Replication Mechanism of ZEMIS Ref

Tanja Küry

University of Bern

tanja.kuery@students.unibe.ch

09.01.2017
ZEMIS Ref

"ZEMIS Referenzdatenverwaltung"
Administration application for so called 'reference data'

- About 380 tables with relational dependencies
- Several applications use the data (clients)
- New applications outside scope of project ZEMIS
- Focus on replication mechanism
Replication: Initial State

Push architecture with direct database access
ZEMIS Ref replicates changes directly into the client’s db

- Direct database access
- Client - data mapping
- Database connection and schema
- ZEMIS Ref release required to add new clients
- Tight coupling - makes changing the schema complex
- Robust, no synchronisation issues, failures are detected, no heavy workload
Replication: Variant 1

Push architecture with web service
ZEMIS Ref pushes changes via SOAP web service

- Client provides web service
- Client - data mapping
- Dynamic subscriber list
- No direct database access
- Coupling loosened, schema can be changed
Variant 1: Pro and Contra

Push architecture with web service
Will be used for new ZEMIS application (temporary solution)

- Closest to initial state
- Robust
- No synchronisation issues
- Failure detection
- No heavy workload

- Contradicts pull over push policy
- Clients provide web service
- Coupling still tight
Replication: Variant 2

Direct pull architecture (on the go)
Clients pull each time they need information

- Always up to date
- Permission and authentication instead of mapping
- No direct access to database
- No need to store the ref data
Variant 2: Pro and Contra

Direct pull architecture (on the go)
30’000 users, 2500 calls/minute

- Always up to date
- No client - data mapping
- No direct access to database
- No need to store the ref data ⇒ browser caching

- Depends on ZEMIS Ref availability
- High performance requirements ⇒ several servers ⇒ high cost
- Robustness low, heavy workload
Replication: Variant 3

Pull architecture with caching

ZEMIS ref offers SOAP web service, clients pull and cache the data

- No direct access to database
- Deltas can be pulled
- Loose coupling, clients can be added easily, schema can be changed
- Clients can have differing data $\Rightarrow$ conflicts
- Robustness medium high
Variant 3: Pro and Contra

Pull architecture with caching
ZEMIS ref offers SOAP web service, clients pull and cache the data

- Follows pull over push policy
- Web service provided by ZEMIS Ref
- Loose coupling
- Robustness medium high

- When shall the clients pull?
- Clients can have differing data $\Rightarrow$ conflicts
**Variant 3: First Draft**

```xml
<x:refEntity joinedViewName="Addressdaten">
  <x:table name = "Addressdaten_BS" entityPart="business">
    <x:row index = "1" status="new">
      <x:active>true</x:active>
      <x:entry column = "last name" dataType="ALPHA_NUM">DJ</x:entry>
      <x:entry column = "first name" dataType = "ALPHA_NUM">Bobo</x:entry>
      <x:entry column = "address line 1" dataType = "ALPHA_NUM">Hauptstrasse 33</x:entry>
      <x:entry column = "address line 2" dataType = "ALPHA_NUM">4108 Witterswil</x:entry>
      <x:entry column = "phone number" dataType = "NUM">0610123456</x:entry>
      <x:entry column = "swissJN" dataType="BOOLEAN">J</x:entry>
      <x:entry column = "birth date" dataType="DATUM">08.08.1988</x:entry>
    </x:row>
    <x:row index = "2" status = "modified">
      <x:active>true</x:active>
      <x:entry column = "last name" dataType = "ALPHA_NUM">Doe</x:entry>
      <x:entry column = "first name" dataType = "ALPHA_NUM">John</x:entry>
      <x:entry column = "address line 1" dataType = "ALPHA_NUM">Münstergasse 14</x:entry>
      <x:entry column = "address line 2" dataType = "ALPHA_NUM">4053 Basel</x:entry>
      <x:entry column = "phone number" dataType = "NUM">0790123456</x:entry>
      <x:entry column = "swissJN" dataType="BOOLEAN">N</x:entry>
      <x:entry column = "birth date" dataType="DATUM">09.09.1999</x:entry>
    </x:row>
  </x:table>
</x:refEntity>

<x:table name="Adressdaten_txt" entityPart="text">
  <x:row index="1">
    <x:active>true</x:active>
    <x:entry column="id" dataType="NUM">1</x:entry>
    <x:entry column="lang_cd" dataType = "ALPHA_NUM">de</x:entry>
    <x:entry column="txt" dataType="ALPHA_NUM">Adressdaten eins deutsch</x:entry>
  </x:row>
</x:table>
```
Variant 3: Validation of First XML Draft

