Data Model Perspectives for XML Schema

Felix Michel (ETH Zürich), Erik Wilde (UC Berkeley)

XTech Conference 2007, Paris, France

May 16, 2007

Outline

A Closer Look at XML Schema

- XML Schema allows to describe relationships between structures in an expressive, semantically relevant way
  - Type derivation, substitution groups, identity constraints
- XML Schema offers extension points, or "hooks"
  - `xs:appinfo` and non-Schema namespace attributes

XML Schema-Validation

- Processing of a document against an XML Schema is more than just "validation" on an accept/reject basis
- Schema-validation is twofold; it comprises two aspects:
  - Validity checking
  - Type annotation
- Assessment outcome and type annotations are added to the Infoset, resulting in the *Post Schema-Validation Infoset* (PSVI)

XML Schema and Instance Processing

- XSLT 1.0:
  - Works on XML documents
  - More specifically, on the *XPath data model*
- XSLT 2.0 and XQuery 1.0:
  - Works on XML documents, which are possibly *Schema-validated* by some XML Schema
  - More specifically, on the *XQuery 1.0 and XPath 2.0 Data Model* (XDM)

Utilizing XML Schema from Instances: Idea

- Use XML Schema information while processing instances!

Utilizing XML Schema from Instances: Benefits

- Use Schema information as metadata
  - Structural metadata such as type hierarchy or ID/IDREF relations
  - Other metadata such as RDF annotations or embedded rules
- Perform type introspection and reflection
  - Write more resilient applications
- Emphasize the second aspect of Schema-validation, which is *annotation*
  - Move some validation tasks from processors to application logic $\rightarrow$ more versatile and fine-grained validation becomes possible

Utilizing XML Schema from Instances: Problems

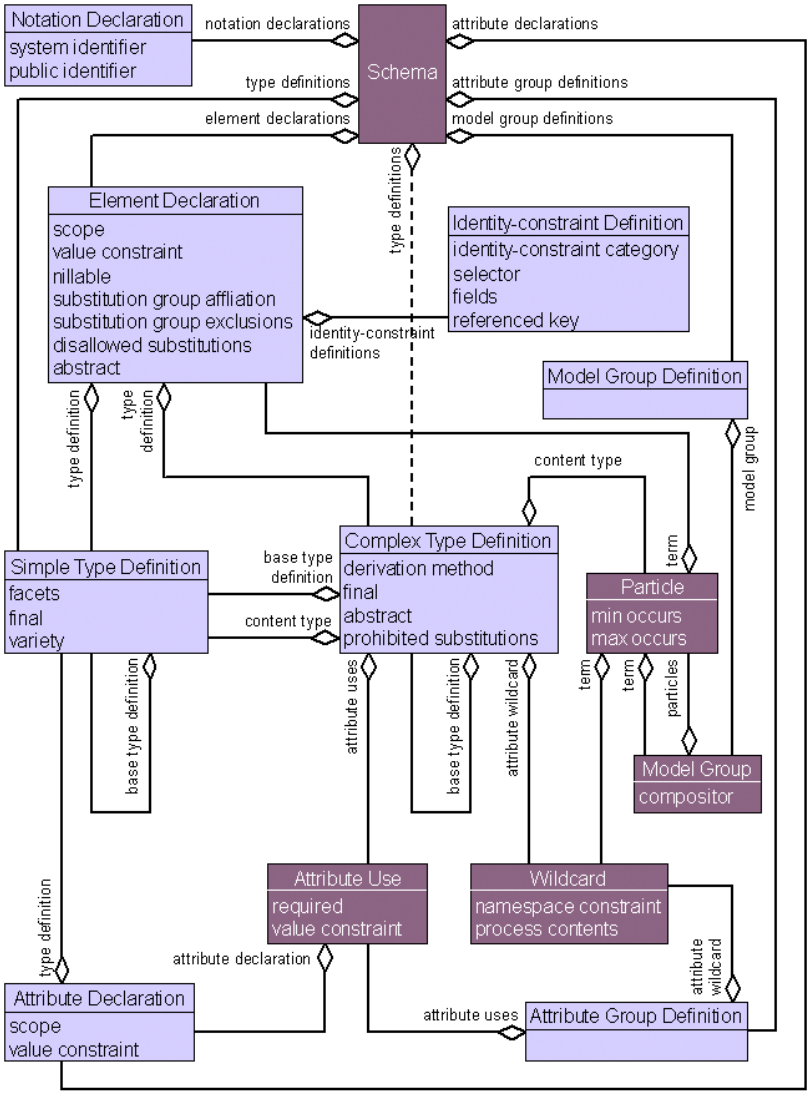
- Which parts of XML Schema information are accessible?
  - From the PSVI, it is impossible to determine where wildcards can occur
- How is this information exposed?
  - XPath 2.0 represents XML Schema type definitions through QNames
- Which data models are available? Which one is the best?

