# Four Basic Building Principles (Patterns) for XML Schema

Anne Brüggemann-Klein

TU München

# Confronting the dragon (XML Schema)

**Reasoning** about XML Schema on the topic of the four building principles or **patterns**

- Russian Doll
- Salami Slice
- Venetian Blind
- Garden of Eden

and their **descriptional power**, to answer the following **questions**

- Is it possible to describe any set of documents that can be described by some XML schema with a schema that follows any of the four patterns?
- What is the relationship in descriptional power?
- Is there an element of constructiveness in the relationship?

on the basis of a **concise model** and **well-defined properties**

➢ **investigate the folklore, settle the myths**

# Agenda

Models for XML documents and for XML schemas

Patterns

Transformations

Conclusion

# Models

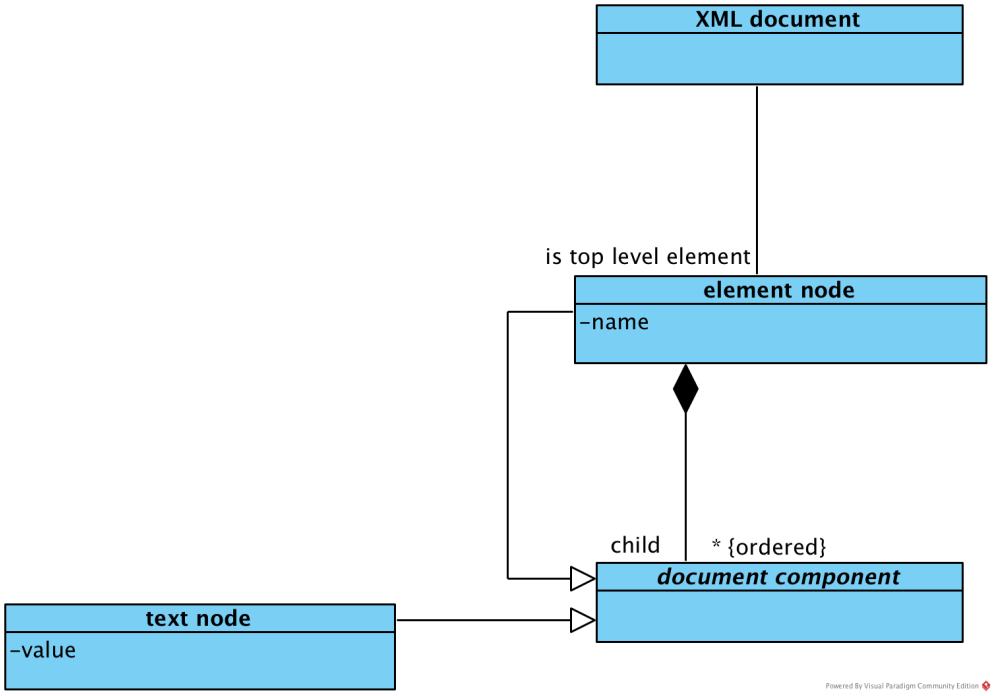
# XML document from core components

```
3  <book>
4      <title>Being a Dog Is a Full-Time Job</title>
5      <author>Charles M. Schulz</author>
6      <character>
7          <name>Snoopy</name>
8          <friend-of>Peppermint Patty</friend-of>
9          <since>1950-10-04</since>
10         <qualification>extroverted beagle</qualification>
11     </character>
12     <character>
13         <name>Peppermint Patty</name>
14         <since>1966-08-22</since>
15         <qualification>bold brash tomboyish</qualification>
16     </character>
17 </book>
```

***element nodes***

***text nodes***

# Modeling XML Core




XML document has two types of nodes

- element nodes
- text nodes
- no attribute, comment, namespace etc nodes

with

- properties (name ~ expanded name, value ~ sequence of Unicode code points)
- a relationship (edges): child

Constraints for ordered tree structure

- finite, at least one element
- no circles
- in-degree 0 or 1 (orphan or unique parent)
- connected

➢ unique root element, finite paths leading to leaf nodes

➢ depth (distance from root) and height (distance from leaves)

