Active XML Schemas

30/11/2001

Michael Schrefl
Martin Bernauer

Johannes Kepler University Linz
Department of Business Informatics
Data & Knowledge Engineering

JOHANNES KEPLER UNIVERSITÄT LINZ
A-4040 Linz – Austria – Europe
Altenberger Straße 69

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

Motivation

- Web documents of e-business partners
  - change dependencies
  - autonomous, loosely coupled
- Content drawn from databases or other documents, some content is in no database
  - web-based solution for web-only content
  - ex.: “active document” deletes its out-dated job-ad
- Managing content
  - automatic
  - asynchronous
  - potentially according to event history

Active Behavior of Web-documents

- Bonifati/Ceri/Paraboschi: *Active Rules for E-Services* (VLDB Journal 2001)
  - apply ECA-paradigm to XML documents
  - use active rules for pushing reactive services
  - introduce basic approach based on *simple mutation events*
- Active XML-Schemas: from simple to high-level support for *active behavior* in XML
  - based on experience from
    - conceptual modeling of Business Rules
    - active object-oriented database design
    - event-based systems
  - employ Bonifati's approach at lower level
    - for simple mutation events
    - for implementing event notification

3

Active XML Schemas build on ...

- Conceptual modeling of Business Rules
  - object-oriented representation: *Situation/Activation Diagrams* (ICDE 1997)
  - events are first class objects, have an event type, are collected into event classes
  - calendar events, method events, abstract events, ..
  - past, current, and future (scheduled) events
  - logical event classes: provide functionality of complex events, member events determined by querying other event classes
- Event-based systems
  - low coupling
  - publish/subscribe events

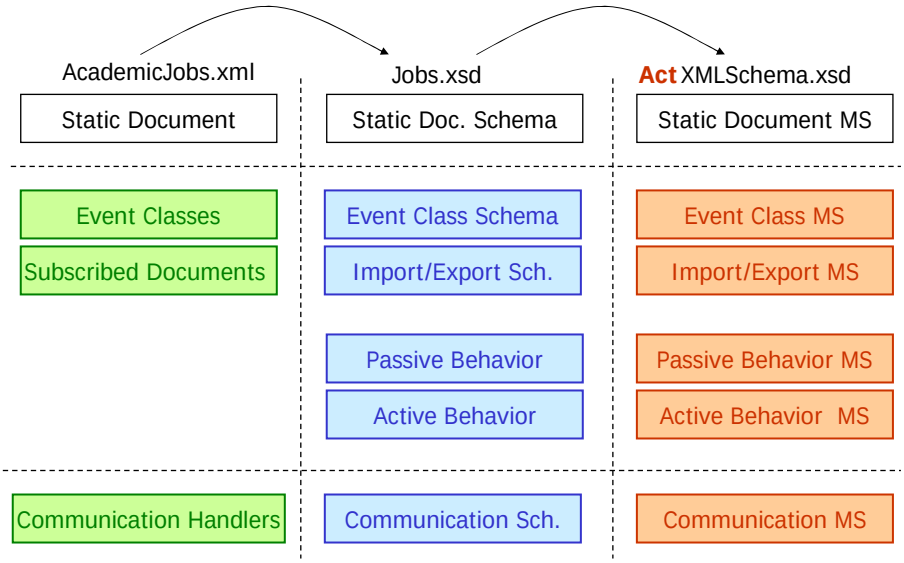
4

Objectives and Approach

- Extension of XML-Schema
  - active behavior comes with document schema
  - provides for reuse and interoperability
  - easier to design, implement, and maintain
- Document-centered
  - events are “first-class elements”
  - past, current, and scheduled events can be queried
- Distribution
  - low coupling of documents: event communication
  - full locality of active rules
- Transparency
  - event source transparency (database vs document, remote vs local)
  - communication transparency (push vs pull)