Data Models for XML Schema

- At present, different data models for XML Schema are available
- Each one has its own scope, specific goals, and some limitations
- There is nothing like "XDM for Schema"

The Abstract Data Model

- The recommendation defines XML Schema in terms of an *abstract data model*
- The abstract data model is comprised of *Schema components*



XML Schema Data Models and Interfaces

- The normative XML syntax for XML Schema
- XML Schema APIs
  - Microsoft's XML Schema Object Model (SOM)
  - Eclipse's XML Schema Infoset Model (XSD)
  - Java.net's XML Schema Object Model (XSOM)
- The (abandoned) *Formal Description* of XML Schema
- *XQuery 1.0 and XPath 2.0 Formal Semantics* also (re-)models essential parts of XML Schema's data model (e.g., the type system)

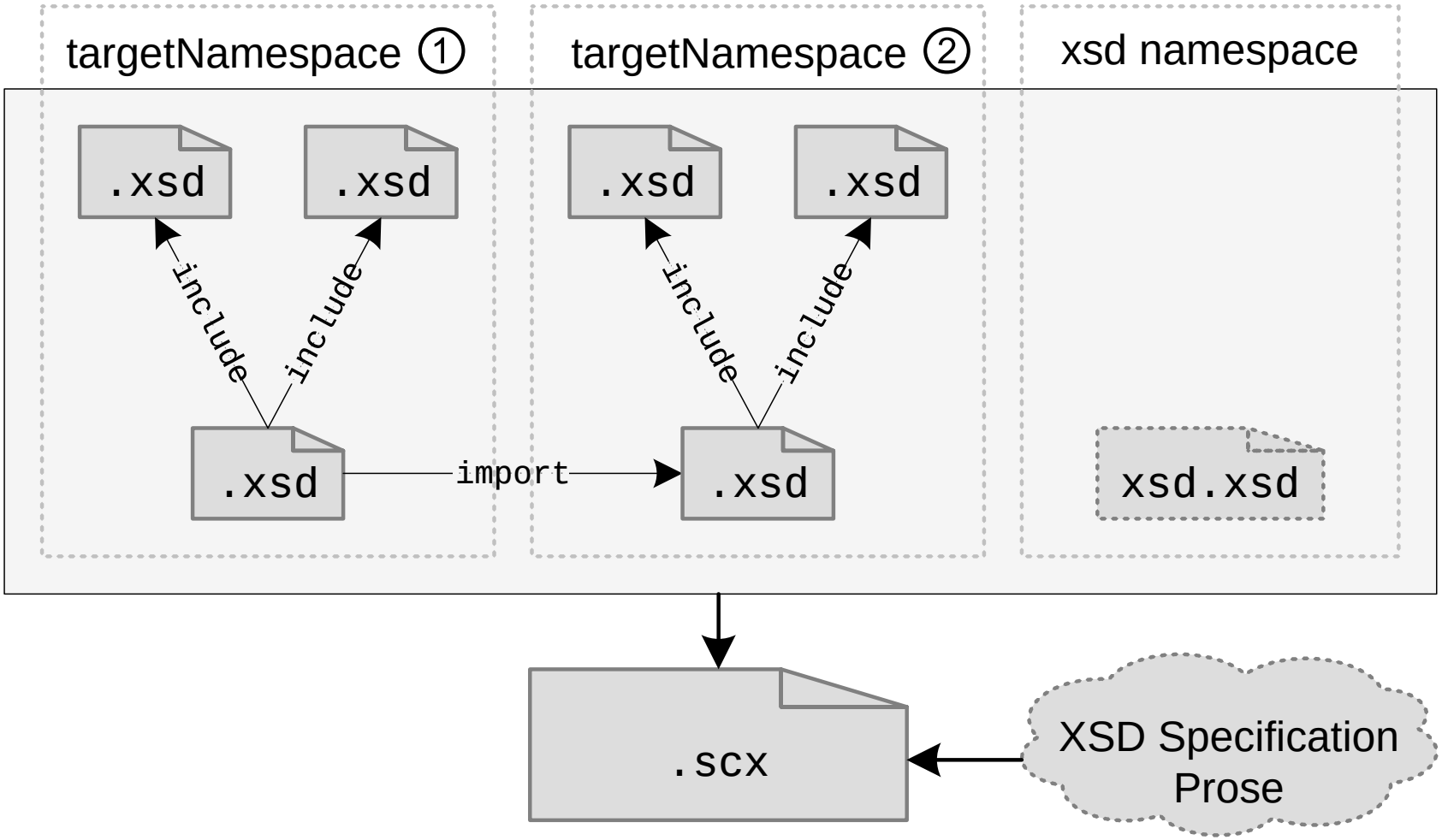
Data Model Accessibility: Requirements

- A *unified*, *accessible data model* would be very helpful
- We require such a data model to include:
  - A unified data model with concise rules for canonicalization
  - A clear and explicit concept of component identity
  - A set of accessor functions
- Ideally, the data model is accessible from within standard technologies such as XPath 2.0

SCX: Schema Components XML Syntax

- Self-contained XML representation of an assembled Schema
- Supported by an XSLT 2.0 function library, SCX exposes the full Schema information...
  - ...in an accessible way: `$elem/scf:get-type(.)`
  - ...in a navigable way: `$type/scf:base-type-definition(.)`
  - ...solely using standard technologies