XML document = root element

# XML document from core components

# XSD schema from core components

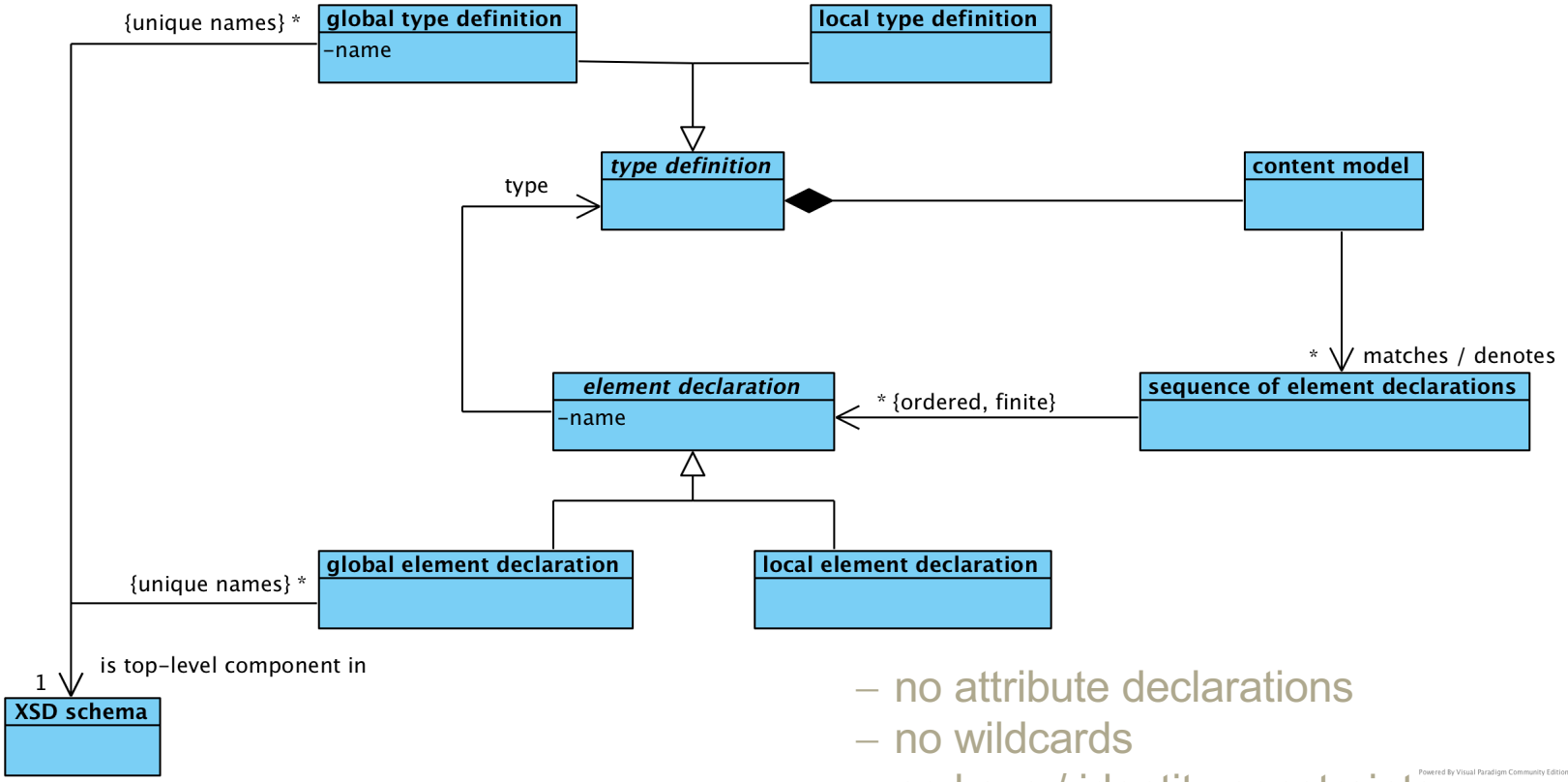
*element declarations (local / global)*

*type definitions (local / global)*

*references to global components*

```
4  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
5      <xs:element name="book" type="bookType"/>
6      <xs:element name="title" type="xs:string"/>
7      <xs:element name="author" type="xs:string"/>
8      <xs:complexType name="bookType">
9          <xs:sequence>
10             <xs:element ref="title"/>
11             <xs:element ref="author"/>
12             <xs:element name="character" minOccurs="0" maxOccurs="unbounded">
13                 <xs:complexType>
14                     <xs:sequence>
15                         <xs:element name="name" type="xs:string"/>
16                         <xs:element name="friend-of" minOccurs="0" maxOccurs="unbounded"
17                             type="xs:string"/>
18                         <xs:element name="since" type="xs:string"/>
19                         <xs:element name="qualification" type="xs:string"/>
20                     </xs:sequence>
21                 </xs:complexType>
22             </xs:element>
23         </xs:sequence>
24     </xs:complexType>
25 </xs:schema>
```