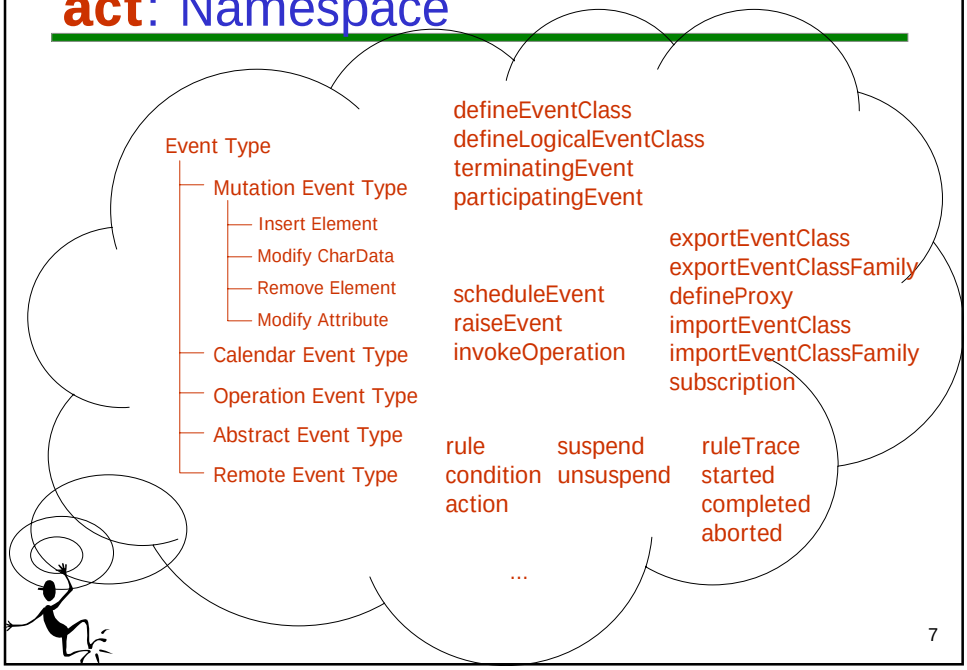
5

Extension of XML Schema

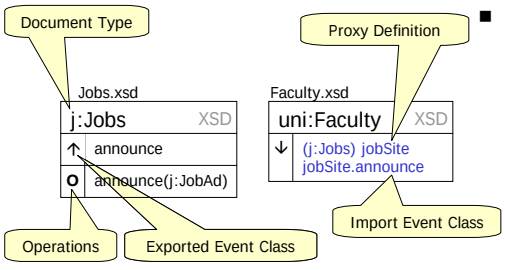


6

act: Namespace



Export/Import of Event Classes



- Schema level
  - define event class

```
<act:defineEventClass
  name='announce'
  evtCategory='OperationEvt'
  memberType='j:ExecAnnounce' .../>
```

  - export event class definition

```
<act:exportEventClass
  name='announce'/>
```

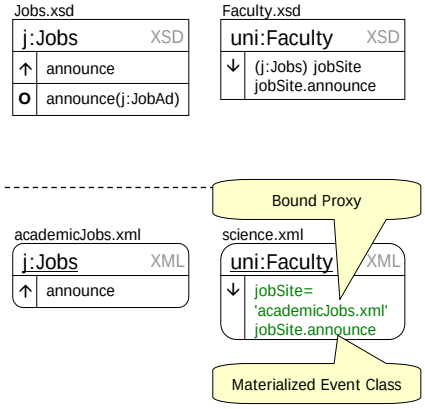
  - define document proxy with document type

```
<act:defineProxy name='jobSite'
  forDocType='j:Jobs'/>
```

  - import event class definition

```
<act:importEventClass proxy='jobSite'
  eventClass='announce'/>
```

Export/Import of Event Classes



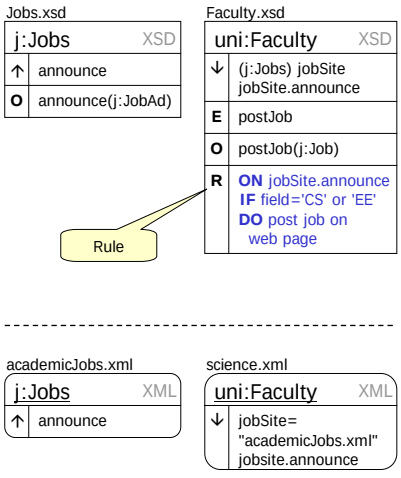
- Instance level
  - bind proxy

```
<act:subscription proxy='jobSite'>
 <act:remoteDoc>academicJobs.xml</>
</act:subscription>
```