- Transformation implies a certain amount of canonicalization

SCX: Self-Contained Representation



Scenario-Specific Data Models

- Recall the initial application scenario:
  - Utilizing XML Schema-information while processing instances
- Present data models do not provide dedicated support for this particular scenario
  - Can we do better?
- What are the questions which are likely to arise in such a scenario?
  - Which elements do I have to expect in a given context?

Utilize Regular Language Theory

- Make use of interesting parts of regular language theory:

Marked Expressions: Regular expressions, where occurrences of all symbols are made distinct by subscripts

Follow Sets: A concept related to Brzozowski derivatives of regular expressions

Follow Sets

- Introduced by Berry and Sethi in 1986
- *Follow sets* contain all positions that can legally follow after a given position in a string
  - $follow(E, a) = \{b \mid uabv \in L(E)\}$
- A *first set* contains all positions with which a string can be legally started
  - $first(E) = \{a \mid av \in L(E)\}$

Follow Sets

- Introduced by Berry and Sethi in 1986
- *Follow sets* contain all positions that can legally follow after a given position in a string
  - $follow(E, a) = \{b \mid uabv \in L(E)\}$
- A *first set* contains all positions with which a string can be legally started
  - $first(E) = \{a \mid av \in L(E)\}$
- Applications:
  - Efficient construction of the related finite automaton
  - Definition of *competing* symbols
  - Checking of content models in XML grammars