# Modeling XSD Core

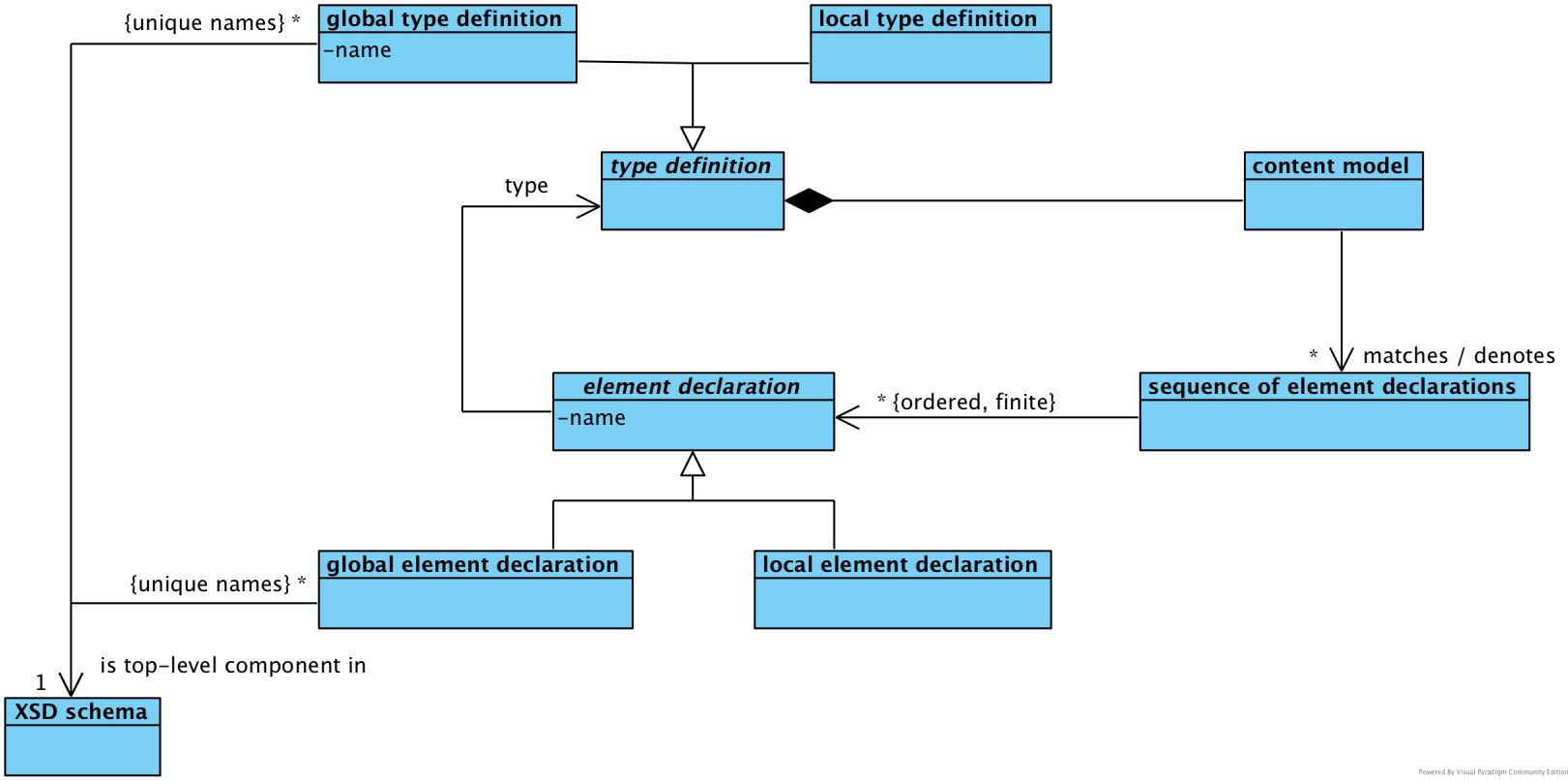



XSD Core has two main types of components (both can be global or local)

- element declarations
- (complex) type definitions

- no attribute declarations
- no wildcards
- no keys / identity constraints
- no assertions
- no annotations
- no inheritance
- no data types
- no control of mixed content

