types
    - » businessEntity: Information about the parts publishing the data
      - – E.g. Name, description, contact information
    - » businessService: Describing a particular family of technical services
      - – E.g. Purchase: Submission, confirmation, notification
    - » bindingTemplate: Technical information on a a single service
      - – E.g. entry point, content specification
    - » tModel: Descriptions of specifications for services or taxonomies
      - – E.g. webservice type, protocol, category system; could be WSDL
      - – Not contained within the registry itself; this contains only pointers to it!
  - → Everything is accessed by unique identifiers (UUID)
- Security
  - → Access control through a policy
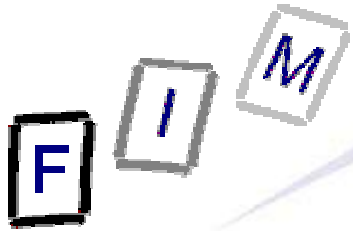 - → Support XML signatures verify integrity/publisher

- Replication
  - → Support for messages to keep several registries with same content
- Subscription
  - → Publishing: Uploading information to a registry
  - → Subscribing: Notification on changes within the registry
- Internationalization: Mostly provided by XML already
  - → Uses the "xml:lang" attribute
  - → Supports language dependent collation
- UDDI provides both data structures and API!
  - → Inquiry, publish, …: WSDL specification
  - → Communication with registry: To be implemented with SOAP
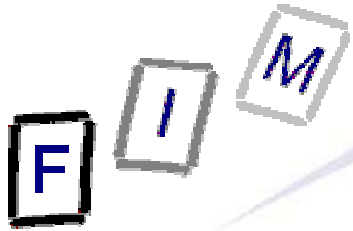
Web service example: Amazon.com

- Allows users to interface and extract lots of data
  - → One-way service only: Retrieving/Querying data only
    - » One exception: Sending a shopping cart back to Amazon so visitors to your site can directly purchase goods selected on your site!
  - → New extension: Search engine integration (Alexa.com)
    - » Web search, URL information, crawl metadata (size, checksum, links, images, ...), web map (list of in- and outbound links)
      - – Limited to 10000 requests/day per subscription ID
      - – Free for now, but will be charged for later
- Available data:
  - → Product data: availability, price, size, image,....
  - → Customer content: Reviews, wish lists, lists
  - → Seller information: General info and customer feedback
    - » For third-party shops/sellers affiliated with Amazon
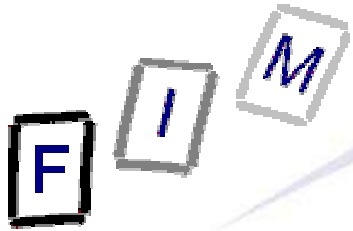 - → zShops and Marketplace product data

- Several restrictions apply, e.g.
  - → No copying of content (selling, redistribution, price extraction, ...)
  - → Usage restrictions (violence, illegal activities, ...)
  - → Frequency limit (1 call/second/IP-address)
  - → Caching limited to 24 hours
- Incentive to use it
  - → Percentage of sales made through this
    - » Put some recommendations on your website, insert a "local" whopping cart and hope your visitors buy from Amazon later
- Interface: REST (=HTTP GET) or SOAP
  - → Result: XML
    - » REST: XSLT transformation with provided stylesheet also supported

More info: http://www.amazon.com/webservices/

- ebXML Homepage
  http://www.ebxml.org/
- Collaboration-Protocol Profile and Agreement
  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-cppa
- CPPA Specification 2.0 (23.9.2002)
  http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf
- Messaging services homepage
  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-msg
- ebXML Messaging Services Specification 2.1 (WD 25.3.2004)
  http://www.oasis-open.org/committees/download.php/6130/wd-ebMS-2_1-04.pdf
- freebXML
  http://www.freebxml.org/
- Mertz: Understanding ebXML
  http://www-106.ibm.com/developerworks/xml/library/x-ebxml/
- Chase: Introduction to ebXML
  http://www-106.ibm.com/developerworks/xml/edu/x-dw-xebxml-i.html

- SOAP Version 1.2 Part 0: Primer
  http://www.w3c.org/TR/2003/REC-soap12-part0-20030624/
- SOAP Version 1.2 Part 1: Messaging Framework
  http://www.w3c.org/TR/2003/REC-soap12-part1-20030624/
- Web Service Description Language (WSDL) 1.1
  http://www.w3.org/TR/wsdl
- UDDI Homepage
  http://www.uddi.org/
- Apache Axis
  http://ws.apache.org/axis/
- Java SAAJ (SOAP with Attachments API for Java)
  http://java.sun.com/xml/saaj/index.jsp
- Java Web Services Developer Pack
  http://java.sun.com/webservices/downloads/webservicespack.html