XML Schema Content Models

- The abstract data model uses *particles* and *model groups* for defining content models
  - Model groups have operators in $\{$*all*, *sequence*, *choice*$\}$
  - A model group contains *particles*
  - Particles have occurrence indicators *min occurs* and *max occurs*
  - A particle can reference a *wildcard*, an *element declaration*, or a *model group*
  - Thus, model groups can be nested
- This is convenient for defining content models, but not necessarily for exploring them

Occurrence-Based Alternative

- *Occurrence* components have the following properties:

  optional: a boolean that indicates whether the occurrence may be omitted

  unbounded: a boolean that indicates whether arbitrary repetitions of the occurrence are permitted

  follow: an (unordered) set of occurrences that legally can follow after the occurrence

  terminal: a reference to either an element declaration or a wildcard

- *Numeric exponents* of particles (`minOccurs`, `maxOccurs`) and particles within *all* model groups are expanded

Conventional vs. Occurrence-Based: An Example

```
<xs:sequence>
  <xs:element ref="ex:a" minOccurs="1" maxOccurs="2"/>
  <xs:choice>
    <xs:element ref="ex:b" maxOccurs="unbounded"/>
    <xs:element ref="ex:c" minOccurs="0"/>
  </xs:choice>
</xs:sequence>
```

occurrence	a_1	a_2	b_1	c_1
optional	false	true	false	true
unbounded	false	false	true	false
follow	a_2, b_1, c_1, !	b_1, c_1, !	b_1, !	!
terminal	ex:a	ex:a	ex:b	ex:c

Equivalence of Expressiveness

- Using *occurrences*, XML Schema content models can be expressed without loss of information
  - Conventional content models correspond to regular expressions, whereas occurrences and their follow sets relate to the corresponding automaton's states and transitions, respectively
- However, bijective mappings are not ensured in general, unless further precautions are taken
  - But this is a question of the syntax, not an inherent problem

Prototype Implementation

- Our prototype implementation of SCX additionally supports the occurrence-based perspective
- The prototype is written in plain XSLT 2.0
- In the current prototype, the expansion of *all* model groups has factorial complexity in memory size

Occurrence Look-Ahead

- Assume a variable `$elem` which points to a certain node
- Retrieve the corresponding occurrence...

```
<xsl:variable name="oc" as="element(occ:occurrence)"
    select="$elem/occ:get-occurrence(.)"/>
```

- ...and return the set of potential next siblings

```
<xsl:sequence select="$oc/occ:follow(.)"/>
```

- Written as one, single path expression:

```
$elem/occ:get-occurrence(.)/occ:follow(.)
```

Versatile Validation

- Content model checking easily becomes possible as well:

```
<xsl:variable name="validation-result"
    as="element(val:result)"
    select="occ:validate($elem)"/>
```

- The function above `occ:validate` returns an XML structure which includes detailed information about...
  - ...validation errors, if any
  - ...possible legal elements
  - ...the precise validation context

Possibilities and Opportunities

- Applications with more resilient behavior
- Better compatibility and interoperability in the context of versioning and extensibility
  - Implementation of Bau's *Validation-by-projection* or Thompson's *V2S*
- We expect such features to become particularly important and useful in the context of *loosely coupled scenarios* such as *Web-based services* and XML pipelines

Conclusions

1. Accessible data models support the development of more powerful and more resilient applications
2. Depending on the application scenario, alternative data model perspectives can be advantageous
3. A concisely defined canonical data model is essential as a pivot between different data model perspectives

