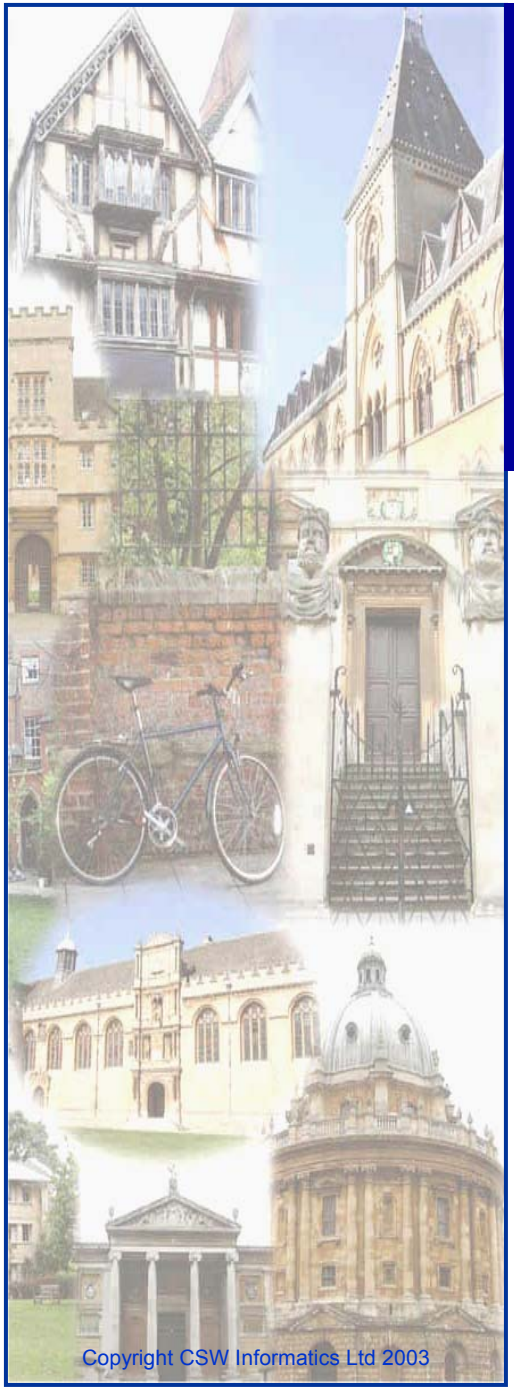


SUMMER XML CSW 2003 SCHOOL

XML for e-Business

Eve Maler
Sun Microsystems, Inc.

Goals for this session

- Learn about the Universal Business Language (UBL) and its significance to, and place in, modern e-business
- Study UBL's design center and underlying model
  - A model that may be useful for many information domains
- Study UBL as an application of XML
  - And its lessons for other large XML undertakings
- Take a look at some real UBL inputs and outputs along the way

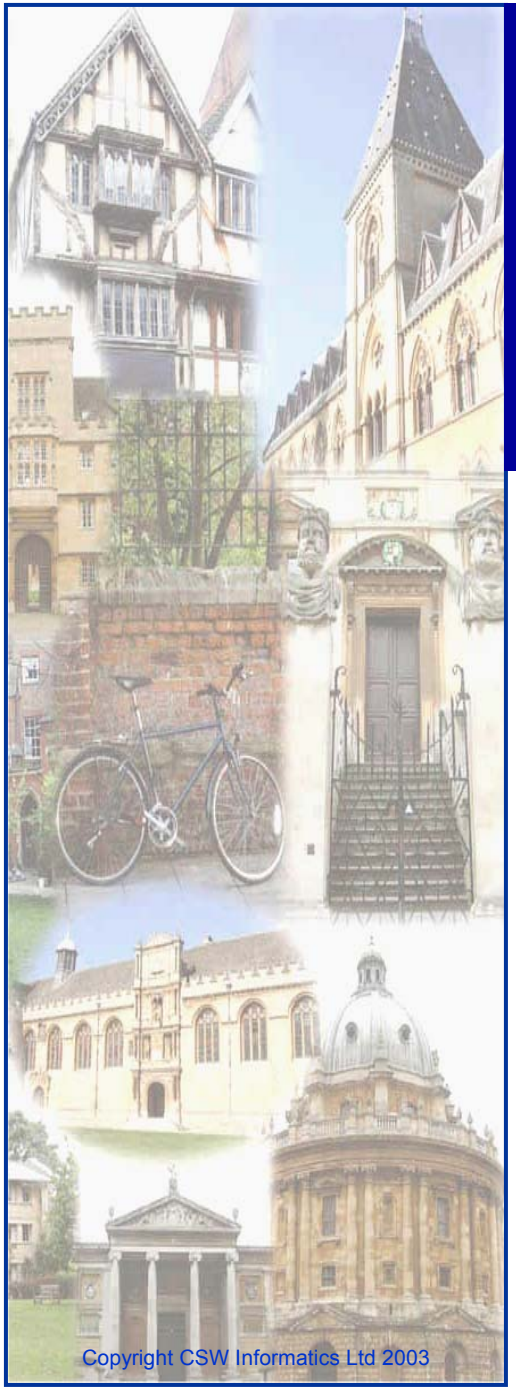
A little about me

- I'm an XML Standards Architect in Web Technologies and Standards at Sun
- I co-founded the OASIS UBL Technical Committee and formerly chaired its biggest technical subcommittee
- In previous lives I helped develop XML itself, DocBook, XLink, Pipeline, and more
- I also coordinate Sun's interaction with XML/web services security standards and take part in several related standards efforts

Agenda

- XML for e-business: why and how?
- EDI, ebXML business web services, and UBL's role
- Making UBL happen
- ebXML Core Components
- The UBL modeling methodology
- Designing the UBL schemas
- Resources

Thanks to Jon Bosak and others in the OASIS UBL TC for content assistance!

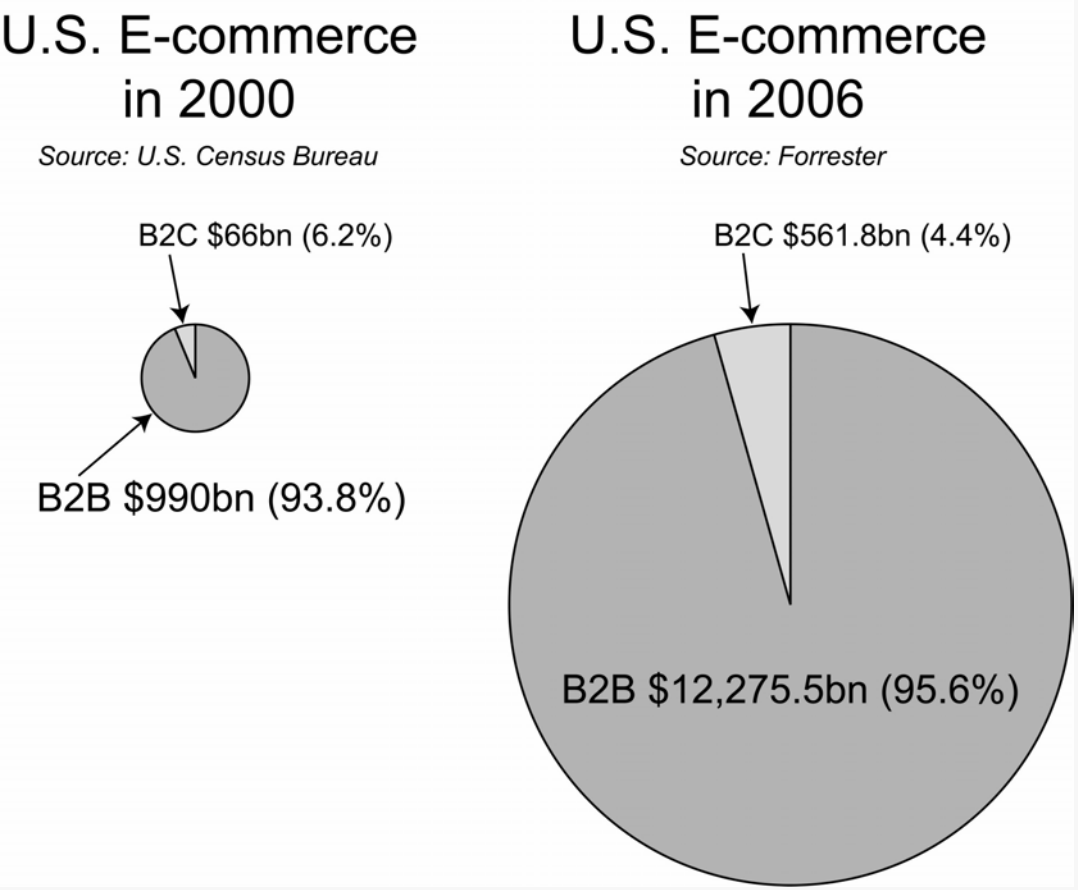


SUMMER SCHOOL XML csw 2003

XML for e-Business: Why and How?

Did you know…?

- E-commerce essentially *means* electronic B2B
- Modernizing and improving B2B can provide huge benefits



Unreasonable goals for B2B

- Magically enable universal interoperability merely through “using XML”
- Reinvent (disrupt?) our concept of what business means
- Abandon existing EDI (Electronic Data Interchange) systems
- Commoditize the universe
- Stop spending lots of effort on business relationships
- Eliminate humans from decision-making

More facts about e-commerce

- It's difficult to take the people out of business process, for reasons of:
  - Trust relationships
  - Error handling
  - Legal action
- Business is built on the concept of standard, legally binding documents
- Legal *intent* requires *meaning*
- XML alone will never give you this

Reasonable goals for B2B

- Web-enable existing paper-based business practices
  - Save money by eliminating re-keying
- Preserve investment in existing systems and allow businesses to migrate at their own pace
- Integrate SMEs into existing EDI-based supply chains
- Maintain a legally accessible audit trail
- Incrementally enable true global market availability

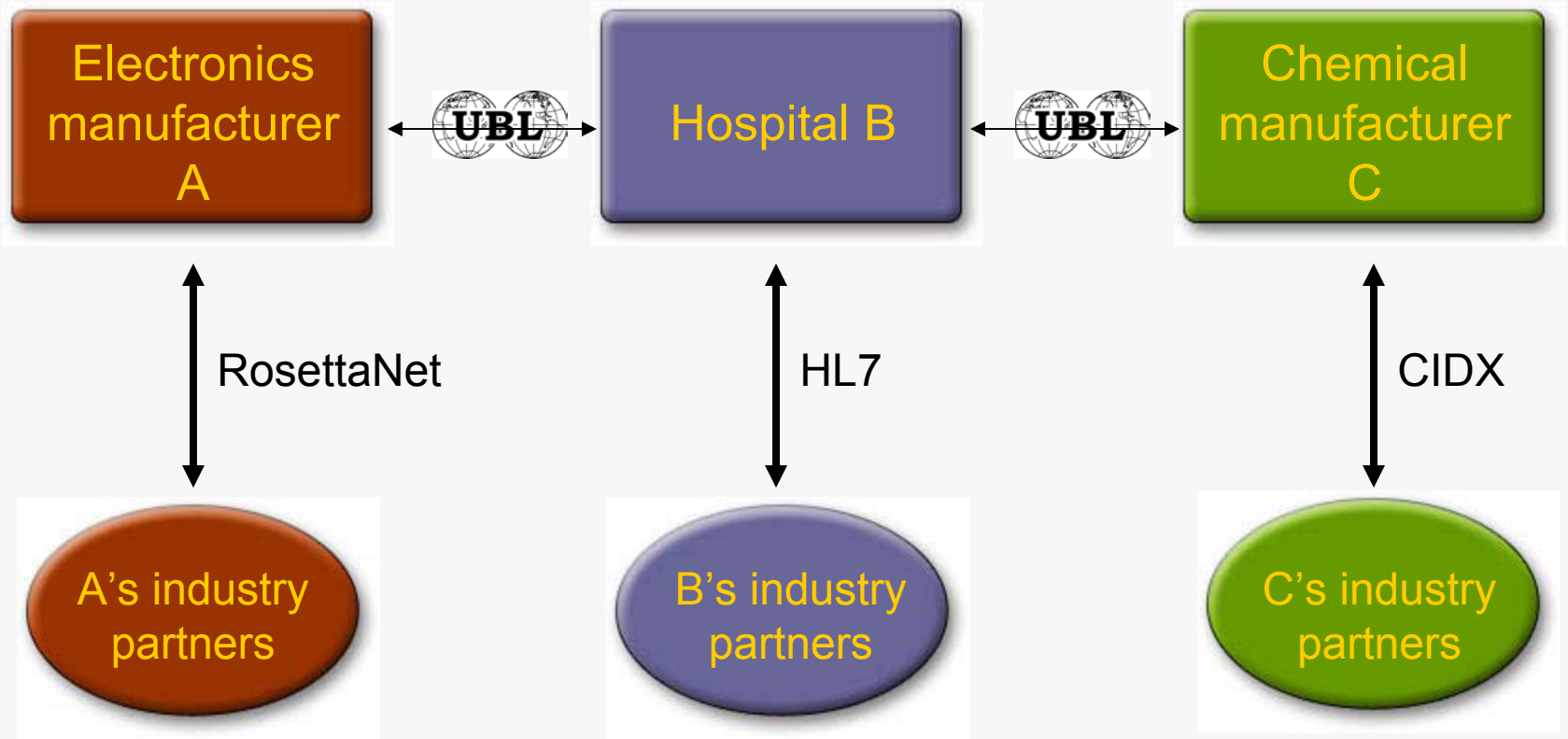
A global XML standard can help achieve these goals

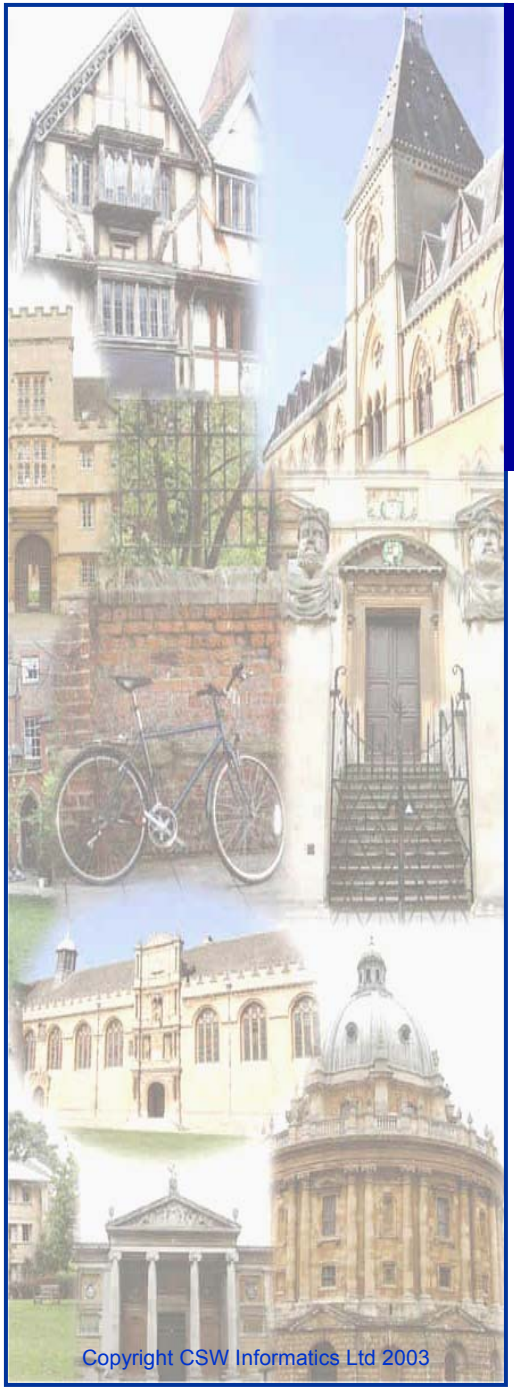
- Lower cost of commercial software
- Easier learning curve
  - Standardized training
  - More skilled workers
- Lower cost of integration through reuse of common structures
  - Universally available pool of system integrators
- Lower overall cost of entry
  - Thus, quicker adoption by SMEs

