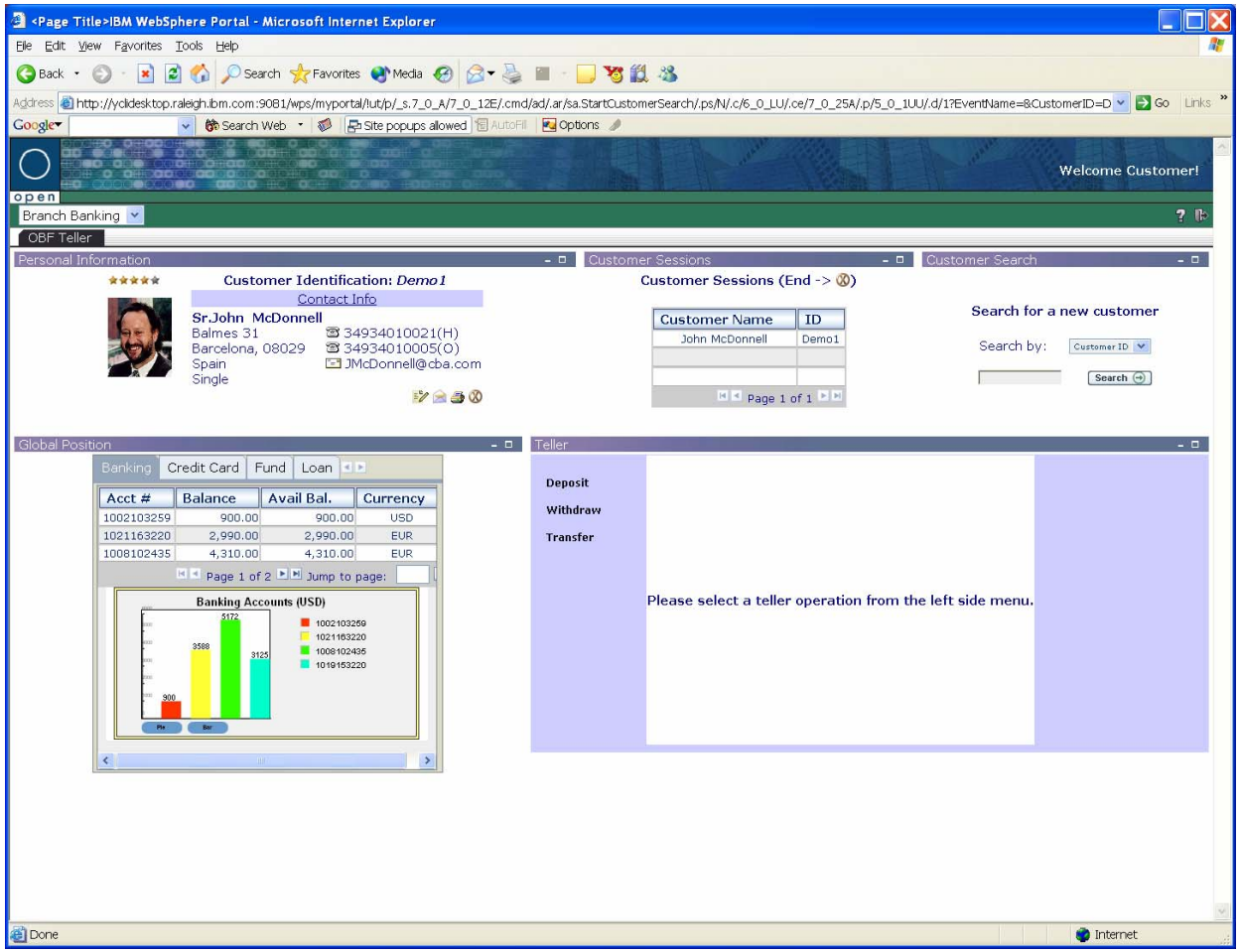
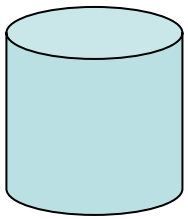
A Programming Framework for Mobilizing Enterprise Applications