More Information

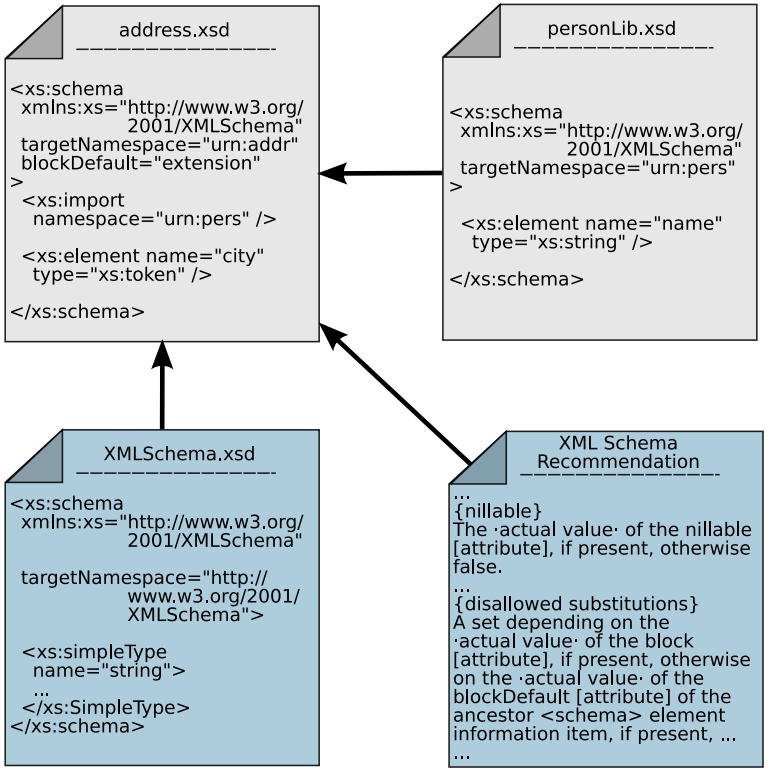
- WWW2007 poster on *XML-Based XML Schema Access:*
  - `http://www2007.org/poster934.php`
- My Master's Thesis on *Representation of XML Schema Components:*
  - `http://dret.net/netdret/docs/mt-ws2006-michel.pdf`
- Technical report on *SPath: A Path Language for XML Schema:*
  - `http://dret.net/netdret/docs/wilde-irep07-001-spath.pdf`

Related Work

- *XML Schema Component Designators* ("SCDS"; a W3C Working Draft from 2005)
- Litani's proposal for *An API to Query XML Schema Components and the PSVI* (XML Europe 2004)
- Sperberg-McQueen's paper on *Applications of Brzozowski derivatives to XML* (Extreme Markup Languages 2005)
- XML representations of augmented Infosets
  - Thompson's "synthetic XML" for representing the PSVI (2003)
  - Wilde's *Extensible XML Information Set* (2003)
  - Processor-proprietary dump formats of the PSVI
- Various publications by Brüggemann-Klein and Wood

Misalignments of the Transfer Syntax

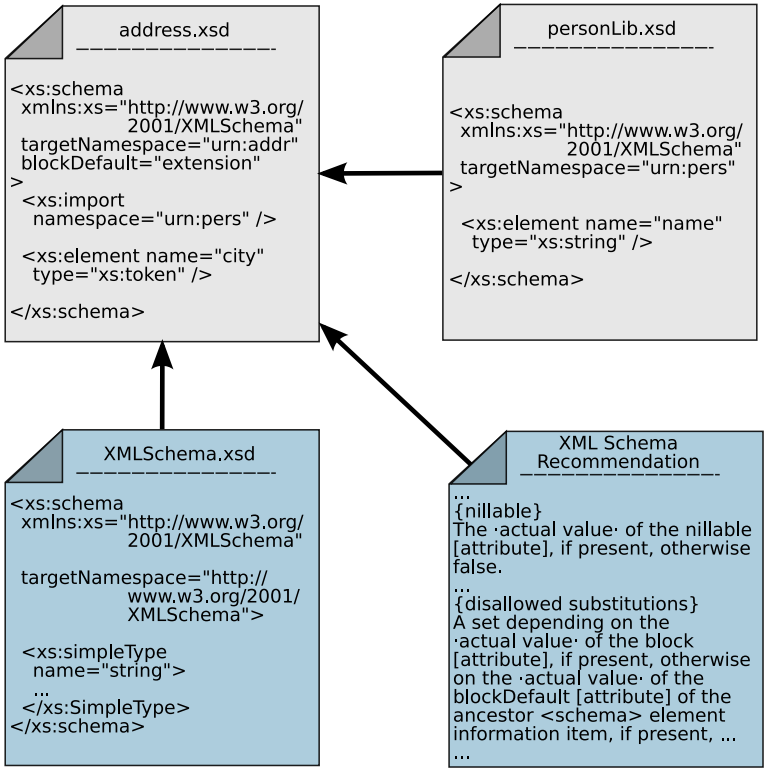
Schema-assembly in the transfer syntax:



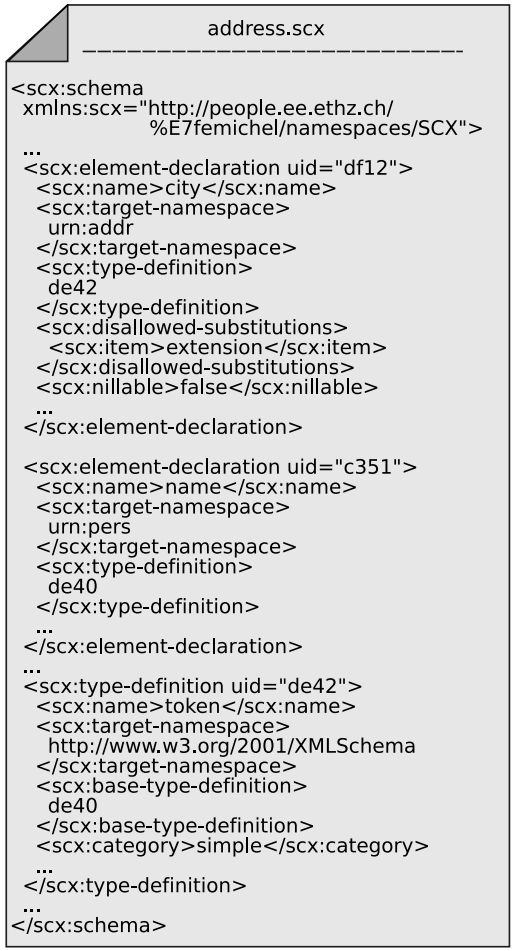
- Components are spread over multiple documents
- Properties are separated from the respective components
- Some information is not part of the documents:
  - *Built-in types*
  - *Default values*
  - Rules from the recommendation