# Modeling XSD Core



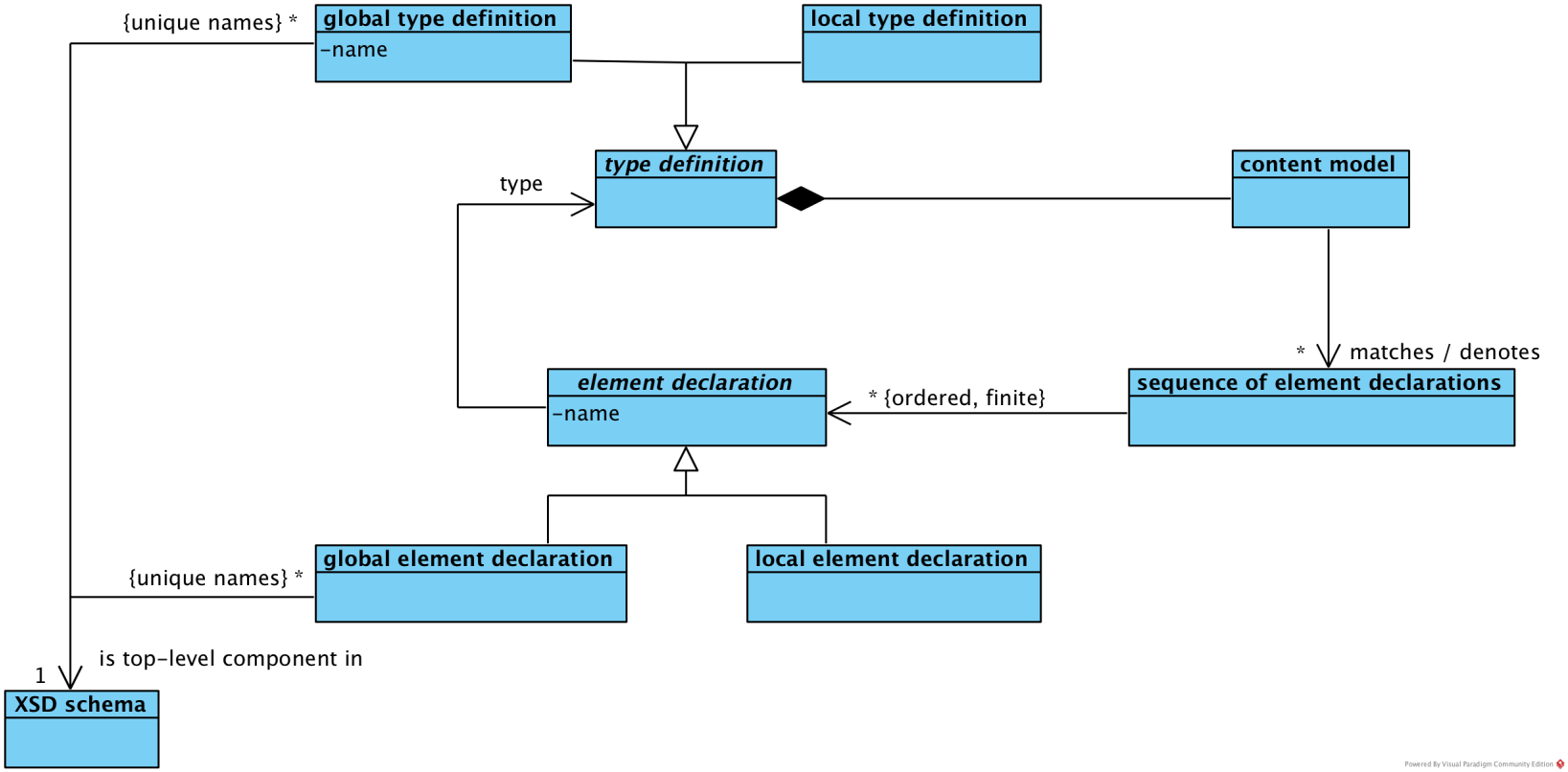

Element declaration associates a type definition

- references a global type definition by name
- contains an unnamed local type definition

Type definition has a content model (expression over element declarations), which

- references global element declarations by name
- contains named local element declarations

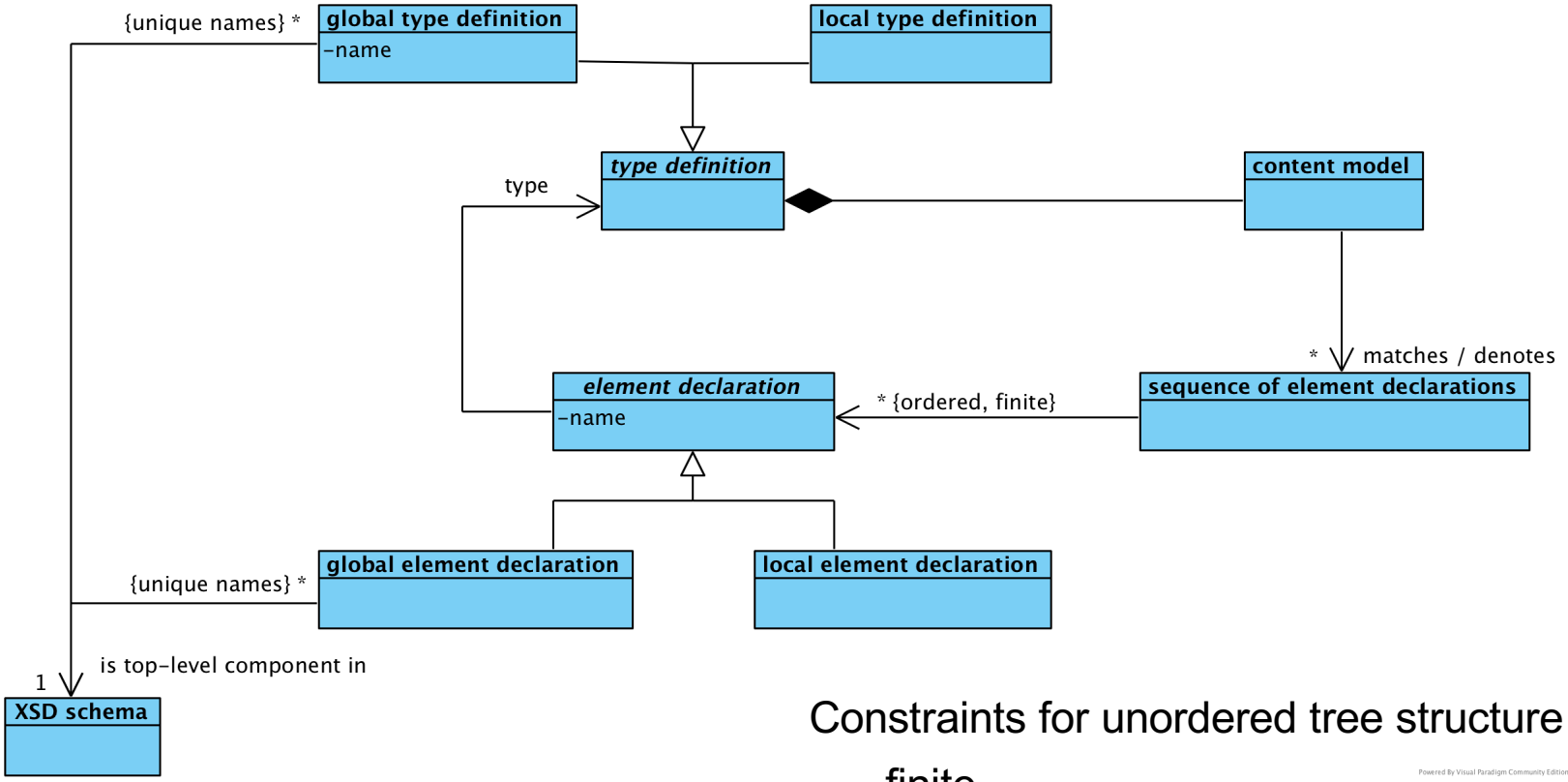
# Modeling XSD Core




Content model (via the expression) matches / denotes finite sequences of element declarations