Enter UBL, the Universal Business Language

- An XML-based business language standard
- Leverages knowledge from existing EDI and XML B2B systems
- Applies across all industry sectors and domains of electronic trade
- Modular, reusable, and extensible in XML-aware ways
- Non-proprietary and committed to freedom from royalties
- Intended to become a legally recognized standard for international trade

UBL's potential fit with existing XML B2B

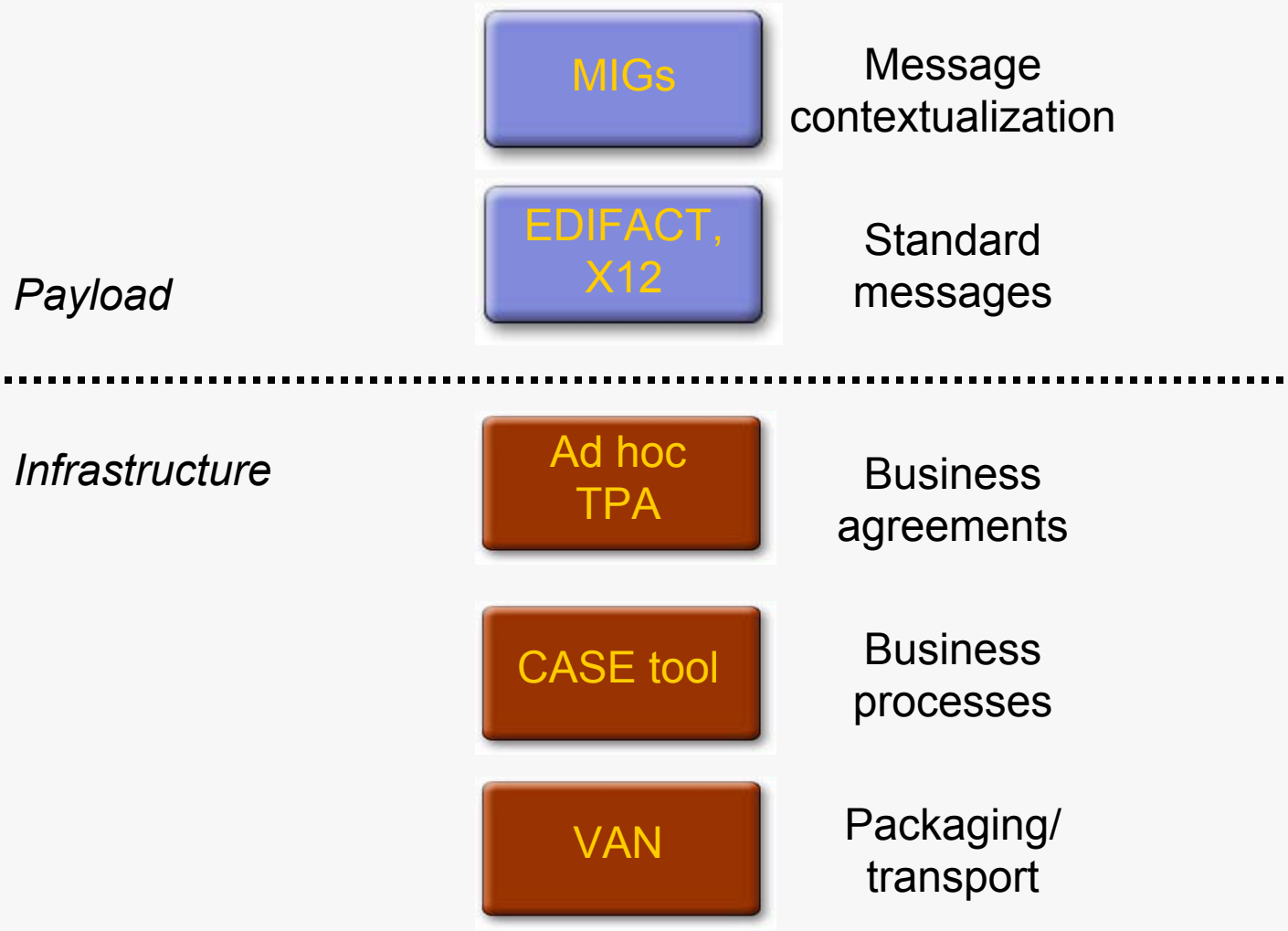




SUMMER XML CSW 2003 SCHOOL

EDI, ebXML Business Web Services, and UBL's Role

The traditional EDI stack



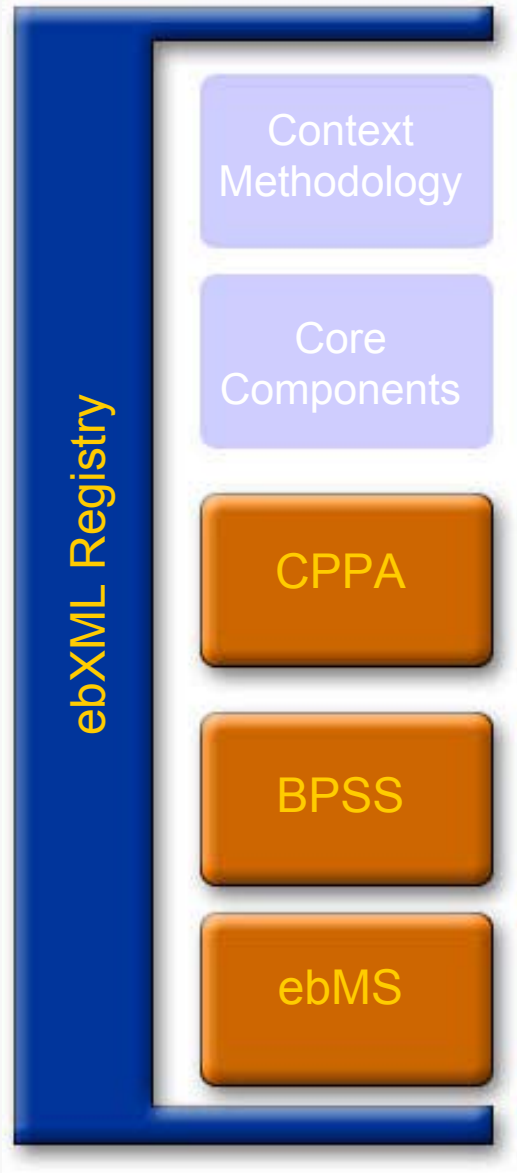
Some EDI pressure points

- Private networks are expensive and require extensive point-to-point negotiation
  - Though AS1 and AS2 mitigate this concern
- The business process data is “soft”, not machine-readable
- The interchange pipeline is large, with infinite possible subsets
- The data for adapting to different business contexts is also “soft”

Enter ebXML, the Electronic Business XML initiative

- A joint 18-month effort, concluding in May 2001, of OASIS and UN/CEFACT
  - The work continues in several forums today
- Over 1000 international participants
- The vision: a global electronic marketplace
- Enterprises of any size, anywhere, can find each other electronically and conduct business by exchanging XML messages

The ebXML stack for business web services



Discovery/ retrieval

Message contextualization

Standard messages

Business agreements

Business processes

Packaging/ transport

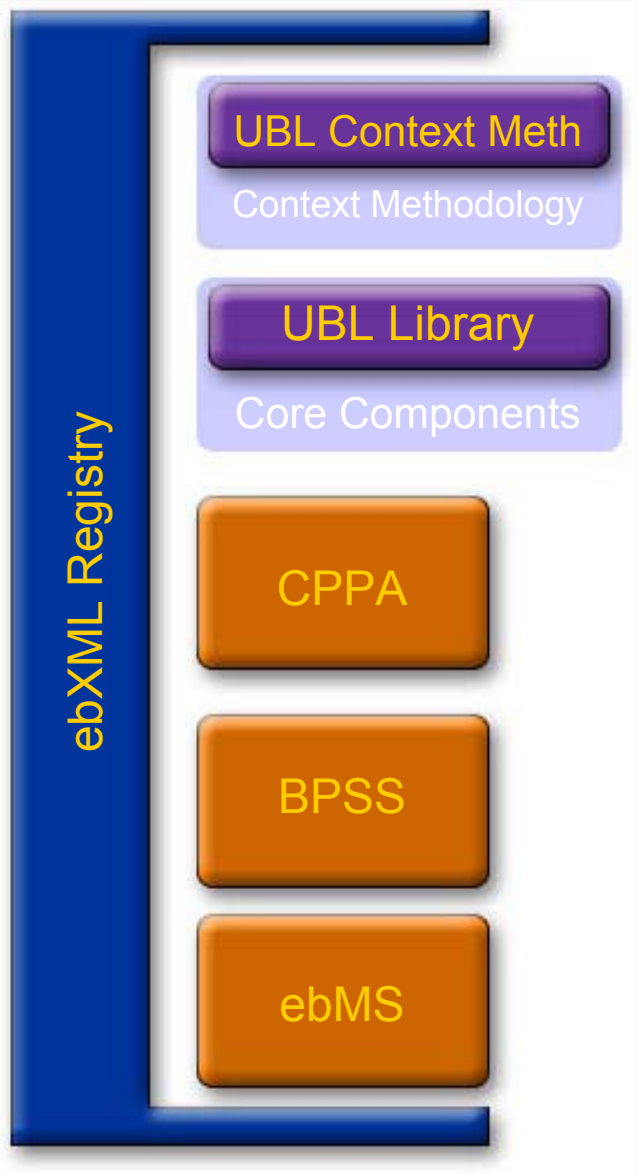
ebXML for infrastructure is basically ready

- Components approved as OASIS Standards:
  - ebXML Message Service (ebMS) V2.0
  - ebXML Registry (formerly “ebXML Reg/Rep”) V2.0
  - ebXML Collaboration Protocol Profile and Agreement (ebXML CPPA) V2.0
- Business Process Schema Specification (BPSS) work is ongoing in UN/CEFACT
- Many implementations and interoperability/test events to date

ebXML for the payload is proceeding, but conceptual

- The ebXML Core Components Technical Specification is at V1.90
  - Syntax neutral and ready for mapping
- This includes the Context Methodology work
  - Again, syntax neutral rather than syntax bound

UBL proposes to flesh out the ebXML stack

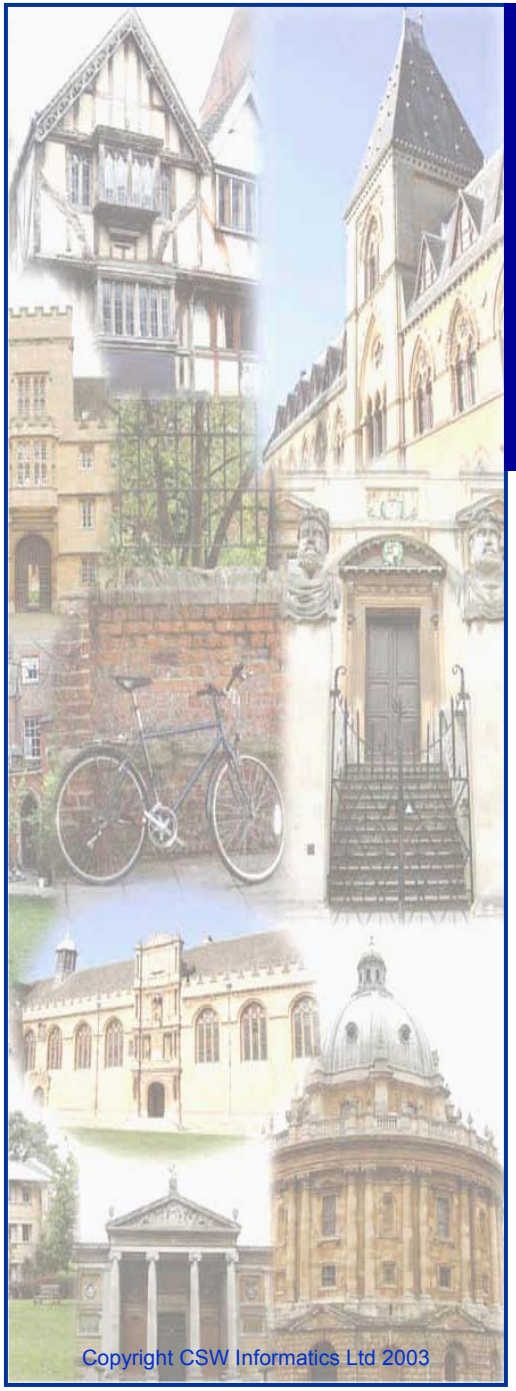


The basic requirements

- Semantic clarity through a binding from Core Components to a syntax
- Choosing XML as that syntax!
- Royalty-free IPR
- Usable “on the cheap”
- No ties to particular back-end implementations
- Urgency
- Allow for contextualization

The special requirement for context

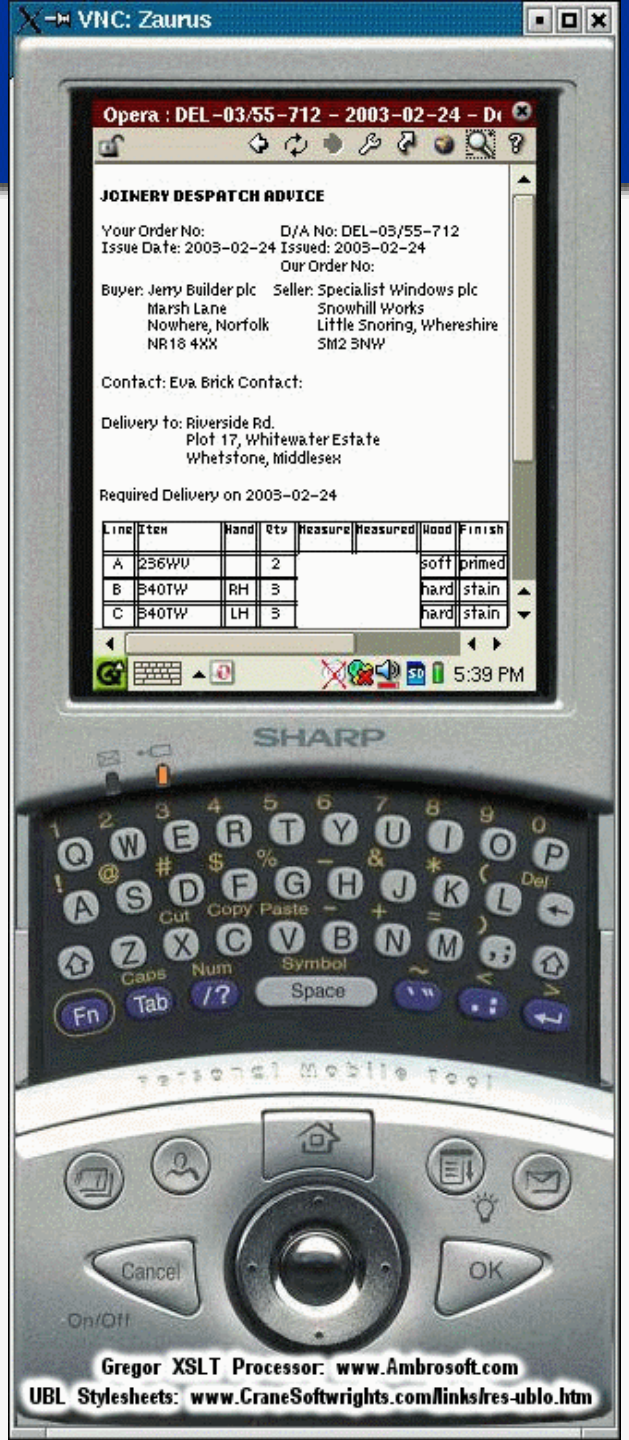
- "Standard" business objects need to be different in different business contexts
  - Addresses in Japan and the U.S. have different fields
  - In some industries, addresses need GPS coordinates rather than streets
  - Invoice items for shoes need to provide size information; for coffee, roast information
- These differences need to be accommodated without sacrificing interoperability



SUMMER SCHOOL XML csw 2003

Making UBL Happen

UBL really is happening!