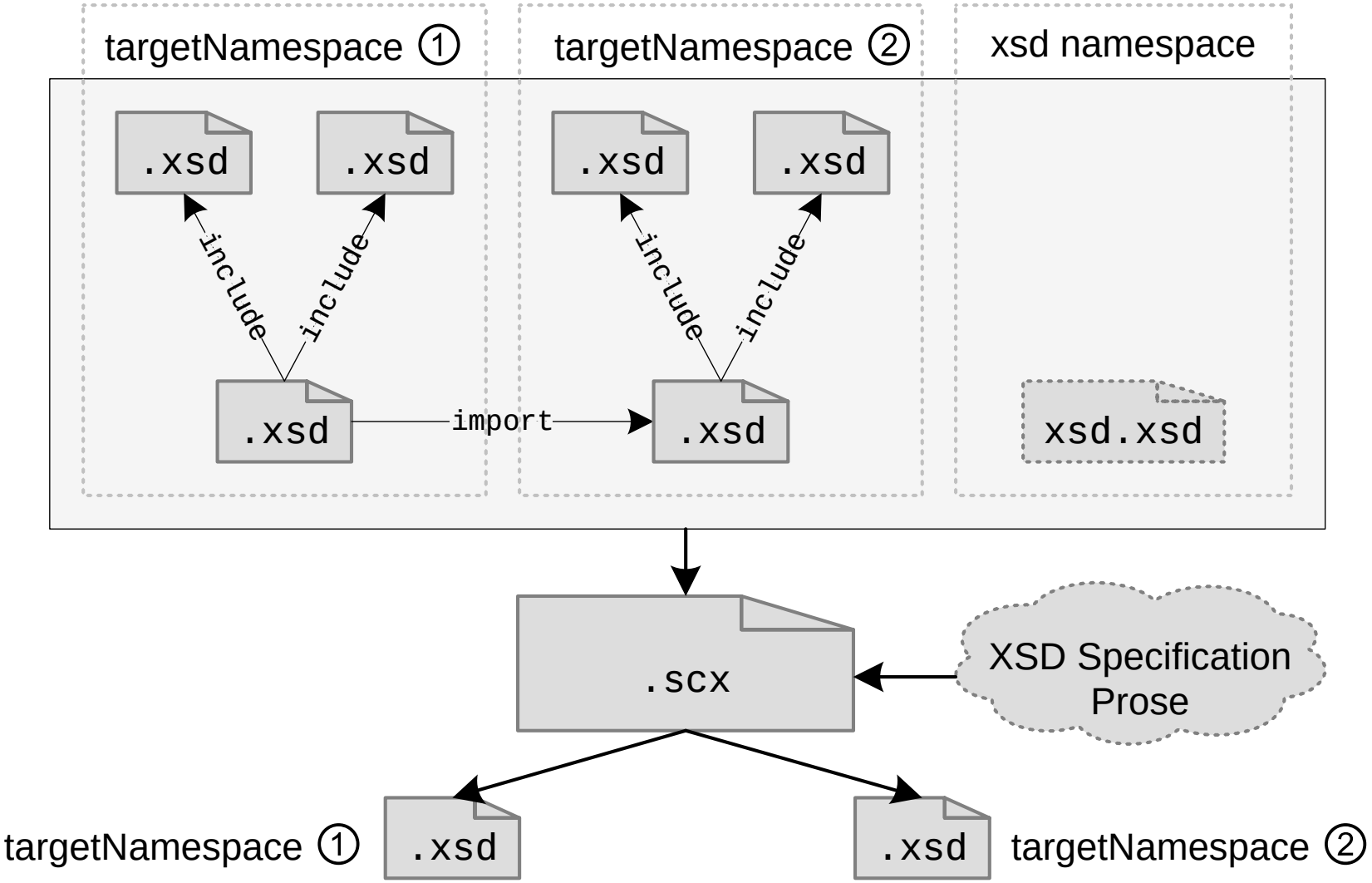
Self-contained Representation through SCX

Schema-assembly in the transfer syntax:



Assembled Schema in one document:




targetNamespace ①
targetNamespace ②
xsd namespace
.xsd
.xsd
.xsd
.xsd
include
include
include
include
.xsd
import
.xsd
xsd.xsd
.scx
XSD Specification Prose
targetNamespace ①
.xsd
.xsd
targetNamespace ②

Marked Regular Expressions

- Introduced by McNaughton and Yamada in 1960
- A *marking* makes symbols of a regular expression distinct by adding indices
  - $E = (c(a+b)^*a)$ becomes $E' = (c_1(a_1+b_1)^*a_2)$
- Applications:
  - Ambiguity checking of regular expressions
  - Efficient construction of the related finite automata

Derivatives

- Introduced by Brzozowski in 1964
- A derivative of a regular expression with respect to a certain marked symbol is the regular expression which describes the set of strings that can follow this symbol
  - Given a marked regular expression