  - materialize event class

```
<act:eventClass name='jobSite.announce'>
 <act:event id='e171' status='processed' … >
  <act:remoteEvent
     evtCategory='OperationEvt'>
   <act:params>
    <j:job id='j2001-11-19_351'>
     <j:field>CS</field>
     <j:title>Application Engineer</>
     <j:appDeadline>2001-12-10</>  …
    </>
   </>
  </>
 <act:event id='e172' status='occurred' … >
  …
 </>
</>
```


Rules

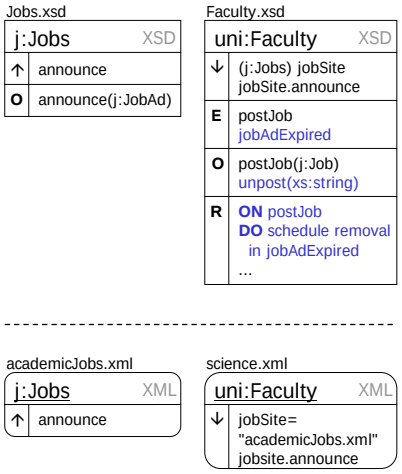


- Defined at schema level
- All components local
  - for communication: export and import events

```
<act:rule on='jobSite.announce'
 name='announceJobRule'>
 <act:condition>
  <xsl:value-of select="$e//j:job[j:field='CS' or
   j:field='EE']"/></>
 <act:action>
  <act:invokeOperation>postJob($c)</>
 </>
</>
```

$e .. bound event, $c .. bound condition result

Scheduling Events

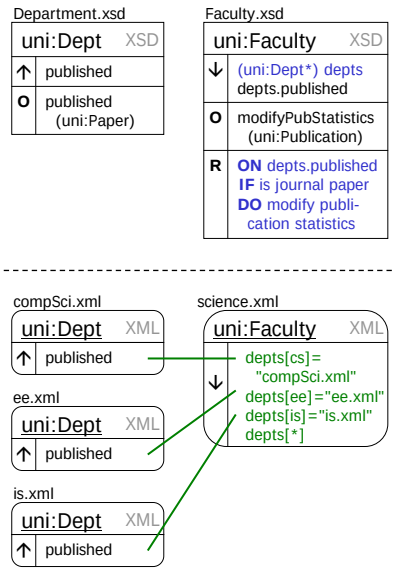


- Rules may schedule future events (may be modified later like calendar entries)
- Ex.: schedule removal of posting when job add is posted

```
<act:defineEventClass name='jobAdExpired' .../>
<act:rule on="postJob" ...>
  <act:action>
    <act:scheduleEvent in='jobAdExpired'>
      <act:event occTime='{$e//j:appDeadline}'>
        <uni:expJob id='{$e//j:job/@id}'/>
...</>

<act:rule on='jobAdExpired' ...>
  <act:action>
    <act:invokeOperation>
      unpost({$e/uni:expJob/@id})
...</>
```


Event Class Families



- Several related documents export same type of event class
- Collected into event class family
- Referenced by set proxy
- Member qualifiers used to select specific event class
- Unqualified reference returns union

Event Class Families

faculty.xsd

```
<act:defineSetProxy name='depts'
 forDocType='uni:Dept'/>
<act:importEventClassFamily
 proxy='depts'
 eventClass='published'/>
```

```
<act:rule on='depts.published'
 name='modifyPubStatRule'>
 <act:condition>
  <xsl:value-of select="$e//uni:publication
   [@type='journal']"/>
 </>
 <act:action>
  <act:invokeOperation>
   modifyPubStatistics($c)</>
 </>
</>
```

science.xml

```
<activeDocument>
 <staticDocument>
  <uni:faculty>
   <uni:name>Science</> ...
 </></>
 <act:subscription proxy='depts'>
  <act:remoteDoc qual='cs'>compSci.xml</>
  <act:remoteDoc qual='ee'>ee.xml</>
  <act:remoteDoc qual='is'>is.xml</>
 </>
 <act:eventClasses>
  <act:eventClass name='depts.published'
   qual='cs'>
   <act:event id='e745' status='occured' ...>
    <act:remoteEvent>
     <act:params>
      <uni:publication>...</> ...
   </></>
  <act:eventClass name='depts.published'
   qual='ee'>...</>
</></>
```