The standards venue

- UBL is being developed in an OASIS Technical Committee (TC)
- OASIS offers:
  - An objective process
  - Openness of its work to public view in real time
  - Easy and inexpensive opportunities to join
- Jon Bosak is the chair and main founder
- The membership is diverse, including:
  - Users, vendors, and governments
  - XML and e-business experts

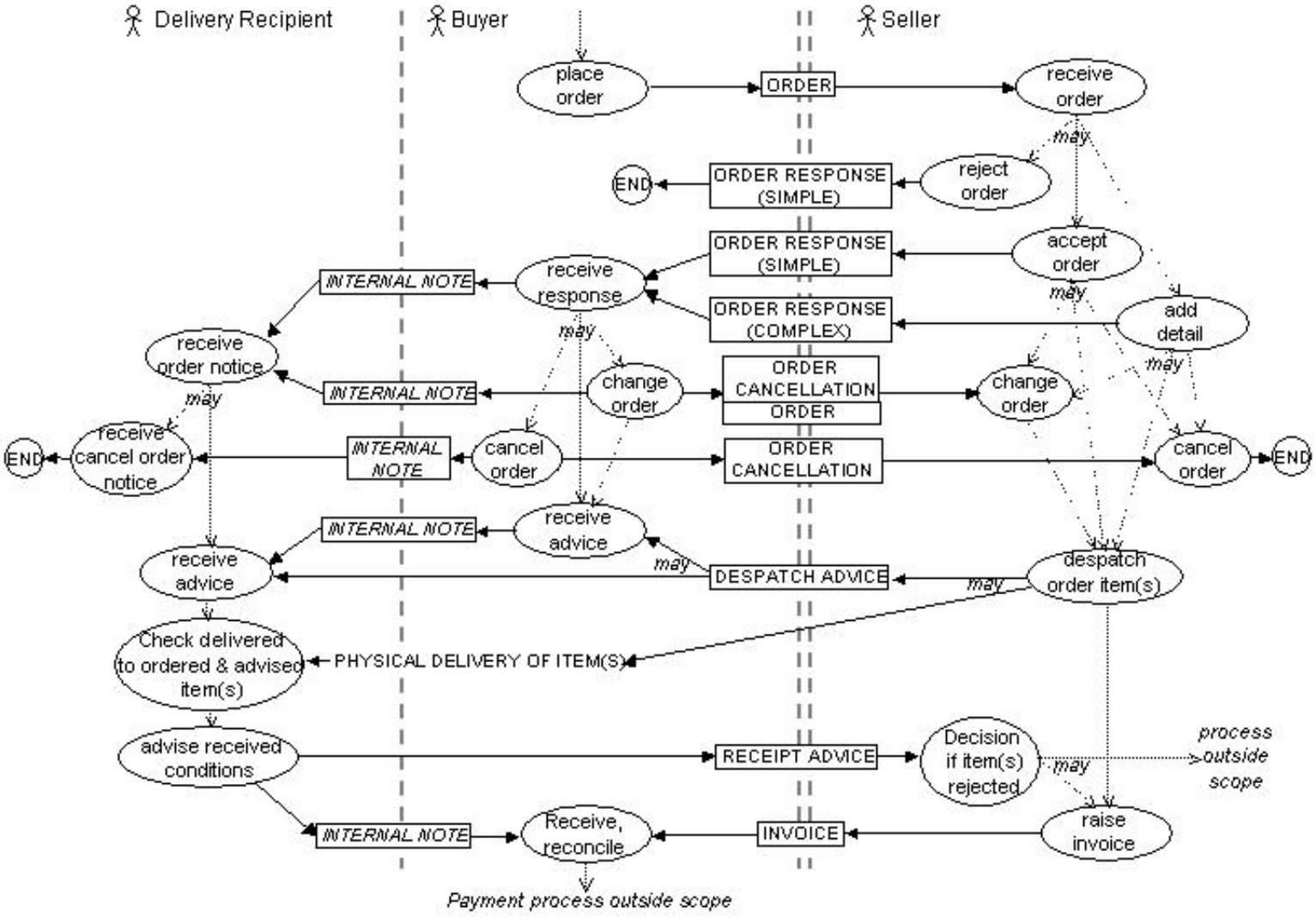
Formal liaisons (so far)

- ACORD (insurance)
- ARTS (retail sales)
- ebXML Asia Committee (ebXML)
- e.centre (EAN UK)
- EIDX (electronics)
- HL7 (healthcare)
- Information Technology Standards Committee of Singapore
- NACS (convenience stores)
- Open Applications Group
- RosettaNet (information technology)
- SWIFT (banking)
- UIG (utilities)
- VCA (optical supplies)
- XBRL (accounting)
- ASC X12 COTG
- UN/CEFACT TBG
- UN/CEFACT ATG
- OASIS eGov TC
- OASIS CIQ TC

Business documents addressed in UBL

- The initial draft (V0p70) includes these trading cycle documents:
  - Common building blocks
  - Order
  - Order response (simple)
  - Order response (complex)
  - Order cancellation
  - Despatch advice
  - Receipt advice
  - Invoice
- Others will follow for materials management, payment, transport/logistics, catalogs, etc.

The trading cycle scenario



Deliverables

- The UBL Library
  - Data model in spreadsheet form
  - Normative W3C XML Schema (XSD) modules
  - Non-normative UML class diagrams and ASN.1 schemas
- Schema naming and design rules
- Modeling methodology
- Simple (for now) context methodology
- Stylesheets for viewing and printing
- perl scripts for generating the schemas
- Sample XML instances and outputs
- Additional documentation

The work is done by subcommittees

- Modeling and content
  - Library Content (**LC**)
  - Context Drivers
- XML representation and mechanisms
  - Naming and Design Rules (**NDR**)
  - Context Methodology
  - Tools and Techniques
- Administrative functions
  - Marketing
  - Liaison
  - Subcommittee Chairs

The UBL timeline

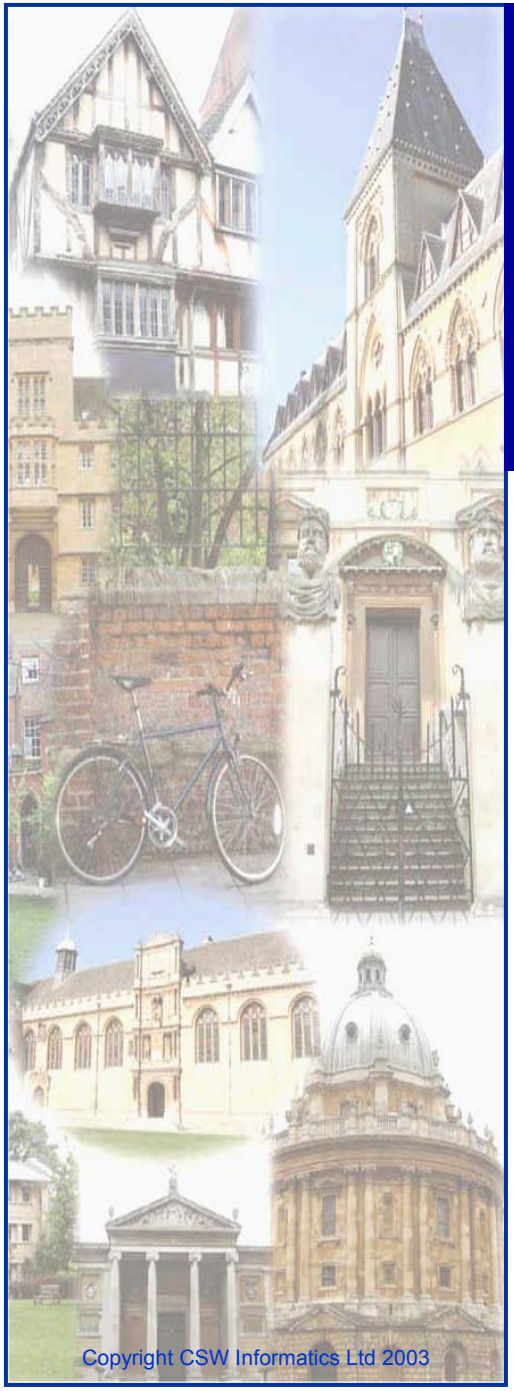
- The V0p70 review period is nearing its end
- V0p80 was scheduled for release in June 2003, specifically for review of RosettaNet and eGov issues
- The plan calls for a final UBL V1.0 release in mid-October 2003

Development strategies

- Start with the low-hanging fruit
  - Focus on the 20% of documents and business objects actually used by 80% of e-business partners
- Defer the rocket science
  - Produce useful, concrete outputs ASAP
- Don’t start with a blank slate
  - Work from xCBL V3.0 (with no expectations of backwards compatibility)
- Take advantage of XML and business expertise

Some additional principles

- Straightforward Internet use
- Account for usage of “various and sundry” tools
- Provide only one way to encode information
- Try to be prescriptive, within reason for interchange
- Leverage XML technology
- Be cautious about foreign namespace dependencies

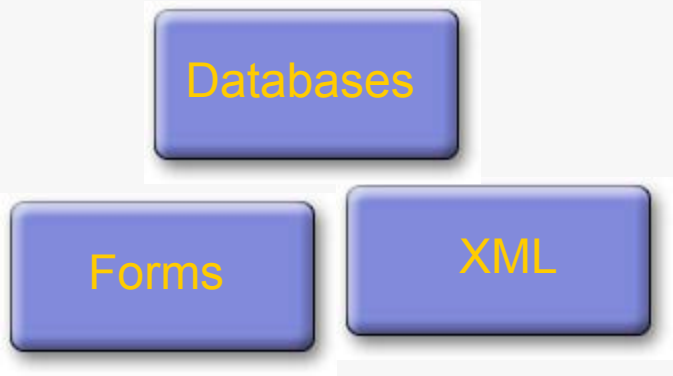


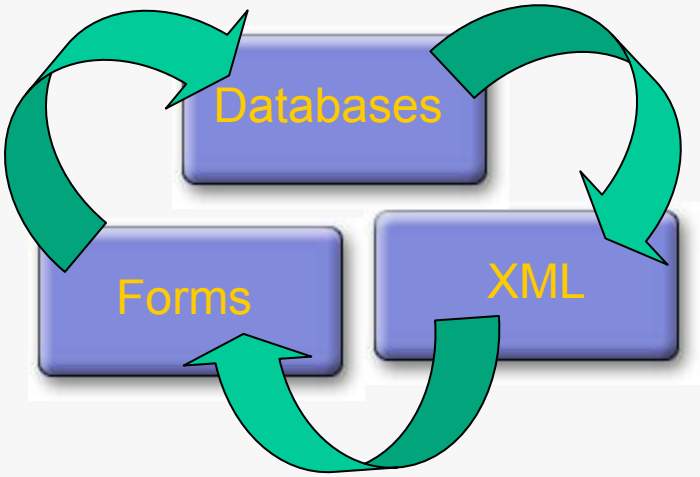
SUMMER SCHOOL XML csw 2003

How the ebXML Core Components Work

Why base UBL on ebXML Core Components?

- The Core Components substrate allows for *correlation* between different syntactic forms of business data that has the same meaning and purpose

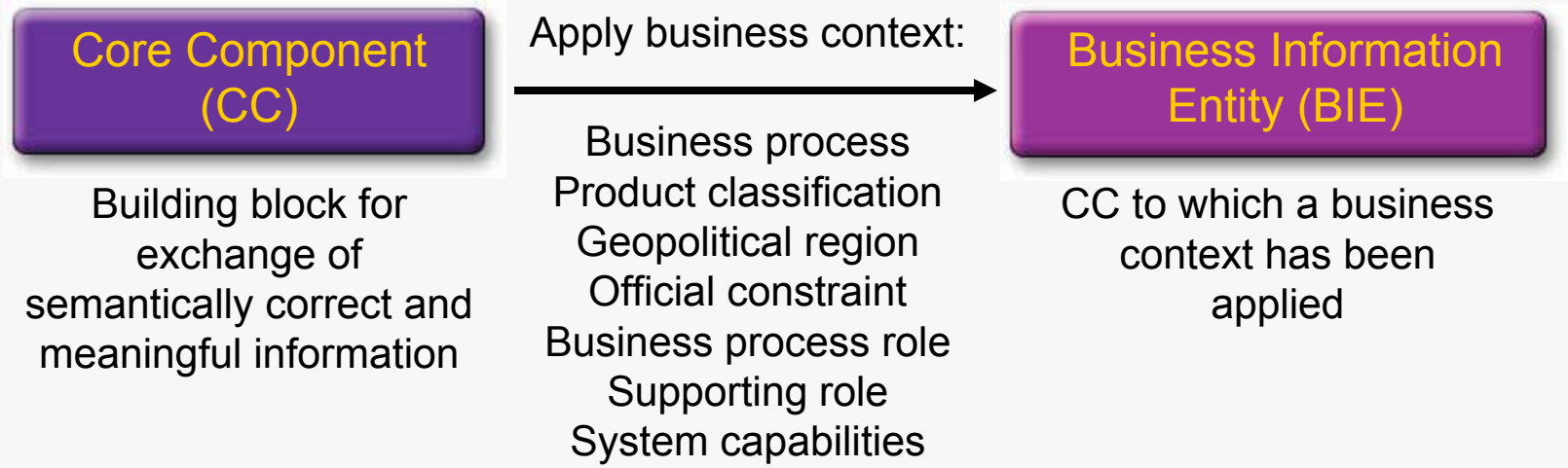




The Core Components specifications

- The Core Components Technical Specification (CCTS) defines a syntax-neutral metamodel for business semantics
- Work is ongoing to define an actual (syntax-neutral) data dictionary based on CCTS
  - Core Components Supplementary Documents (CCSD)
  - Currently non-normative
- UBL is, first and foremost, striving to use the CCTS metamodel accurately

Core components vs. business information entities



- If “address” is defined as a generic CC…
- …a BIE with the geopolitical region set to “U.K.” might be a “U.K. address”
- UBL deals only in BIEs because it sets the business process
  - So we’ll stick to that terminology