- it does not matter if element declarations are referenced or contained in the content model

# Modeling XSD Core




Relationship contains

- schema contains global type defs / el decls
- element declaration can contain local type def
- type definition can contain local element decl

Constraints for unordered tree structure

- finite
- no circles
- in-degree 0 or 1 (orphan or unique parent)

➢ unique root component is XSD schema itself

➢ depth and height

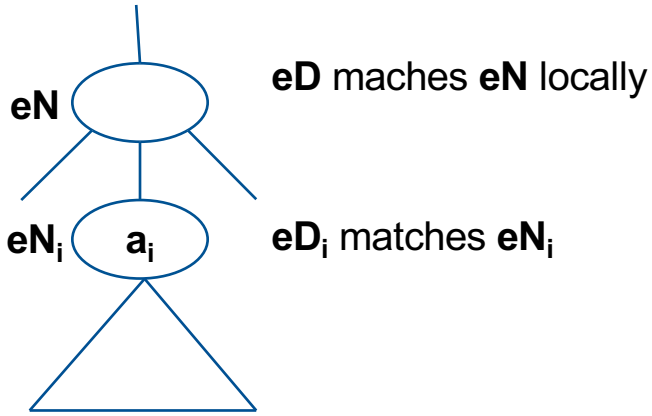
# XSD schema from core components

# Validation (read in language of models)

Element declaration (local or global) **eD** matches an element node **eN**
(**eN** is valid with respect to **eD**)

- **eN** has *element* children $\mathbf{eN_1},\ldots,\mathbf{eN_k}$
- **tD** type definition of **eD** (locally contained or globally referenced)
- content model of **tD** matches / denotes sequence of element declarations $\mathbf{eD_1},\ldots,\mathbf{eD_k}$ so that, for $1 \leq \mathbf{i} \leq \mathbf{k}$,
  a) names $\mathbf{a_i}$ of $\mathbf{eN_i}$ and $\mathbf{eD_i}$ are identical
  b) each $\mathbf{eD_i}$ matches $\mathbf{eN_i}$

→ **eD** matches **eN** locally



# Validation (read in language of models)

XSD schema **s** matches and XML document **d** (or **d** is valid with respect to **s** or **d** is an instance of **s**)

- **s** has a global element declaration that matches **d** (AKA the root element of **d**)

The language **L(s)** or XSD schema **s** is the set of all its instances.

**Corollary**: A schema that has no global element declarations has no instances
(its language is the empty set)

Two XSD schemas are equivalent if their languages (sets of instances) are equal.

- we care about constructive transformations between equivalent schemas.

# Patterns

# Four patterns for XSD schemas

Four „pure“ patterns through binary choice → examples in paper

- element declarations
  - all local (with exception of global entry points that are never referenced)
    (no references to global element declarations)
  - all global
- type definitions
  - all local (with the exception of definitions of pre-defined data types, which may be referenced)
  - all global (including pre-defined data types, which may be considered as globally defined)

| | Element declaration | |
|---|---|---|
| Type definition | local | global |
| local | Russian Doll | Salami Slice |
| global | Venetian Blind | Garden of Eden |

# Global – local – who cares?

Global vs local

- global components can be re-used (within a schema and across schemas)
- local components are encapsulated within their containing components

Facilitates design decisions

- re-use and encapsulation
- coupling and cohesion

Question addressed in this paper: **Are the four patterns created equal**?

- does choice of pattern affect the kinds of languages that we can define through a pattern (**descriptional power**)?
- what can we expect of tools (**algorithmic conversion**)?

# Summary of findings (*constructive* comparisons)

Compare sets of XSD schemas (specialized schema languages) **S** and **S′**
- set of Russian Doll schemas **RD**
- set of Salami Slice schames **SS**
- set of Venetian Blind schemas **VB**
- set of Garden of Eden schemas **GE**

**S′** is at least as powerful as **S** (**S —> S′**)
- there is an algorithms that converts each schema in **S** into an equivalent schema in **S′**