$$E = ((a_1 + b_1)c_1 a_2^*)$$

the derivation with respect to a_1 is

$$D_{a_1}(E) = (c_1 a_2^*)$$

Expansion of *All* Model Groups

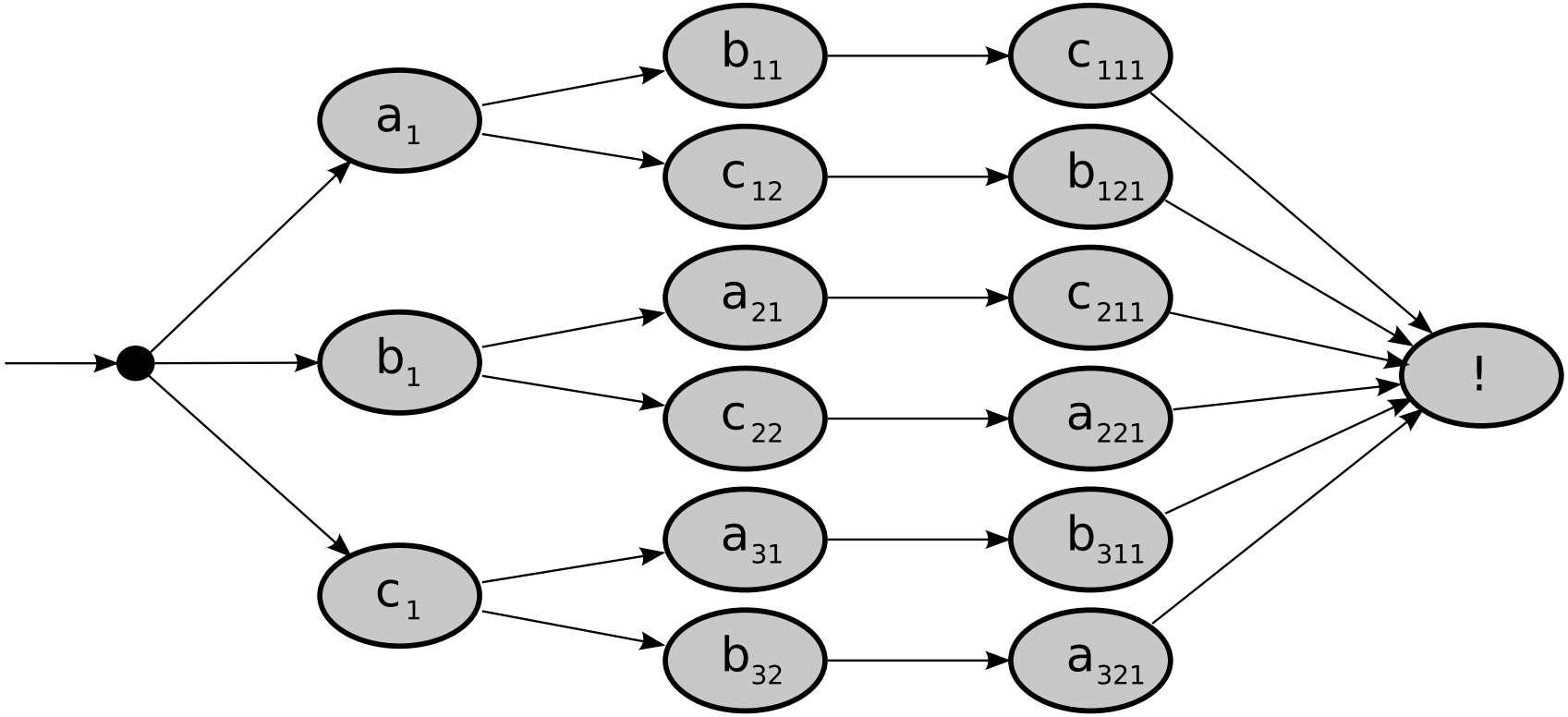


Figure: Expansion of $(a \,\&\, b \,\&\, c)$

Further Use Cases

- Web-based services & versioning:
  - An application receives input documents that conform to an XML Schema which is different from the one against which the application has been written
  - If the Schema's data model is accessible, the application can react in a sensible way (fallback to parent types, application of *mustIgnore* paradigms)
- Annotated Schemas & Ontologies:
  - If the Schema's data model is accessible and meta-information is encoded in an XML format, applications can access the metadata while processing instances:
  - `$elem/scf:get-type(.)//scx:application-information/rdf:*`