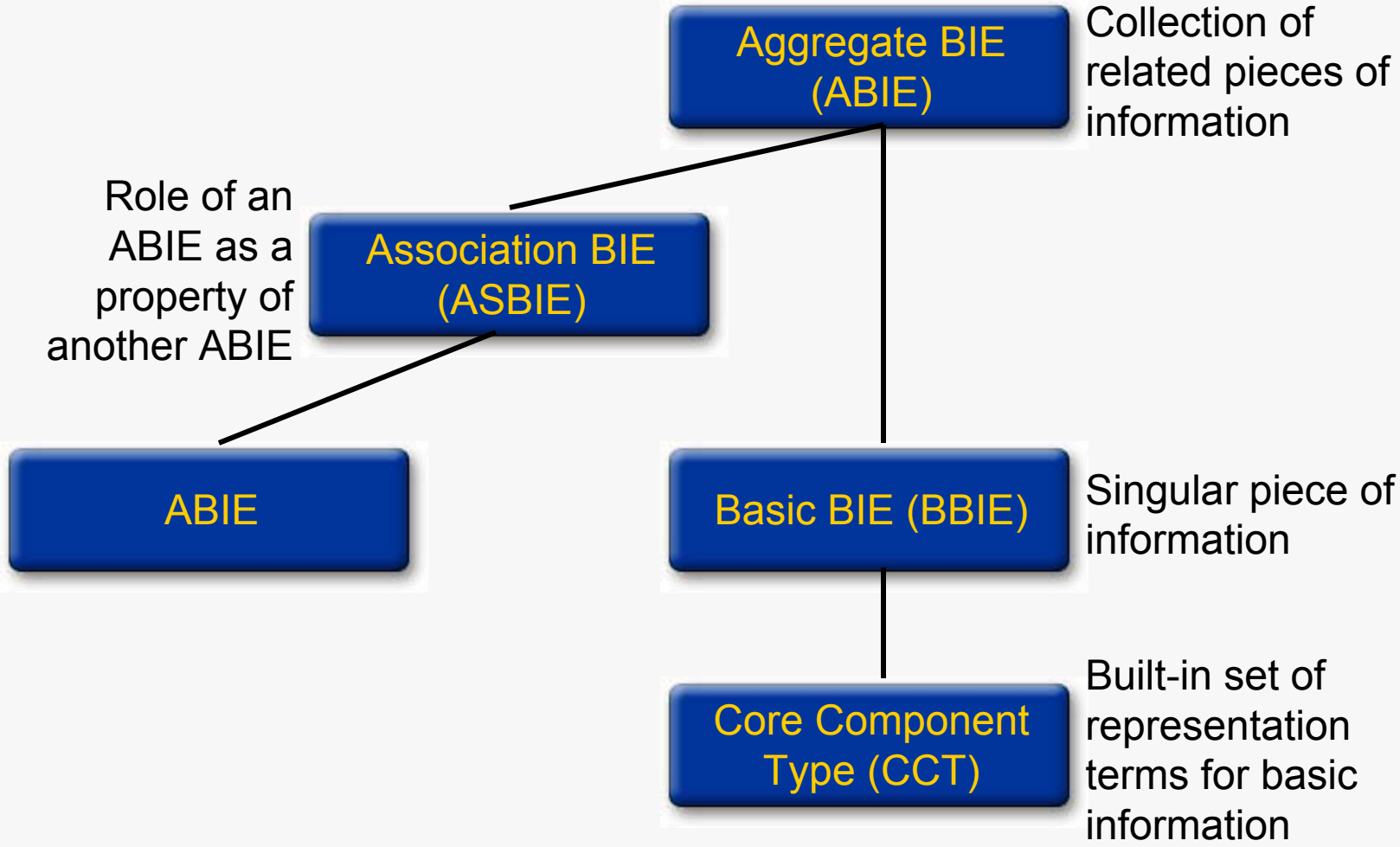
The CCTS follows ISO 11179

- A standard OO-friendly basis for precision in describing pieces of business information and their relationships
- Governs how to define data dictionaries for *object classes*, *properties*, and *representation terms*
- A tiny sample dictionary for illustration (cardinality elided for simplicity):

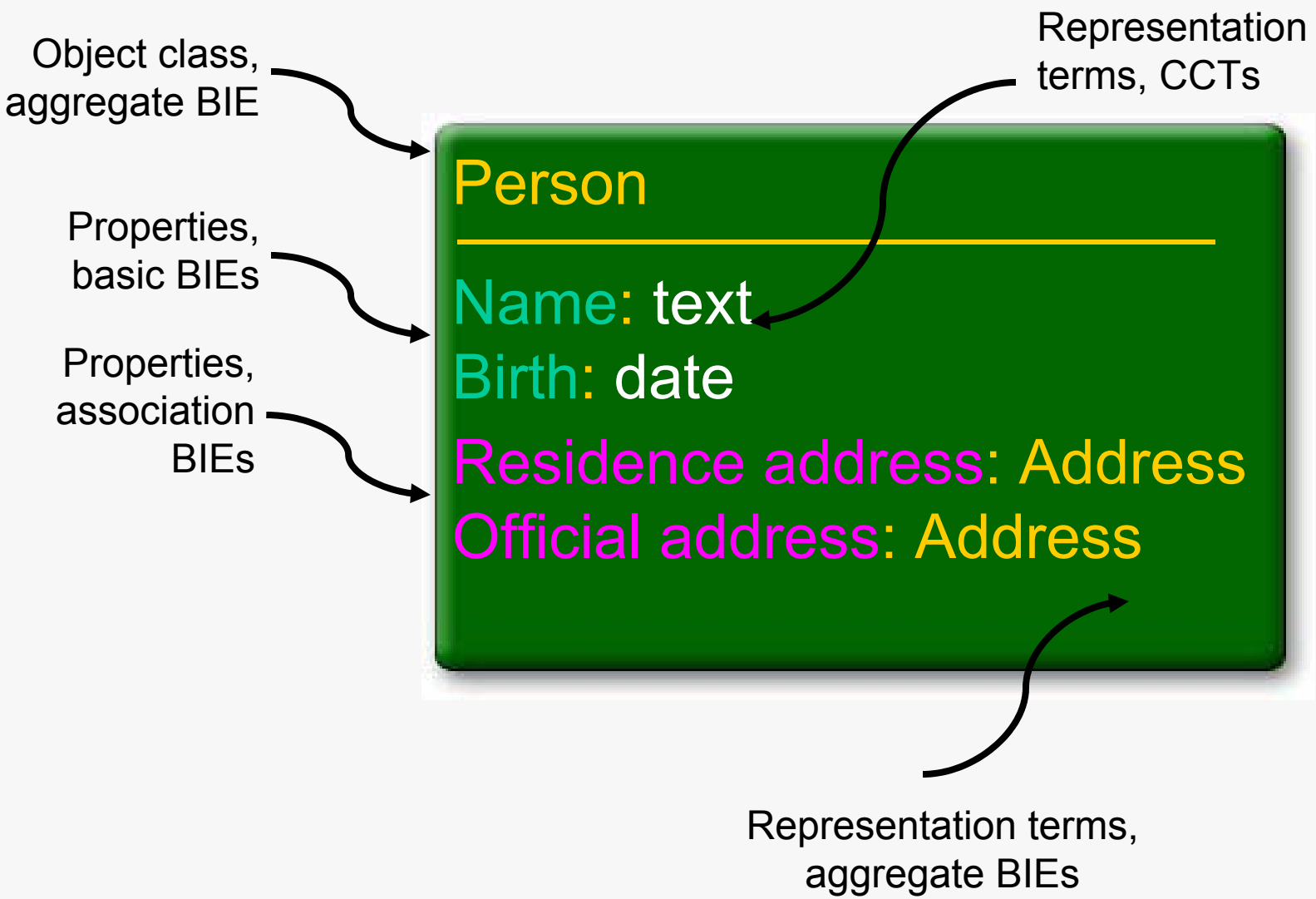
Person
Name: text
Birth: date
Residence address: Address
Official address: Address

Address
Street: text
Town: text
Country: identifier
Post code: text

Summary of the BIE (and CC) system



Mapping our example to the BIE system



The set of Core Component Types

- Conceptually similar to W3C XML Schema built-in types
  - But they don't come with pre-assigned syntactic constraints
  - And they are themselves "complex": primary content plus supplementary metadata

- Amount
- Binary Object (plus Graphic, Picture, Sound, and Video)
- Code
- Date Time (plus Date and Time)
- Identifier
- Indicator
- Measure
- Numeric (plus Value, Rate, and Percent)
- Quantity
- Text (plus Name)

Giving unique names to dictionary entries

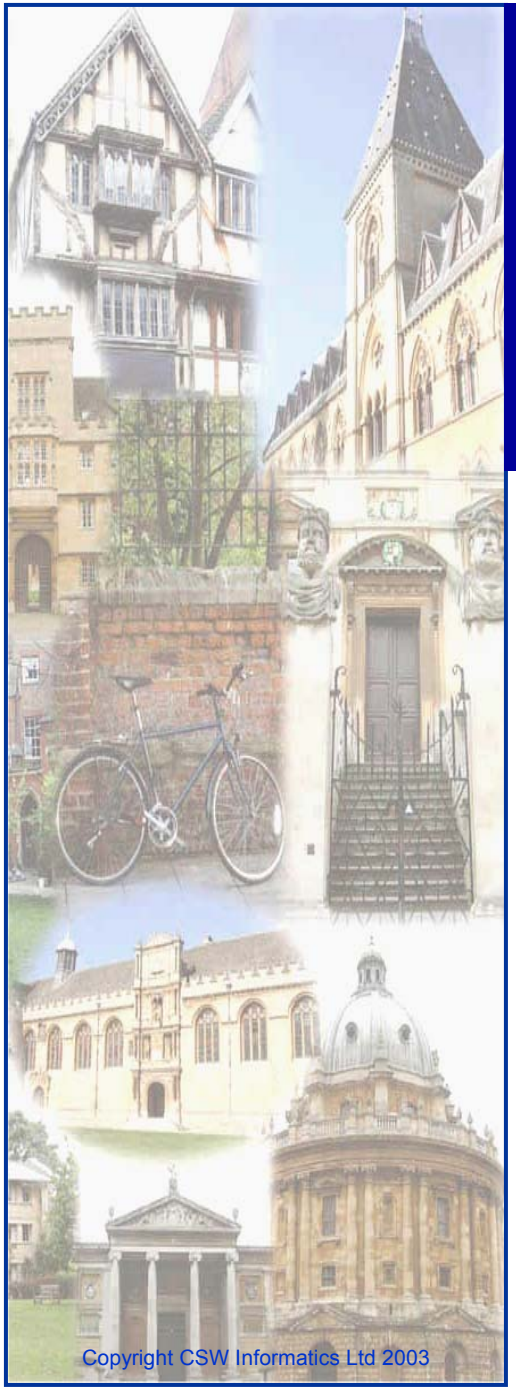
- **Object classes:**
  Object_Class_Term. “Details”
- **Properties:**
  Object_Class_Term. [*Qualifier_*]*Property_Term*. [*Qualifier_*] *Representation_Term*
- **CCTs:**
  CCT_Name. “Type”

Person. Details

Person. Name. Text
Person. Birth. Date
Person. Residence_Address. Address
Person. Official_Address. Address

A real excerpt from UBL's data dictionary

BIE Dictionary Entry Name	Occurrence	Definition
Address. Details	–	The particulars that identify and locate the place where someone lives or is situated, or where an organisation is situated.
Address. Identification. Identifier	1..1	A unique identifier given to a specific address within a scheme of registered addresses.
Address. Postbox. Text	0..1	A post office box number or a numbered post box in a post office assigned to a person or organisation where letters for them are kept until called for, used as part of an address.
…	…	…



SUMMER SCHOOL XML csw 2003

The UBL Modeling Methodology

The modeling process, in brief

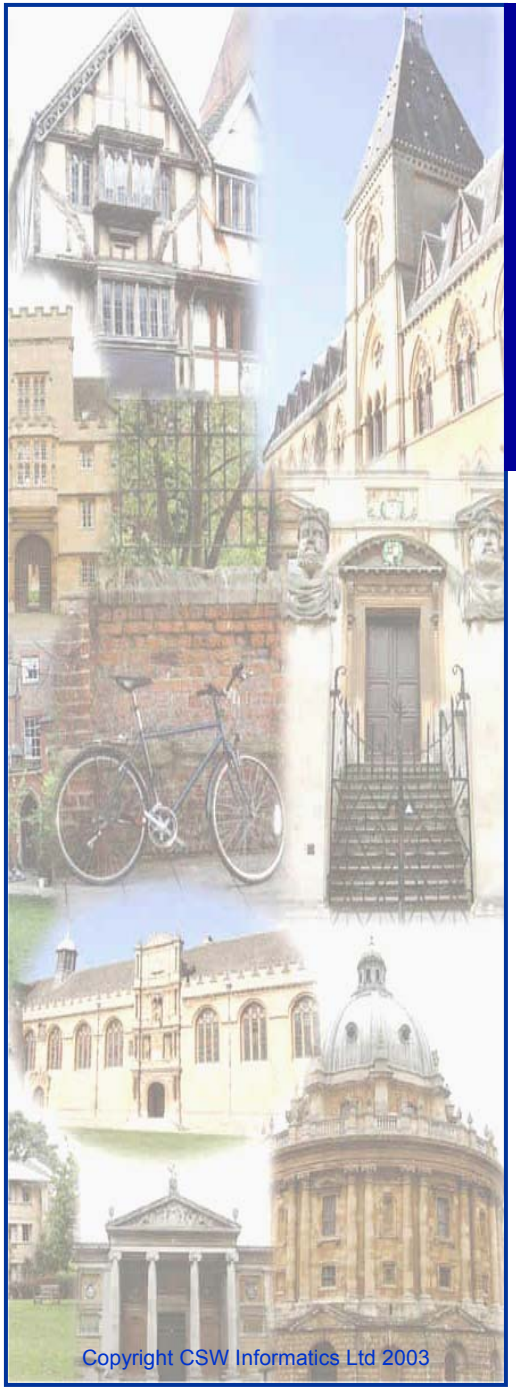
1. Identify content components
   - At three levels: atomic, aggregate, and document
   - Using xCBL V3.0 to prime the pump
2. Identify functional dependencies and normalize the model of each component
3. Choose a single hierarchical “view” from among the possible data relationships
4. Identify the relevant business context
5. Define the whole in terms of a “scope” (business process scenario)

More about normalization

- "If the value of one component changes when another component's value changes, then the former is said to be **functionally dependent** on the latter"
- "**Normalization** yields models that describe the network of associations between logical groups of components in optimal ways that minimise redundancy and prevent inadvertent errors or information loss when components are added or deleted"
  - Many XML information modelers do this intuitively, if not rigorously
  - XML nesting and repeatability pose challenges here