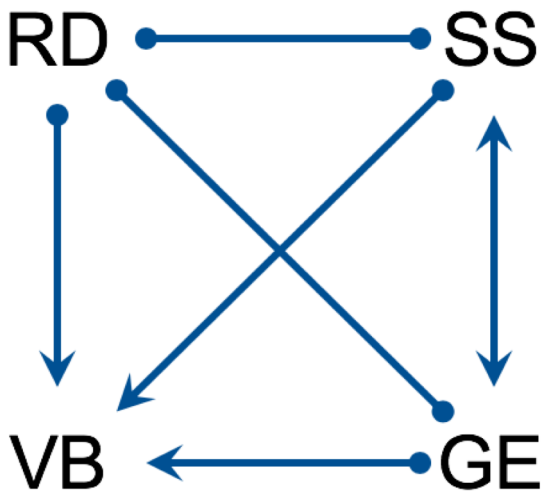
**S′** is incongruent with **S** (**S ◦— S′**)
- we can identify a specific schema in **S′** that is not equivalent to any schema in **S**

**S′** is strictly more powerful than **S** (**S ◦—> S′**)
- **S′** is more powerful than and incongruent with **S**

**S′** is equally powerful to **S** (**S <—> S′**)
- **S′** is at least as powerful as **S** and vice versa

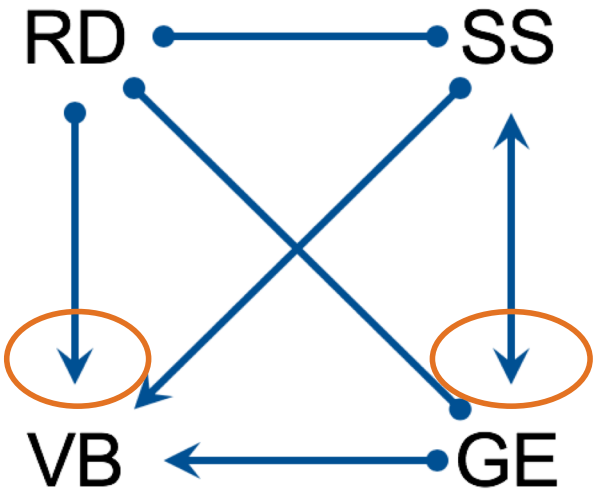
**S′** is incomparable with **S** (**S ◦— ◦ S′**)
- **S′** is incongruent with **S** and vice versa

**NOTE**: these comparisons are considerably more powerful than subset and set difference relationships



# Transformations

# Exhibit A



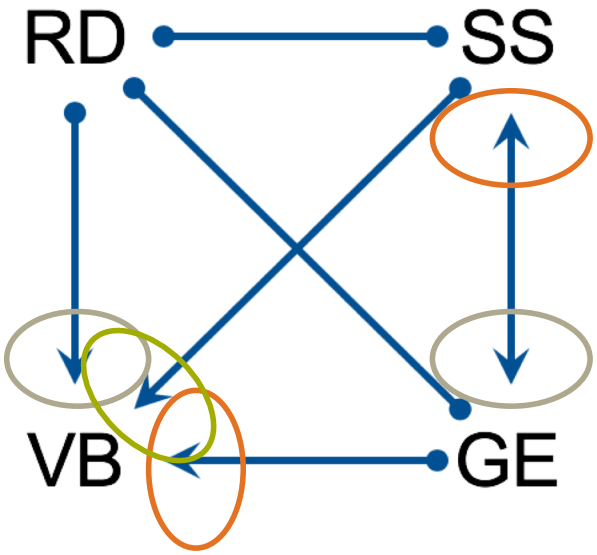

**Local type definitions can be made global**

- how? – one type definition at a time (*constructively*)
  - give the local type definition a new and unique name and copy it to the global level
  - replace the local type definition in the element declaration in which it occurs with a reference to the new global definition
  - ➢ one less local type definition (no new ones were introduced)
  - ➢ instances are unchanged

**Caution: the argument cannot be replicated for local element declarations**
**Double caution: this does not establish the „negative“ statements**

- why?
  - local element declarations are named → name conflicts at global level
  - renaming of elements changes instances (whereas names of type definitions are only relevant within a schema)
- a single local element declaration *whose name does not appear at a global level* **CAN** be made global → special case

# Exhibit B




**A single global type definition can be made local** (a reference to a global type definition can be turned into a local type definition)**.**
**In the case that all element declarations are global**
(and content models use references to global element declarations)**,**
**these changes can be re-iterated**
**and will result in a schema without global type definitions.**

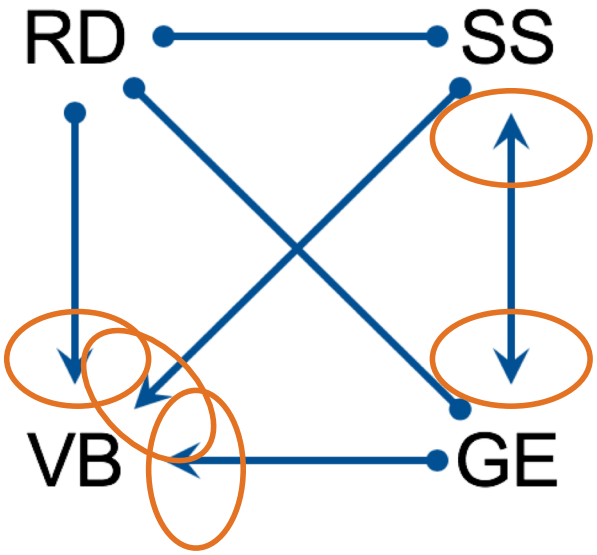
- how? – one type definition at a time (*constructively*)
  - replace a reference to a global type definition with a local copy (without the name)
  - ➢ instances are unchanged
  - ➢ number of references to a global type definition is reduced by one IF the copy of the global type definition has no local element declarations which in turn reference a global type definition
  - ➢ a global type definition that is eventually no longer referenced can be removed