Pull architecture with caching
Implemented web service for pull architecture with caching

- Functionality testing with soapUi
- Load testing with Apache JMeter
- XML files too big
- Problem: XML-Firewall
Variant 3: Final XML Version

<ns4:refEntity joinedViewName="Adresse Daten">
  <ns4:table name="Adresse Daten BS" entityPart="business">
    <ns4:columns>
      <ns4:col dataType="ALPHA_NUM">last name</ns4:col>
      <ns4:col dataType="ALPHA_NUM">first name</ns4:col>
      <ns4:col dataType="ALPHA_NUM">address line 1</ns4:col>
      <ns4:col dataType="ALPHA_NUM">address line 2</ns4:col>
      <ns4:col dataType="NUM">phone number</ns4:col>
      <ns4:col dataType="BOOLEAN">swiss JN</ns4:col>
      <ns4:col dataType="DATUM">birth date</ns4:col>
    </ns4:columns>
    <ns4:rows status="new">
      <ns4:row index="0">
        <ns4:entry>DJ</ns4:entry>
        <ns4:entry>Bobo</ns4:entry>
        <ns4:entry>Hauptsrasse 33</ns4:entry>
        <ns4:entry>4108 Witterswil</ns4:entry>
        <ns4:entry>0610123456</ns4:entry>
        <ns4:entry>08.08.1988</ns4:entry>
      </ns4:row>
      <ns4:row index="1">
        <ns4:entry>John</ns4:entry>
        <ns4:entry>Münsterstrasse 14</ns4:entry>
        <ns4:entry>4053 Basel</ns4:entry>
        <ns4:entry>0790123456</ns4:entry>
        <ns4:entry>09.09.1999</ns4:entry>
      </ns4:row>
    </ns4:rows>
  </ns4:table>
</ns4:refEntity>
Variant 3: Conclusion and Further Work

- Define requirements
- Talk to stakeholders
- WSDL and XSD are the base
- XSD can be used for variant 1 (push)
- Human readable $\Rightarrow$ more abstraction
- Reduce size of XML
- Do not send information twice
Replication: Variant 4

Push notify to pull architecture with web services
ZEMIS ref pushes notifications about updates, clients pull if needed

- Web service provided by clients to push notification
- Web service provided by ZEMIS Ref to pull
- No heavy workload, responses can be scheduled
- Clients need to determine whether to pull or not
- Robustness good, no differing data
- What if one client doesn’t get a notification?
Variant 4: Pro and Contra

Push notify to pull architecture with web services
ZEMIS ref pushes notifications about updates, clients pull if needed

- No direct access to database
- Loose coupling: Easy to add new clients, schema can be changed
- No heavy workload
- Robustness good, no differing data
- Clients need to know whether to pull or not
- Clients and ZEMIS Ref provide web service
Replication: RabbitMQ Proof of concept

Using an open source message broker

ZEMIS Ref sends messages to exchange which forwards it to queues, clients fetch them, Advanced Message Queuing Protocol

- Each client has its own queue
- Queues have binding keys

- Messages have routing keys
- If keys match, message is sent to queue
RabbitMQ: Pro and Contra

- Easy to set up and use
- Loose coupling
- No heavy workload
- No web service needed
- Not bound to XML - Byte arrays
- Robust

- Against architectural concept $\Rightarrow$ optimal solution?
Summary

- ZEMIS Ref administrates reference data shared among applications
- Decapsulation of ZEMIS Ref and its clients
- Analysed several approaches, prototypes
- One not optimal approach is already in development
- PoC for messaging service was success