Looking at UBL's syntax-neutral model

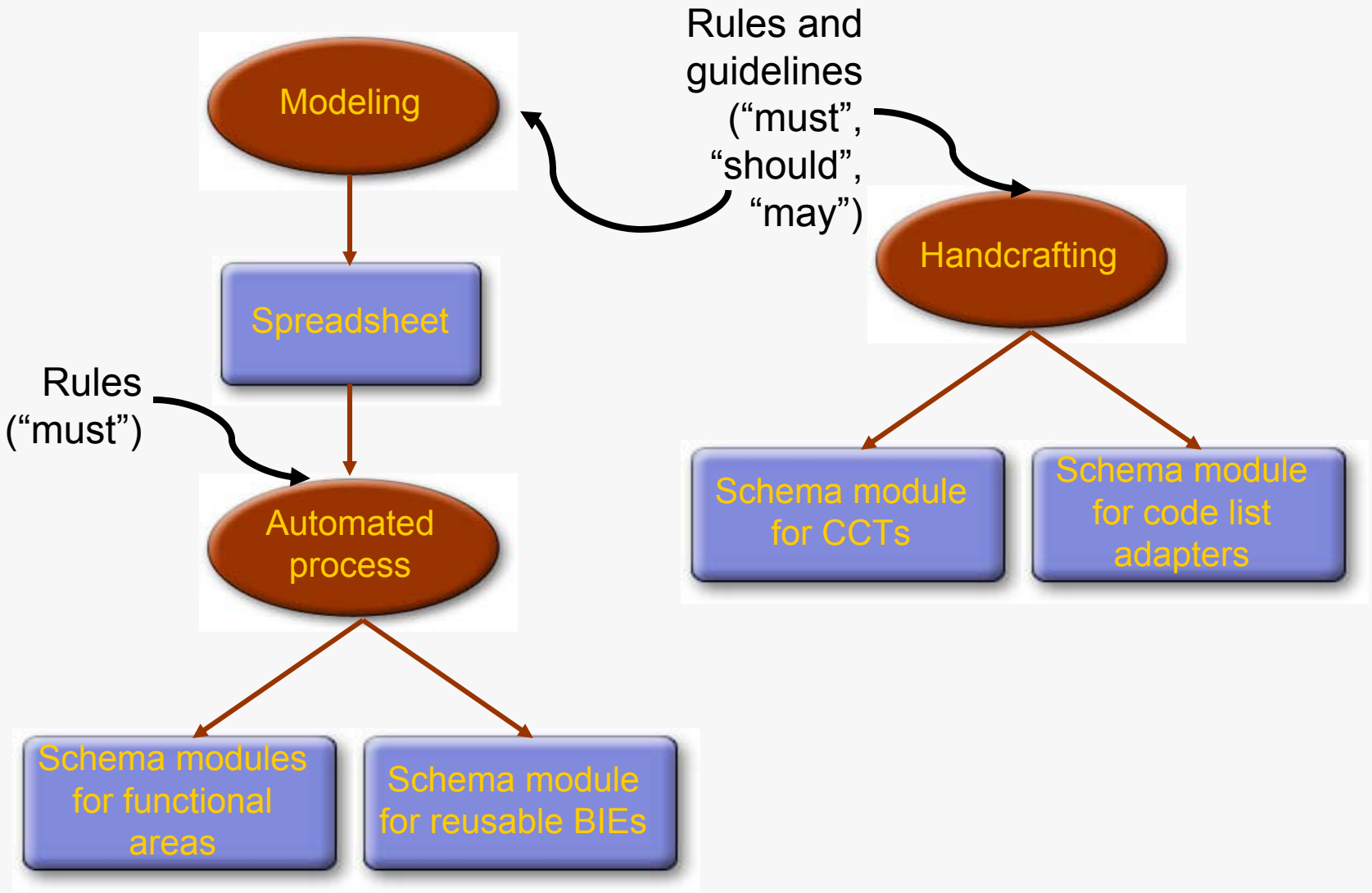
- The data dictionary in spreadsheet form
- The generated UML class diagrams
- The generated ASN.1 schemas
- The syntax-specific XML Schema versions? Patience…



SUMMER SCHOOL XML CSW 2003

Designing the UBL Schemas

The role of design rules in UBL schema creation



For any one model, the schema options are infinite

- The schema representation can vary along many dimensions – for example:
  - Elements and types in separate hierarchies
  - Rich simple types
  - Type inheritance and specialization in the style of OO
  - Independent local scoping of elements, attributes, and types
  - Namespace support for better federation of component creation and reuse
- The instance might look identical in all cases

Some of the major constraints on our rules

- Leverage XML technology, but make choices that keep it interoperable
- Support customization and reuse
  - Allow customizers to use the same rules that govern the UBL Library itself
- Selectively allow “outsourcing” to foreign schemas
- Make the names of XML constructs readable and natural
- Ensure that most of the rules are deterministic

Are the UBL rules applicable to your XML projects?

It all depends…

- Do you share our design principles and constraints?
- Do you share our business object metamodel (or something close to it)?
- Do you have the same profile of XML vs. application-specific tool usage?
- At the very least, you might pick up some interesting ideas
  - Many industry groups are going through this same exercise
  - We've communicated with several of them

A sampling of some draft UBL rules

Rule #	Rule Text
[R1]	All UBL schema design rules MUST be based on the W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[R4]	Names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
[R9]	Upper-camel-case (UCC) MUST be used for naming elements and types.
[R13]	For every object class identified in the syntax-neutral model, a complex type definition and a corresponding global element declaration bound to that type MUST be created.
[R50]	Minor versioning MUST be limited to declaring new optional constructs, extending existing constructs and refinements of an optional nature.

Categories of rules

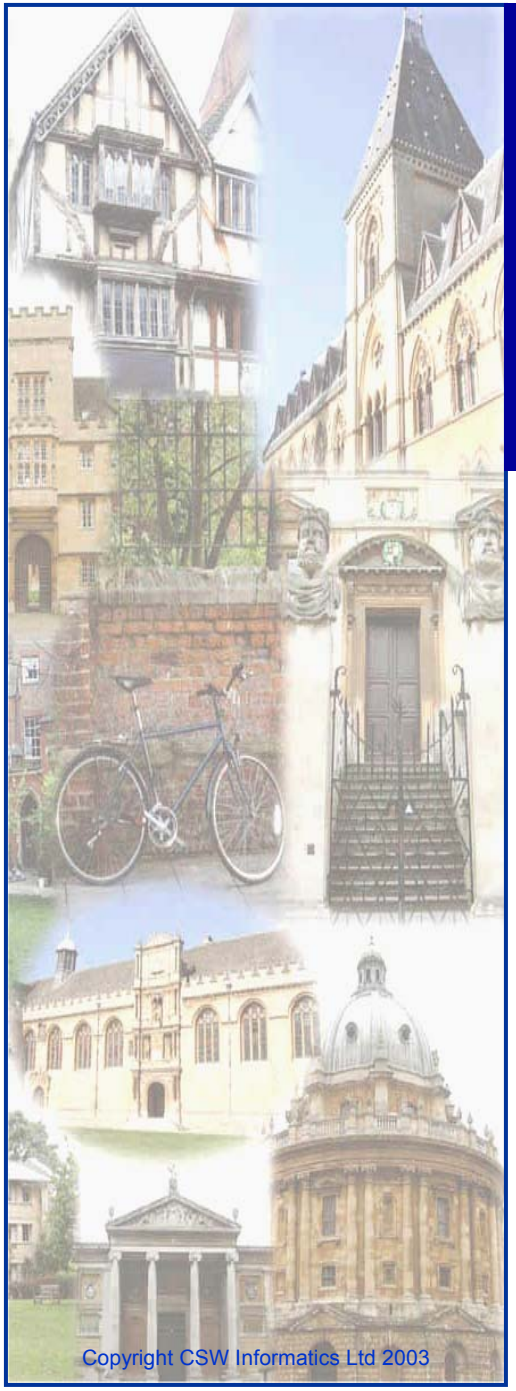
- Overall selection of standards to adhere to
- Constraining names assigned during modeling for I18N and readability reasons
- Constraining the modeling process so that the results are amenable to schema conversion
- Populating schema documentation fields

★ Modularity, namespaces, and versioning of schemas

★ Generating and naming elements, attributes, types, and other constructs derived from the model

★ Handling code lists

★ Enabling the context methodology



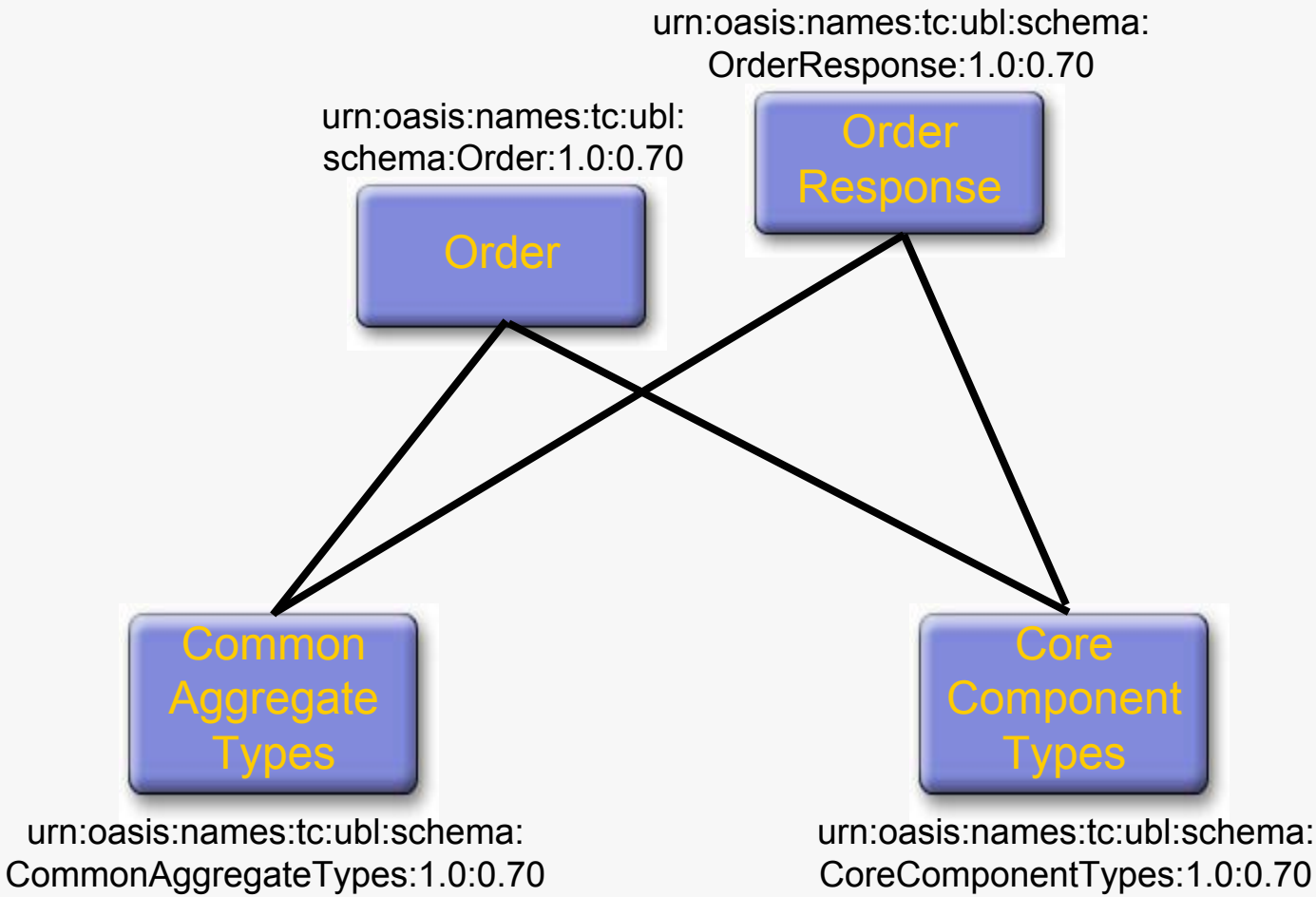
SUMMER SCHOOL XML csw 2003

Modularity, Namespaces, and Versioning

UBL schema packaging concepts

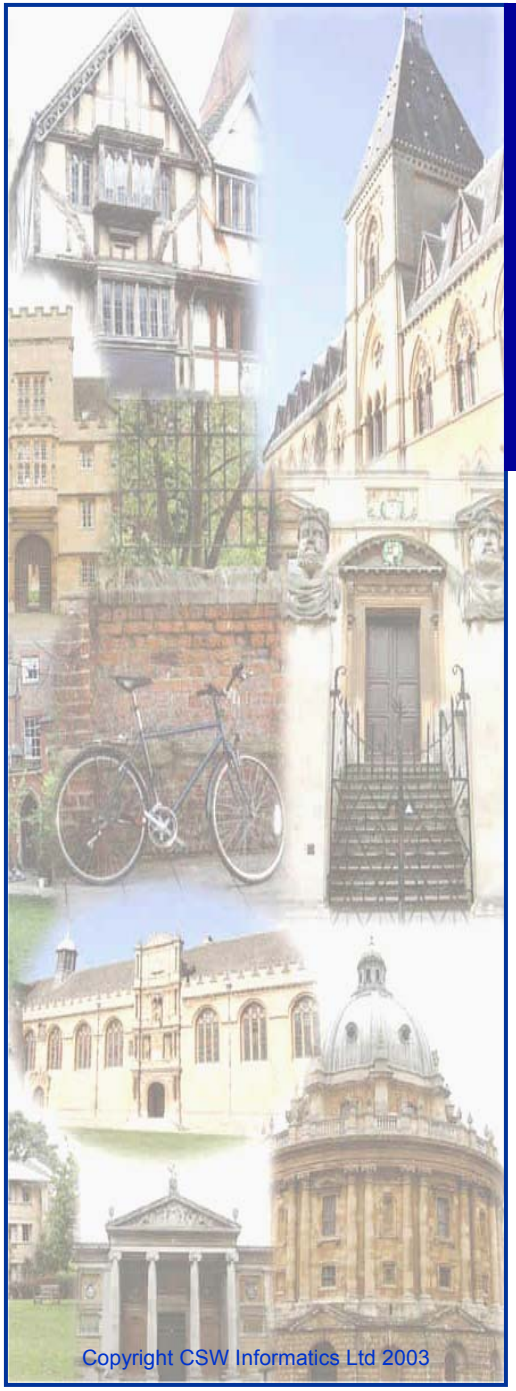
- **Modnamver:** modularity, namespaces, and versioning, of course
- **Schema module:** schema document
- **Root schema module:** declares a target namespace and is likely to include or import other modules
- **Internal schema module:** does not declare a target namespace and is never imported, only included

Examples of UBL Library packaging



Some additional modnamver rules

- Minor versions must remain backwards compatible
  - And can't break software conforming to prior versions through semantic changes
- All new versions, both major and minor, receive unique namespaces
  - All changes are thus persistent and uniquely addressable



SUMMER SCHOOL XML csw 2003

Schema Componentry

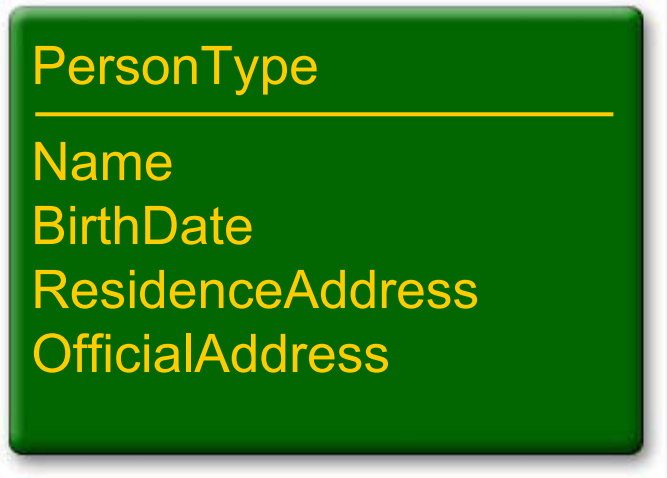
Mapping BIEs to XML constructs

- Object classes (such as *Person. Details*) become complex types
- Properties (such as *Person. Name. Text* etc.) become elements in those types' content models
- Representation terms (such as *Text*, *Date*, and *Address* – or *Address. Details*, actually) become the types bound to the property elements

Mapping BIE names to XML names

- Remove redundant and nearly redundant words in the property field (as in *. *Identification. Identifier*)
- Remove periods, spaces, and underscores
- Replace “Details” with “Type”
- When the representation term is “Text”, remove it
- When the representation term is “Identifier”, truncate it to “ID”
- Remove the object class name on properties, as the XML parent labels it sufficiently

The spreadsheet does this with some wild formulas!