**Dual argument: A global element declaration can be made local. In the case that all type definitions are global** (and element declarations use references to determine their type)**, these changes can be re-iterated and will result in a schema whose global element declarations are never referenced. [Global element declarations will be kept as entry points.]**

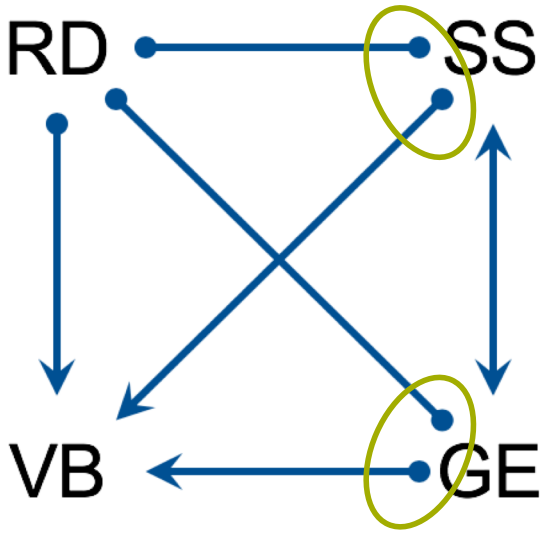
# Corollary

**The set of all XSD schemas is equally powerful to VB.**

- why?
  - start with an arbitrary XSD schema
  - make all its local type definitions global
  - make all its global element declarations local

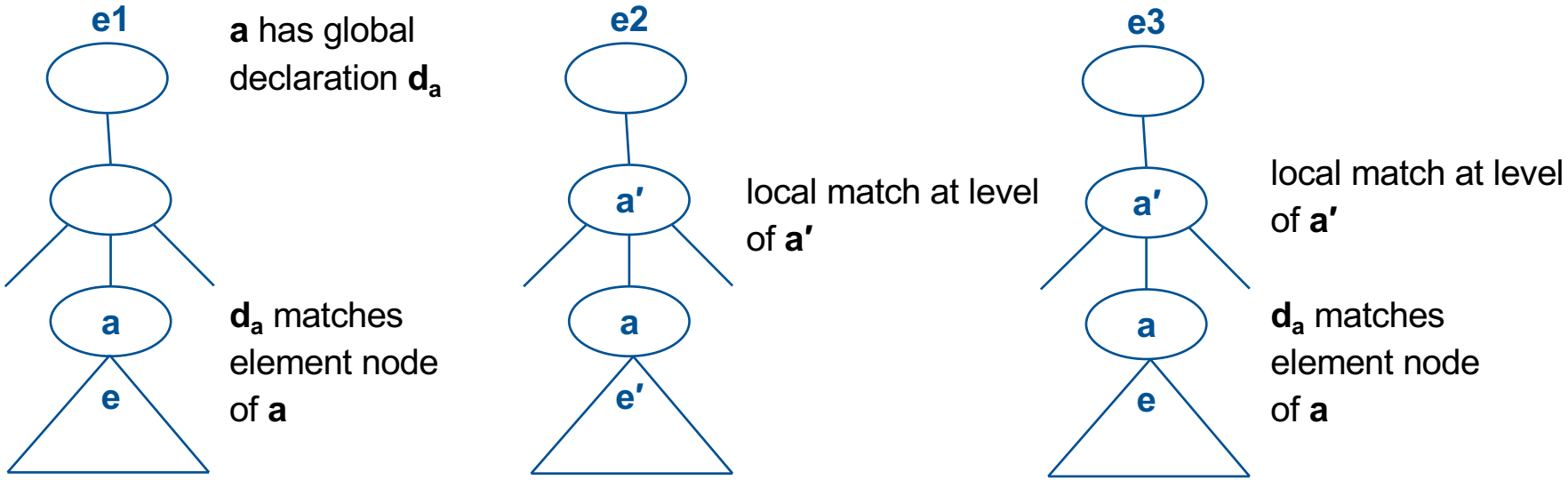


# Exhibit C




**The language of an XSD schema s that only has *global* element declarations (such as of a schema in SS or in GE) satisfies the substitution principle:**

If **s** has two instances **e1** and **e2**, and if **e1** has a sub-element structure **e** that is rooted in an element named **a**, then **e** can be substituted in **e2** for any sub-structure that is rooted in an element named **a**, and the resulting document **e3** is again an instance of **s**.