Multiple time stamps, rule trace

- May deliver events ahead of occurrence time (scheduled time)
  - e.g. salary deposit
- Deadline: *post mark* or *delivered* ?
  - posted time
  - delivered time
- Event status: scheduled/occurred/processed
- For each event, rule trace for each triggered rule: started, aborted/processed
- Time stamps, status, rule trace can be queried in logical event class definitions and rules

Logical events

- Composite events in databases
  - e.g., sequence event C=A ; B
  - relative ordering w.r.t. occurrence time
  - event contexts (chronical, recent, ...)
  - incremental event detection (past → now)
- With web documents
  - multiple time stamps
  - context combination difficult to grasp
- Active XML Schemas: *logical event classes*
  - events recorded with document
  - query events (past ← now → future) starting from *terminating event* to determine "complex" event
  - compare chosen time stamps as needed

Ex.: Logical EC "PubReceivedLate"

- EC *pubReceivedLate* collects all *published* events which
  - have been delivered in the new year despite being posted in the old year
  - providing the "active researcher status" of the author has already been determined for the publication year (otherwise the notification is not too late for being considered!)

Faculty.xsd

uni:Faculty	XSD
↓	(uni:Dept*) depts depts.published
E	pubReceivedLate publishActResStatus
O	publishActResStatus (uni:Author,xs:gYear)

```
<act:defineLogicalEventClass
 name='pubReceivedLate' on='depts.published'>
 <act:basedOn>
  <act:terminatingEvent alias='p'
   fromEventClass='depts.published'/>
  <act:participatingEvent alias='a'
   fromEventClass='publishActResStatus'/></>
 <act:where>
  <xsl:value-of select="$p[
   act:Year(@deliveredTime)>act:Year(@postedTime)
    and @occTime>$a/@occTime
    and uni:author=$a/uni:author
    and act:Year(uni:pubDate)=
       act:Year($a/uni:pubDate)]"/>
 </>
</>
```

Exception Rules

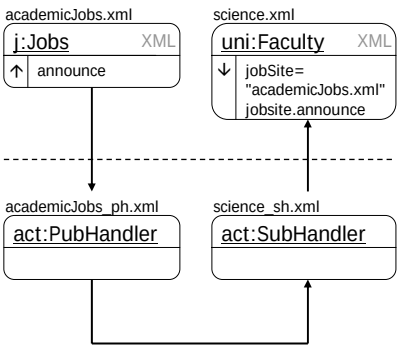
- Rules handling exceptions (e.g., late delivery)
- No additional concepts necessary, only design distinction
- Re-act to specific logical events which determine exceptional situations
- E.g., re-run active researcher check on author of publication in *pubReceivedLate* event

```
<act:rule on='pubReceivedLate'>
  <act:action>
    <act:invokeOperation>
      pubActResStatus($e//uni:author, $e//uni:year)
    </>
  </>
</>
```

17

Communication Transparency

- Event communication delegated to *publication* and *subscription handlers* associated with documents
- Handlers negotiate communication strategy
  - push/pull
  - filtering
- If subscriber needs only subset of published event class, employ Bonifati's approach to push filter triggers to publisher



18

Managing Active XML Schemas

- Architecture and components (rule manager, event detectors, ...) based on experience from active database systems
- Yet, new issues due to specific environment
  - *eager* versus *lazy* strategy for rule consideration (document needs not to be updated immediately after event arrival but only before being read!)
  - collect all incoming events into document's event classes before considering rules
    - this allows rules to know what is "pending"
    - compare: skim remaining e-mails before answering an e-mail (as a subsequent e-mail may give a different picture)

19

Conclusion

- Active XML Schemas provide *active behavior* for XML documents within their schemas
- Events and rules are
  - "first class elements"
  - maintained with document / schema
  - event history can be queried, comes with document (like in old "paper form processing")
- Enabling technology for E-Services
- Future work:
  - design guidelines
  - environment for rule management

20