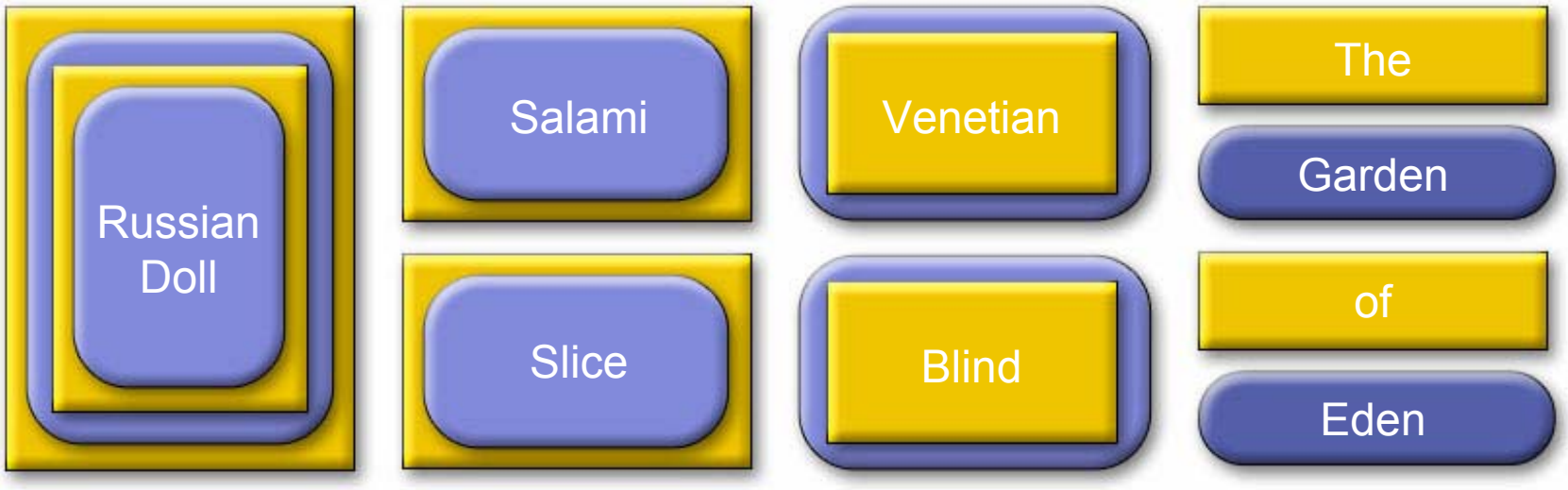


This doesn't tell the whole story

- Within a complex type, should the elements be local (declared *in situ*) or global (references to separate declarations) or some combination?
- How does this issue interact with namespaces?
- On what criteria should these decisions be based?

The four most obvious options

- The yellow squares are elements
- The blue rounded squares are types
- Roger Costello of *xfront.com* invented the first three
- There are many variations we won't go into here



Russian Doll

```
<xs:schema … >
   <xs:element name="Person">
     <xs:complexType>  keep nesting ever more deeply…
       <xs:sequence>
         <xs:element name="Name" type="NameType"/>
         <xs:element name="BirthDate" type="DateType"/>
         <xs:element name="ResidenceAddress">
           <xs:complexType>
             <xs:element name="Street" type="TextType"/>
             …
           </xs:complexType>
         </xs:element>
         <xs:element name="OfficialAddress">
           <xs:complexType> … </xs:complexType>
         </xs:element>
       </xs:sequence>
     </xs:complexType>
   </xs:element>
</xs:schema>
```

Salami Slice

```
<xs:schema … >
  <xs:element name="Person"> only elements are at the top level…
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Name"/>
        <xs:element ref="BirthDate"/>
        <xs:element ref="ResidenceAddress"/>
        <xs:element ref="OfficialAddress"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Name" type="TextType"/>
  <xs:element name="BirthDate" type="DateType"/>
  <xs:element name="ResidenceAddress">
    <xs:complexType> … </xs:complexType>
  </xs:element>
</xs:schema>
```

Venetian Blind

```
<xs:schema … >
  <xs:element name="Person" type="PersonType">
  <xs:complexType name="PersonType">
    <xs:sequence>
      <xs:element name="Name" type="NameType"/>
      <xs:element name="BirthDate" type="DateType"/>
      <xs:element name="ResidenceAddress"
type="AddressType"/>
      <xs:element name="OfficialAddress"
type="AddressType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AddressType">
    <xs:sequence>
      <xs:element name="Street" type="TextType"/>
      <xs:element name="PostCode" type="TextType"/>
      <xs:element name="Town" type="TextType"/>
      <xs:element name="CountryID" type="…"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

mostly types are at the top level…

The Garden of Eden

```
<xs:schema
   targetNamespace="http://www.example.com/BIEs"
   … > everything's at the top level…
   <xs:element name="Person" type="PersonType">

   <xs:complexType name="PersonType">
     <xs:sequence>
       <xs:element ref="Name"/>
       <xs:element ref="BirthDate"/>
       <xs:element ref="ResidenceAddress"/>
       <xs:element ref="OfficialAddress"/>
     </xs:sequence>
   </xs:complexType>

   <xs:element name="Name" type="TextType"/>

   <xs:complexType name="TextType"> … </xs:complexType>

   …
</xs:schema>
```

Some potential criteria for choosing

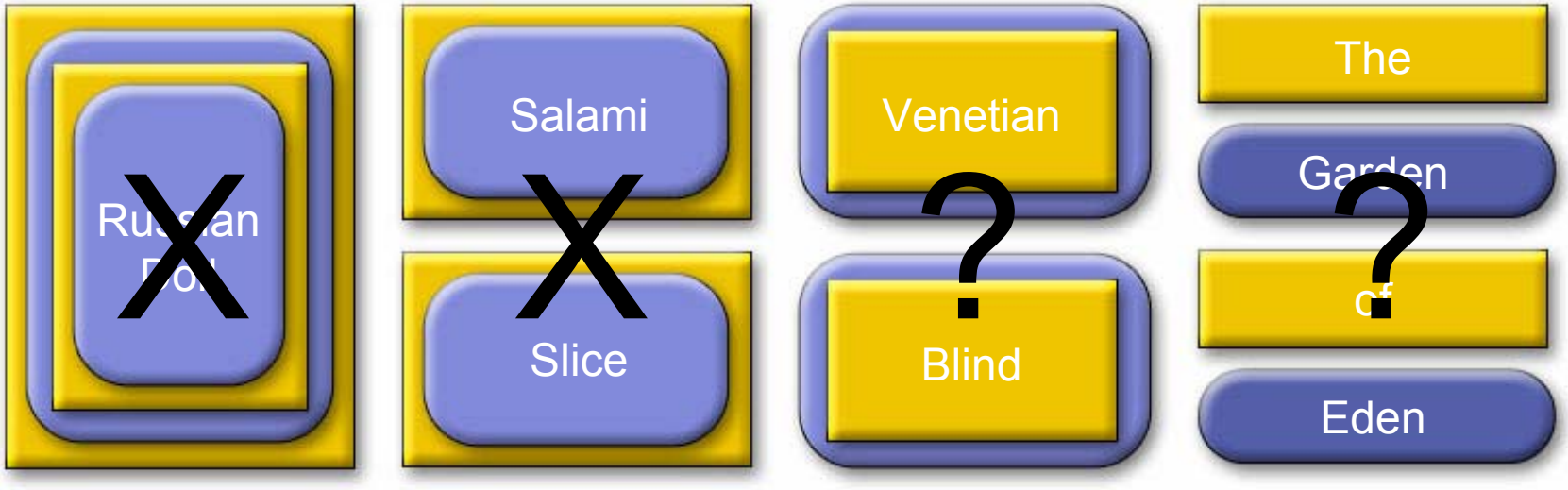
- Flexibility
  - Does the vocabulary need to adapt, chameleon-like, to different namespaces?
- Consistency
  - Is it acceptable for the markup to bounce between qualified and unqualified? between different namespaces?
  - What happens when importing schemas do overrides?
- Reuse
  - What constructs might someone want to reuse wholesale?
- Specialization
  - What constructs might someone want to modify?

UBL's criteria

- The UBL Library is explicitly intended for reuse and specialization
- We have two use cases:
  - **Tweaking** document structures for a new business context
  - **Creating** whole new document types out of existing piece-parts
- The challenge: make the right set of schema components reusable to meet both use cases, while adhering to all our other principles

It's easy for UBL to choose global types

- To support our "tweaking" use case and our requirement for leveraging XML tools, we need to allow for XSD type specialization
- To extend or restrict a type, you must be able to reference it; hence, named top-level types



Benefits of global elements

- A global, namespaced element can potentially be referenced for many purposes:
  - Root element
  - Head element of substitution group
  - Component of wildcard content
  - Component of new foreign content models (directly applying to our “creating” use case)

Costs of global elements

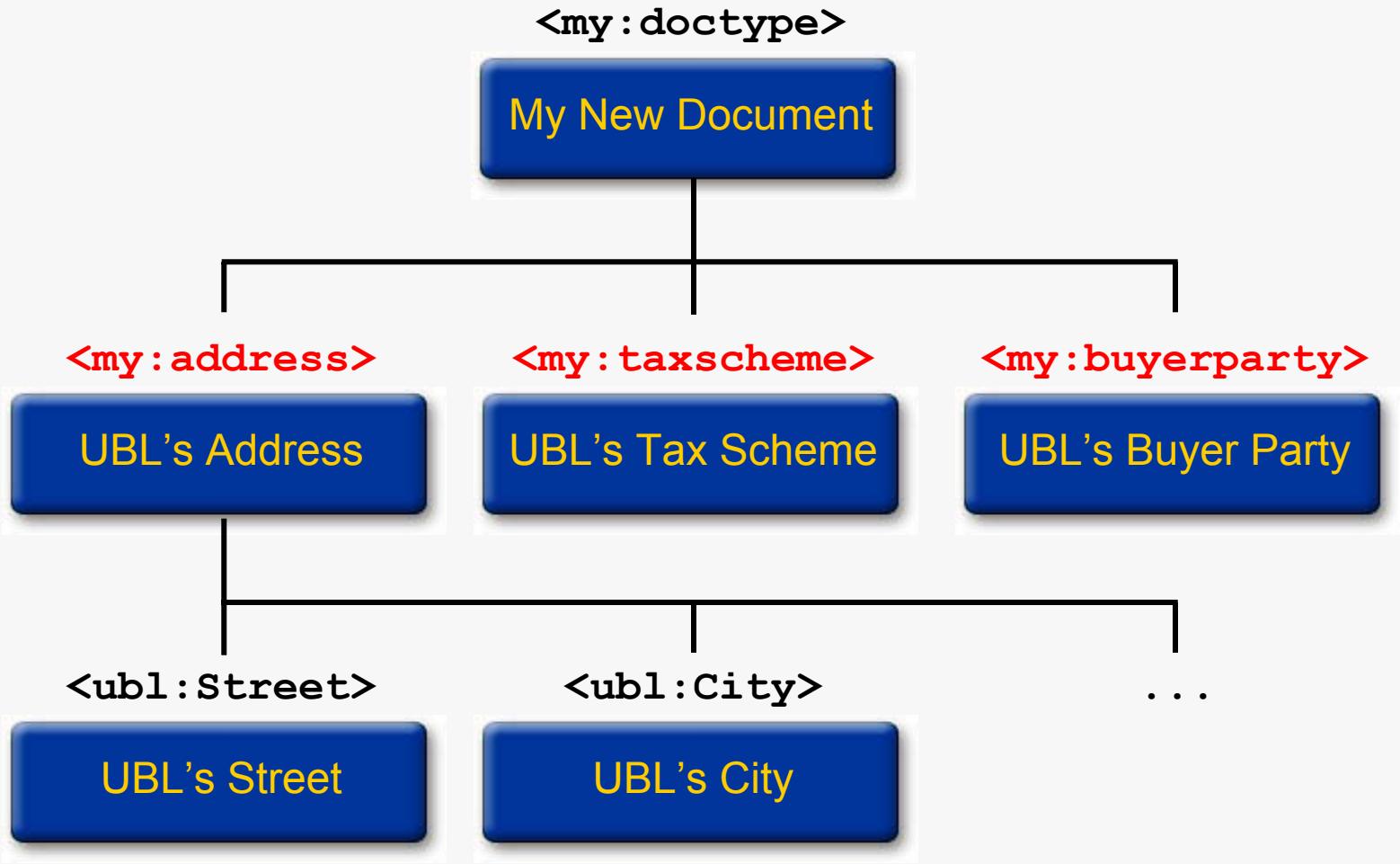
- Every element in a namespace must have a completely unique name
  - Every variation in content must result in a new name
  - This can mean a *lot* of elements
- Generated representations must expand to account for the public interface that the element is projecting
  - Such as UML and JAXB-produced Java code

Benefits of local elements

- A local element is scoped just to the type that defined it
  - Mapping neatly to properties of OO classes
- Multiple local elements can have the same name while having different “guts”
  - Useful for controlling element explosions

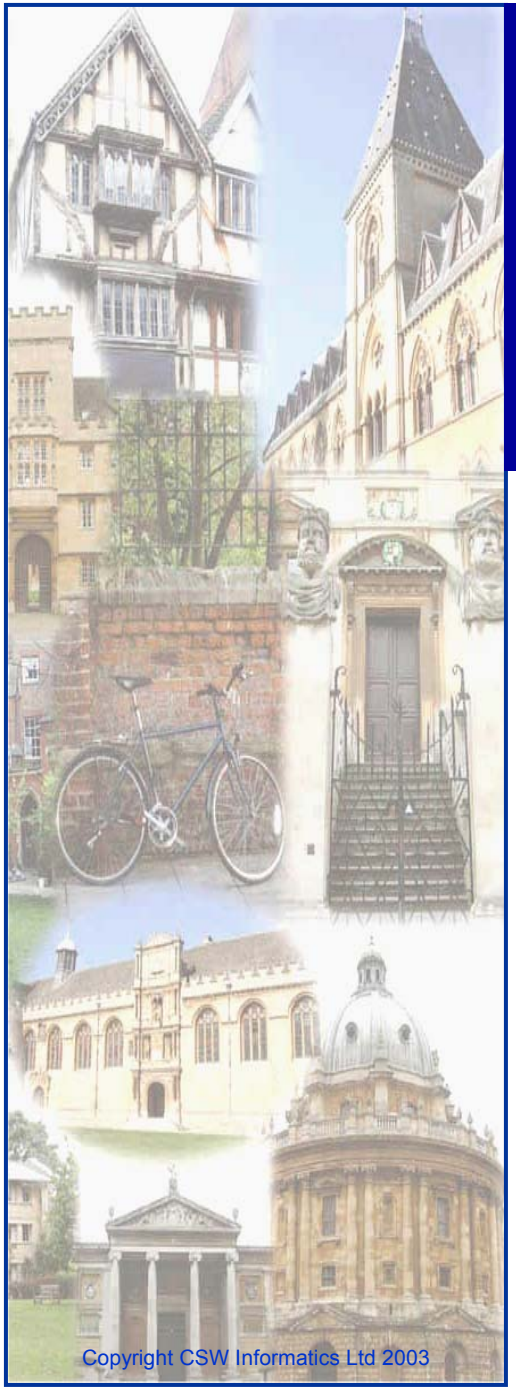
Costs of local elements

- You can only reuse types, not elements – breaking non-type-aware code such as V1.0 XPaths



UBL's choice

- Call it…the Garden of Venice?
- Every object class turns into a complex type, *and* a corresponding generic global element for use by customizers in creating new document types
  - For example, both AddressType and **`<ubl:Address>`**
- Within complex types, element declarations use **`ref=`** instead of **`name=`**
  - With one exception: when the representation term is **. *. Identifier*, make the element local
  - A compromise to account for the syntactic divergence/semantic convergence of the many ID elements

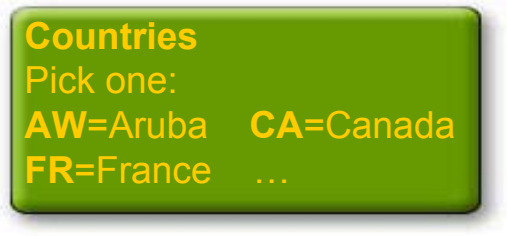


SUMMER SCHOOL XML csw 2003

Code Lists

The huge need for codes

- A code is a character string that represents a definitive value
- Code lists are valuable as unambiguous taxonomies
- In many cases, such as product classifications, code lists are big business
  - Some code list owners charge for their use



Options for formally representing code lists

- Often they are merely maintained in text documents
- But formal encodings are extremely useful, for example:
  - RDF ontologies
  - The ebXML Registry Information Model's `<ClassificationScheme>` markup
  - XSD (such as enumerated simple types)
- You could develop different representations for different purposes

The attractions of code lists in XSD form

- Schema validation can do code checking “for free”
- This step usually occurs early in the processing pipeline
- This encoding benefits from tool availability
  - And could even be generated from a more-primary XML representation
- These all support UBL’s “leverage XML technology” goal

The downsides

- Many code lists are too large (~10K codes) or dynamic (~daily) to take advantage of XSD
  - But one study showed more than one-third of legacy code lists to be variants of Yes/No!
- Validation through schemas will never be complete for some applications
  - Such as codes that become dynamically invalid depending on previous code choices

Each user of a code list could reproduce it in a schema

- But re-coding a code list over and over in different schemas is costly and prone to error
- Better to help code list owners produce their own code list schema modules

UBL's solution: code list schema module rules

- A code list owner can choose to conform to the rules by producing a reusable schema module that defines a code list datatype
- The level of validation is entirely up to them
  - Enumeration
  - Regular expression
  - No constraints
- The "normative status" of the module is also up to them
- They just need to provide enough metadata to uniquely identify the meaning of each code
- We're working with a number of groups to help them do this

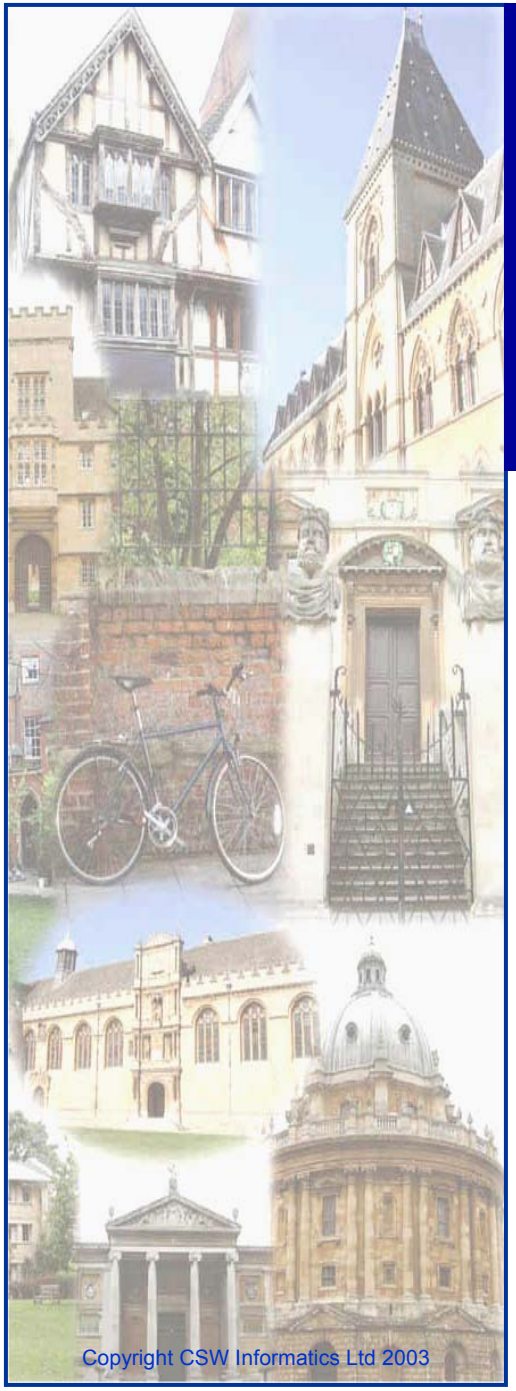
UBL and others can bind the type to their own elements

- UBL elements would be bound to a foreign type defined by a code list owner
  - This would be done in the “code list adapter module”
- The metadata attributes could be defaulted, or even fixed