**a** has global declaration $\mathbf{d_a}$

$\mathbf{d_a}$ matches element node of **a**

local match at level of **a′**

local match at level of **a′**

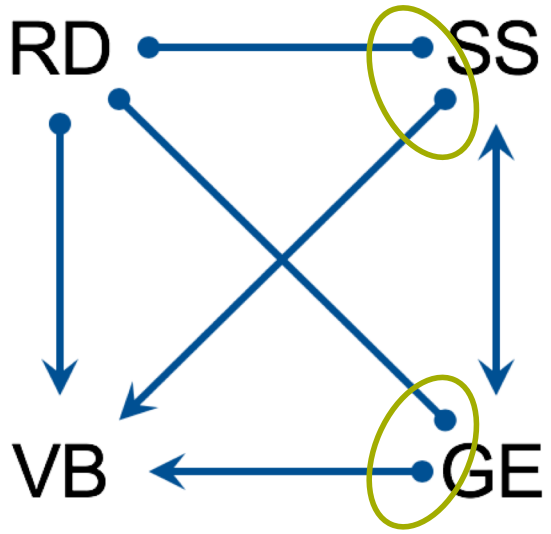
$\mathbf{d_a}$ matches element node of **a**

# Exhibit C

The substitution principle is a weakness of XSD schemas with global element declarations (an in-ability to tie structures to locations).

This weakness is overcome through local element declarations.

**There is a schema in RD (and, hence, of a schema in VB) whose language does not satisfy the substitution principle. Hence, there is no schema in SS and no schema in GE that is equivalent to that schema in RD.**

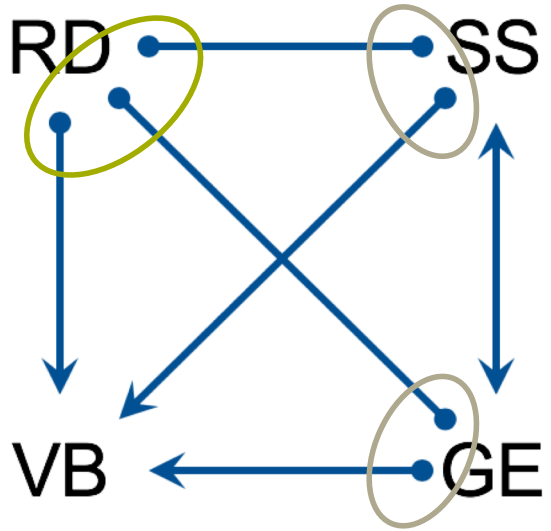



```
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
4      <xs:element name="a">
5          <xs:complexType>
6              <xs:sequence>
7                  <xs:element name="a"/>
8              </xs:sequence>
9          </xs:complexType>
10     </xs:element>
11 </xs:schema>
```

The schema denotes the single document <a><a/></a>

The singleton set with this document as a member does not satisfy the substitution principle

# Exhibit D




**The instances of any schema in RD are limited in height by the height of the schema.**

Proof by induction, looking at depth: Let s be a schema in RD, and let eN be an element node in an instance of s of depth k. Then k is validated against an element declaration in s of depth ≥ k.

**There is a schema in SS (and, hence, in GE and in VB), whose instances have unlimited heights. No schema in RD is equivalent to that schema.**

```
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
4      <xs:element name="a">
5          <xs:complexType>
6              <xs:sequence minOccurs="0" maxOccurs="1">
7                  <xs:element ref="a"/>
8              </xs:sequence>
9          </xs:complexType>
10     </xs:element>
11 </xs:schema>
```

The instances of that schema are all chains of nested elements a (unlimited heights)

# Concluding remarks

# Conclusion

What about DTDs? → they „are“ XSD schemas in SS

What about attributes? → local attribute declarations are easy to integrate

What about namespaces? → names in the core models can be expanded names (no prefixes)

What about specific content models or XML Schema constraints on content models?
→ not considered here
→ the XSD core model provides an abstraction
(element declaration matching a sequence of element declarations)

Can‘t local element declarations be accomodated through XML Schema assertions?
→ yes, probably, use them if you must ☺

Are there aspects in XML Schema validation that I have missed?
→ i am encouraged to look carefully at wildcards and PSVSs
→ I still feel overwhelmed by the subtleties of the recommendation ☹

# Conclusion

Since VB is the most powerful class of schemas, should all schemas be written as VB schemas?

→ no, there are other valid considerations, see
Eve Maler on schemas for UBL (extensibiltity)
AB et al on modeling XML applications with inheritance

→ now you know for sure what you loose if you don‘t,
so you can make principled decisions

Do we need / want to have a more complete summary of the four patterns and potentially more?

→ sounds appealing

→ are there volunteers to collaborate or to take over?

How about Relax NG?

→ the model needs to be expanded, since Relax NG has global element declarations that are referenced not by element name but by independent IDs

→ my hypothesis is that these are replaceble by local element decls

→ this seems to be related to earlier work on parsing with tree automata

→ the true power of Relax NG seems to stem from its relaxation of unambiguity constraints and from the uniform treatment of elements and attributes, not from element declarations per se

→ worth to investigate further

# Conclusion

Do we know the practical distribution of patterns?

- → I am shamefully ignorant and open to be educated (or to define a student project)
- → which schemas should be considered?

What about tools to analyze and to transform schemas?

- → some do something (oXygen, freeformatter.com)
- → some promise more than they can deliver (Netbeans)
- → it seems that TRANG is used successfully
- → it seems worth doing (as a student project)

**Thank you, listeners, for your attention and for your interest and hopefully for your questions**

**A heartfelt thank you to the programming committee and the anonymous reviewers for their outstanding contribution to the paper**