```
<ubl:CountryID
   xsi:type="unece:ISO3166CountryCodeType"
   various metadata attributes...>
 FR
 </ubl:CountryID>
```

A global marketplace in XML-based code lists?

- If all goes well, we could see the following benefits:
  - Less duplication of work in XML vocabulary development
  - Wider application support for well-known code lists
  - Earlier validation of code values
  - Standardization of more code lists, and even formally described subsets and extensions
  - Greater “semantic clarity” through identifying standard code list metadata

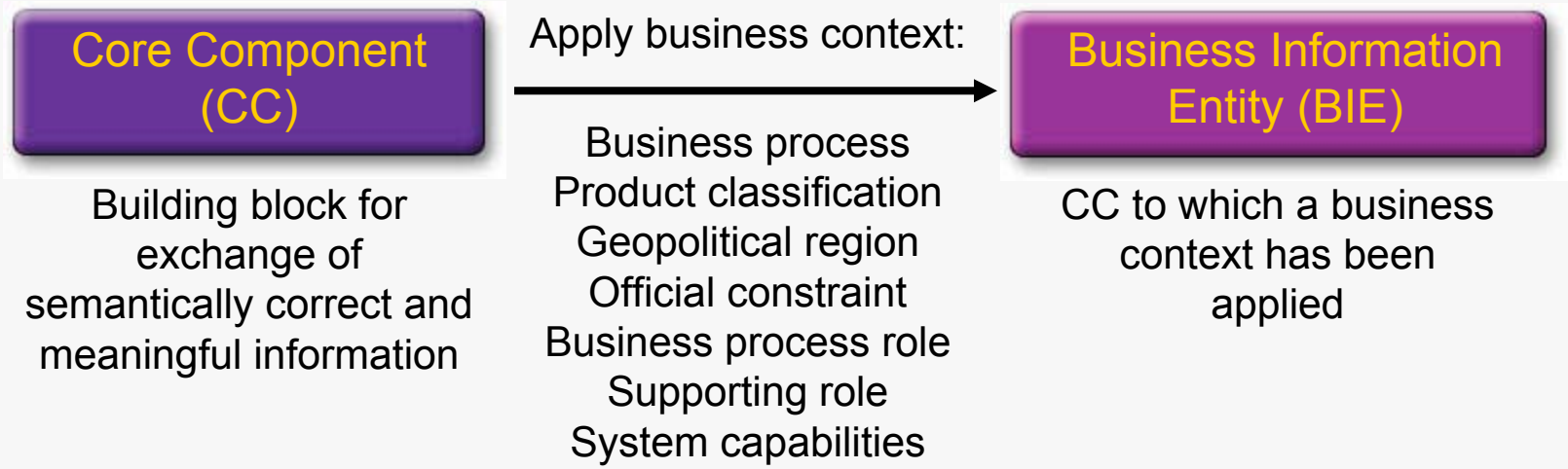


SUMMER SCHOOL XML csw 2003

Adding Business Context to Documents

Recall the UBL requirement for business context

- A lot of business factors can require changes to the “shape” of a business object



The customization community around UBL

- The UBL Library is intended to be an XML-based international “core”
  - Similar to UN/EDIFACT or X12
- Customization is expected
  - By national and industry groups
  - By smaller user communities
- These changes are driven by real-world requirements

The EDI precedent

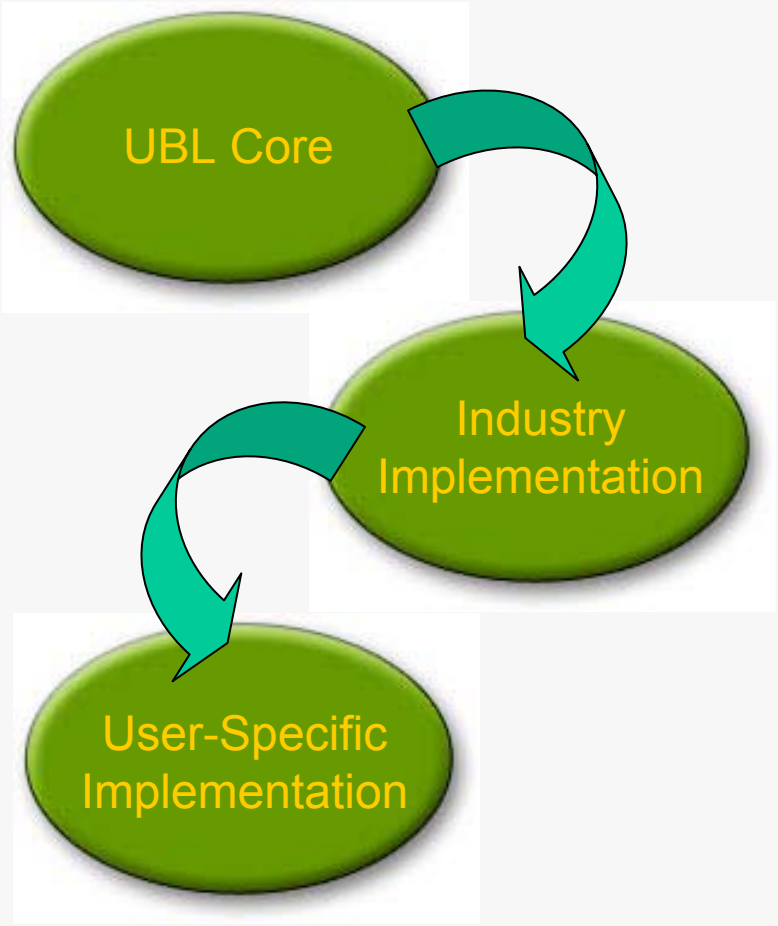
- EDI uses a prose-based subsetting approach
  - UN/EDIFACT ⇨ industry implementation guide ⇨ trading partner IG ⇨ departmental IG
- Some XML-based B2B vocabularies now use schema-based extension
  - Core vocabulary + extensions at each level

UBL leverages both approaches

- It picks an 80/20 point in supplying fields likely to be needed
- Then it allows both subsetting and extension to the limit of XSD's abilities
  - Again, leveraging existing XML software and standards
- UBL makes a key addition: the XSD derivations must be accompanied by a machine-readable description of the business context

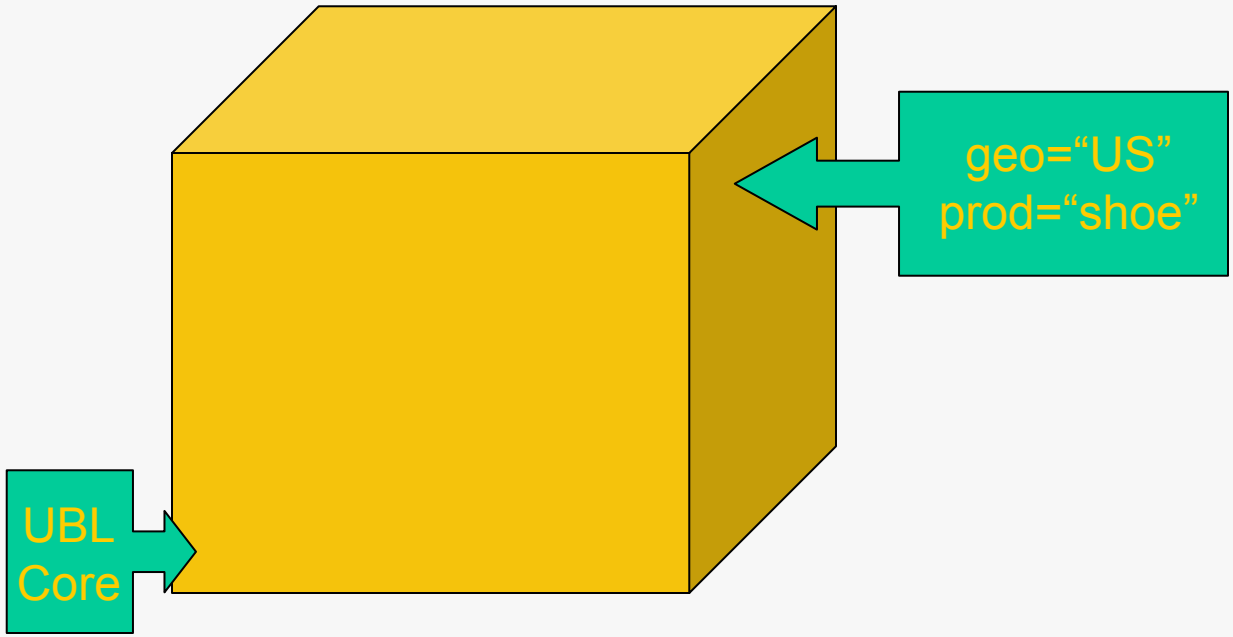
Successive sharing and customization

- The core standard is subsetted and extended further each time
- Each circle would have its own set of schemas/namespaces and corresponding business context metadata



The business context metadata

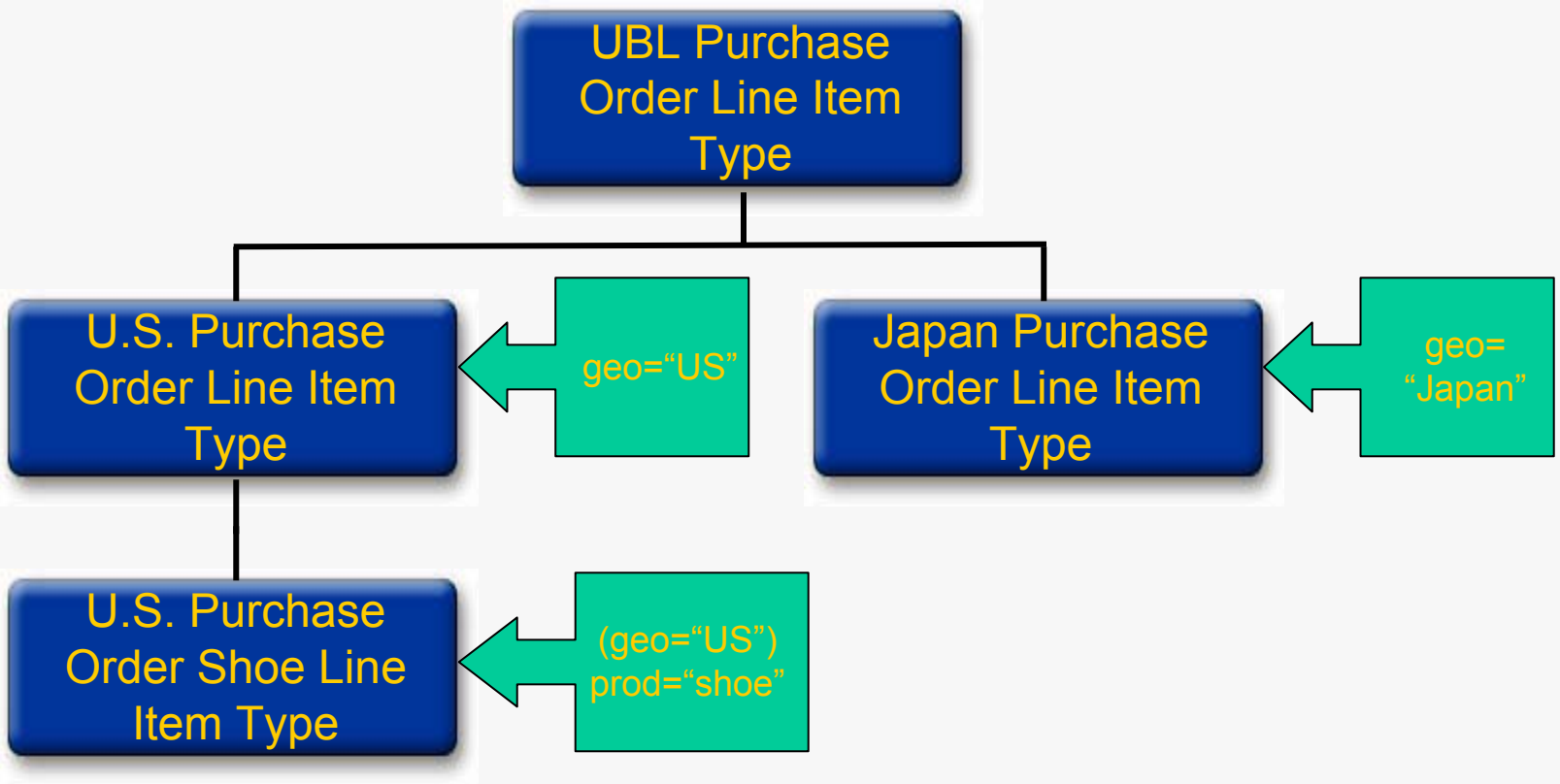
- UBL starts out identifying only the business process
- Supplying values for the eight context drivers gives you a unique business context



The draft contextualization process

- Customizers will need to do two things:
  - Handcraft an XSD derivation, adhering to XSD rules
  - Attach the business context, adhering to UBL rules
- One UBL rule: context drivers can be specialized, but not reset
  - US⇨Maine, not US⇨Japan
- Eventually, the goal is for the context methodology to be more automated
  - So that you can input the drivers to a registry and get a freshly generated schema

An example of how the context can be specialized



Sometimes the 80/20 point isn't sufficient

- Let's say a core UBL Address requires a street, city, country, etc.
  - (Though the cardinalities are currently looser than this)
- But for parts delivery to a mobile oil-drilling platform in international waters, the ship-to information for an order must be only GPS coordinates
- Ideally, software would be able to associate this kind of address with a UBL Address somehow
  - To reuse whatever parts of the processing still apply

When business requirements and technical abilities diverge

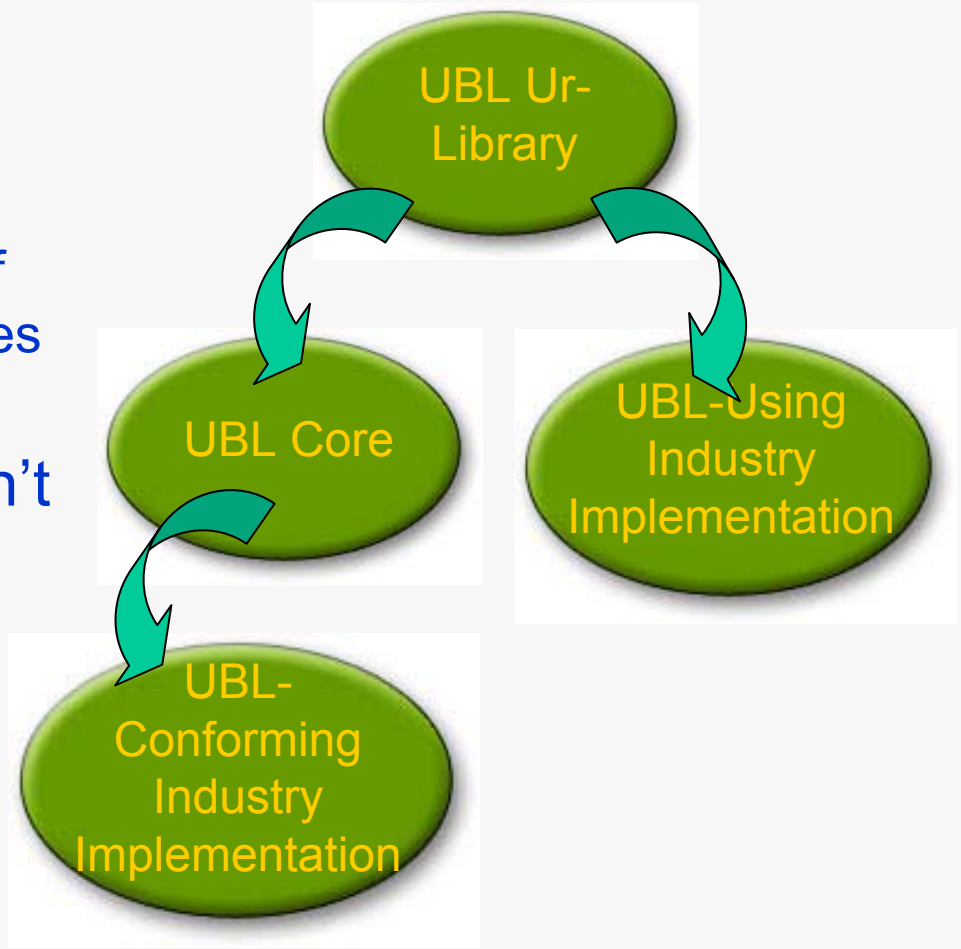
- Several actions are needed here:
  - Characterize this situation as a formally described business context
  - Add GPS coordinate data as required fields
  - *Remove* fields (city etc.) that are normally required
- Neither EDI subsetting nor XSD derivation allows this last one – even if combined

UBL proposes to increase interoperability even here

- One alternative is to build a whole new core
  - But this compromises the investment in the semantic substrate
- Another alternative is to build a “prior core” – an “Ur-Library” – on which to layer the UBL Library itself
  - Its base types would allow for optional fields where UBL doesn’t
  - The types would be abstract
  - UBL would become a restriction of these types

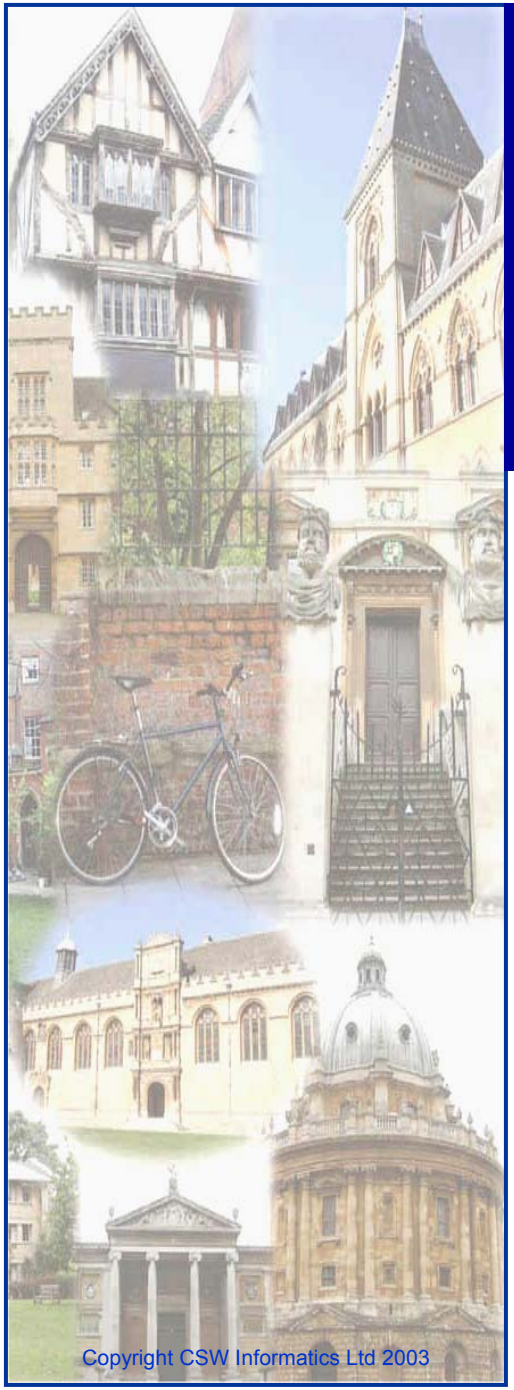
Customizing from the Ur-Library

- Customizers could derive from the Ur-Library if necessary
  - And get the benefits of using well-defined types underlying UBL
- However, they wouldn't be able to claim "UBL conformance"



Current status of the UBL context methodology

- Even the simpler “non-Ur” ideas are as yet not fully tested
- Work remains to be done on the code values for the business context drivers
- However, a good paper describing the work to date has been written
- At least UBL has no reliance on an application-specific mechanism that would require significant investment in extra tools
  - You can use existing tools to build derivations



SUMMER SCHOOL XML CSW 2003

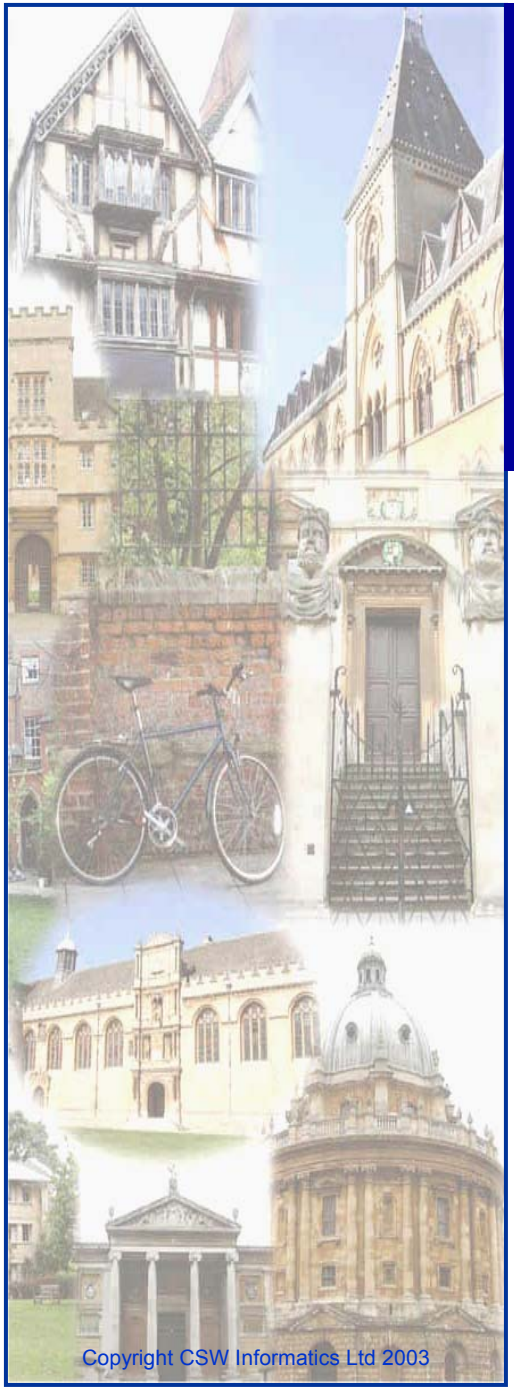
Resources

Learning more about UBL

- The public OASIS web page:
  www.oasis-open.org/committees/ubl
  - The subcommittees have their own portals, linked from here
  - White papers, presentations, and the latest draft release are available
  - You can also find instructions on how to submit comments
- The Cover Pages roundup on UBL:
  xml.coverpages.org/ubl.html
  - Pointers to articles, mailing lists, and so on
- ebXML information:
  - *ebxml.org*, *ebxmlforum.org*, *freebxml.org*

Make sure to review the UBL release cover letter

- It supplies additional business process scenarios and examples
  - Buying Office Supplies
  - Buying Joinery
- These include process diagrams and sample UBL trading documents



SUMMER SCHOOL XML CSW 2003

Conclusion

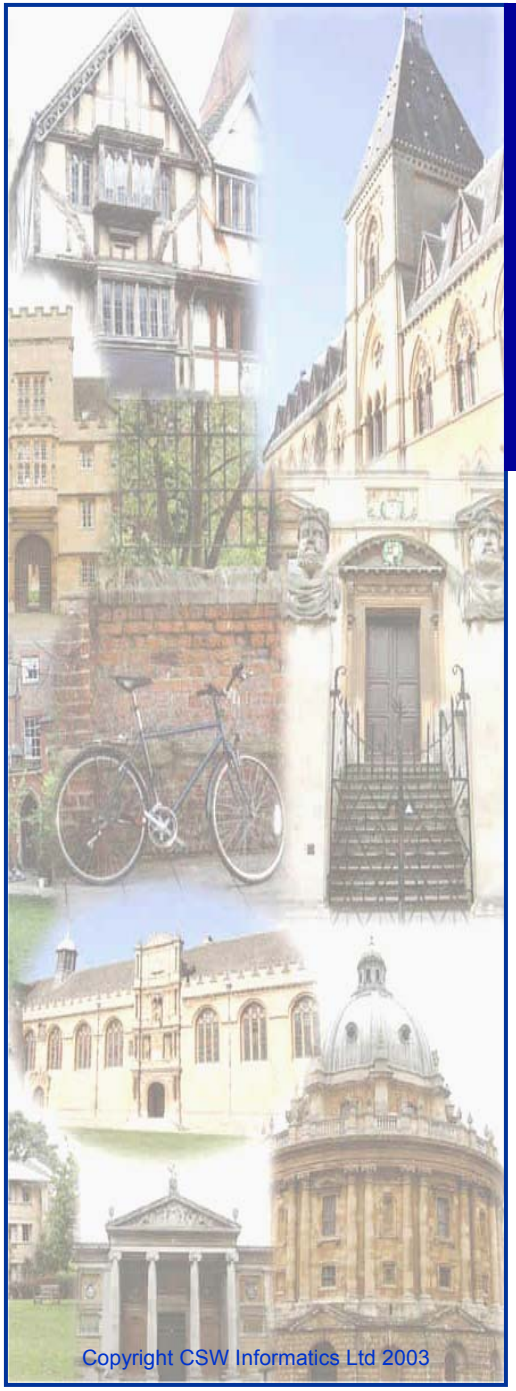
UBL offers important and interesting solutions

- As a B2B standard:
  - It is user-driven, with deep experience and partnership resources to call on
  - It is committed to truly global trade and interoperability
  - Its standards process is transparent
- As an XML application:
  - It is layered on existing successful standards
  - It is tackling difficult technical problems without losing sight of the human dimension

Ponder this

HTTP + HTML = Web Publishing.

ebXML + UBL = Web Commerce?



SUMMER SCHOOL XML csw 2003

Thank You!
Questions?

Eve Maler
eve.maler@sun.com