Paul Castro, Frederique Giraud, Ravi Konuru, Apratim Purakayastha, Danny Yeh

Insurance Application



Motivation: Rich Client for Mobile Clients

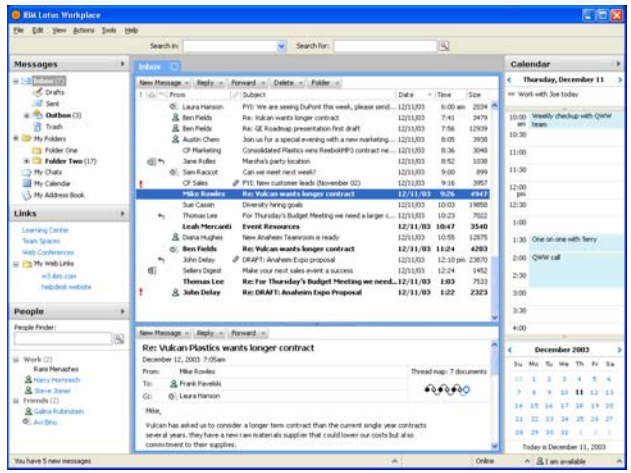


Server-based Application on Provisioning Server

Application Projection

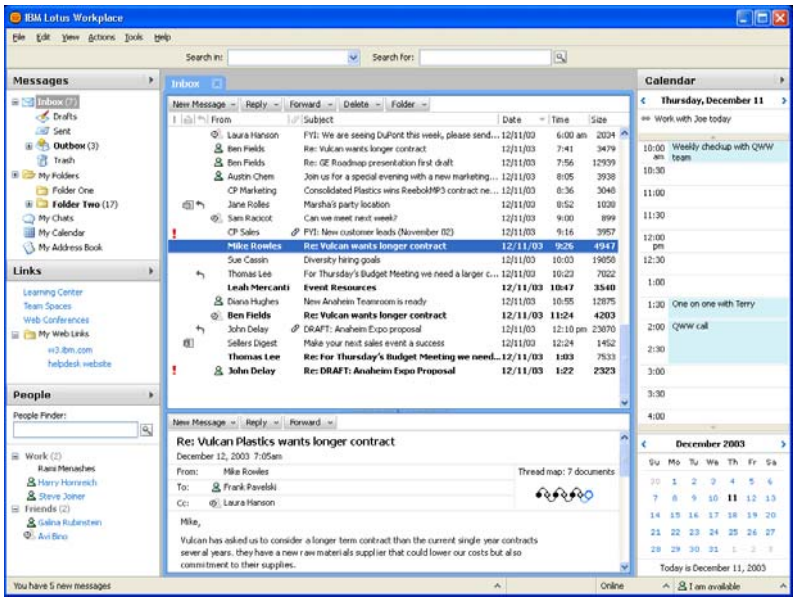
Open standards (WS, OSGI, etc)

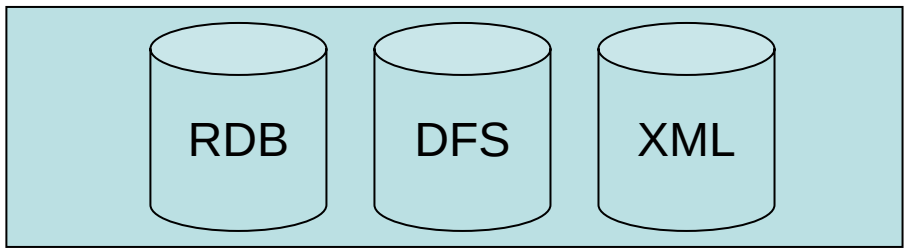
Localized and Tethered Application Components



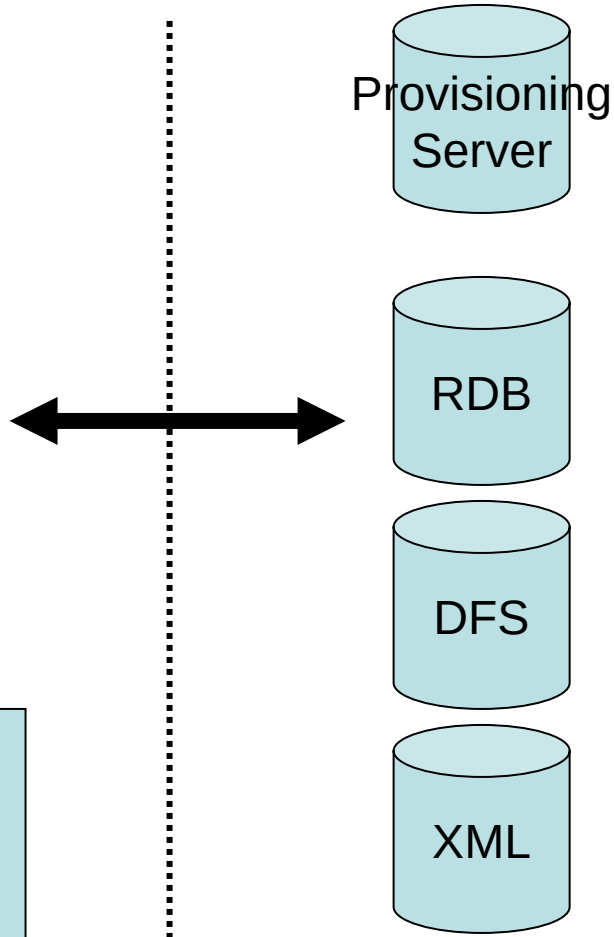
Rich Client

Disconnection in a Mobile Client





Client



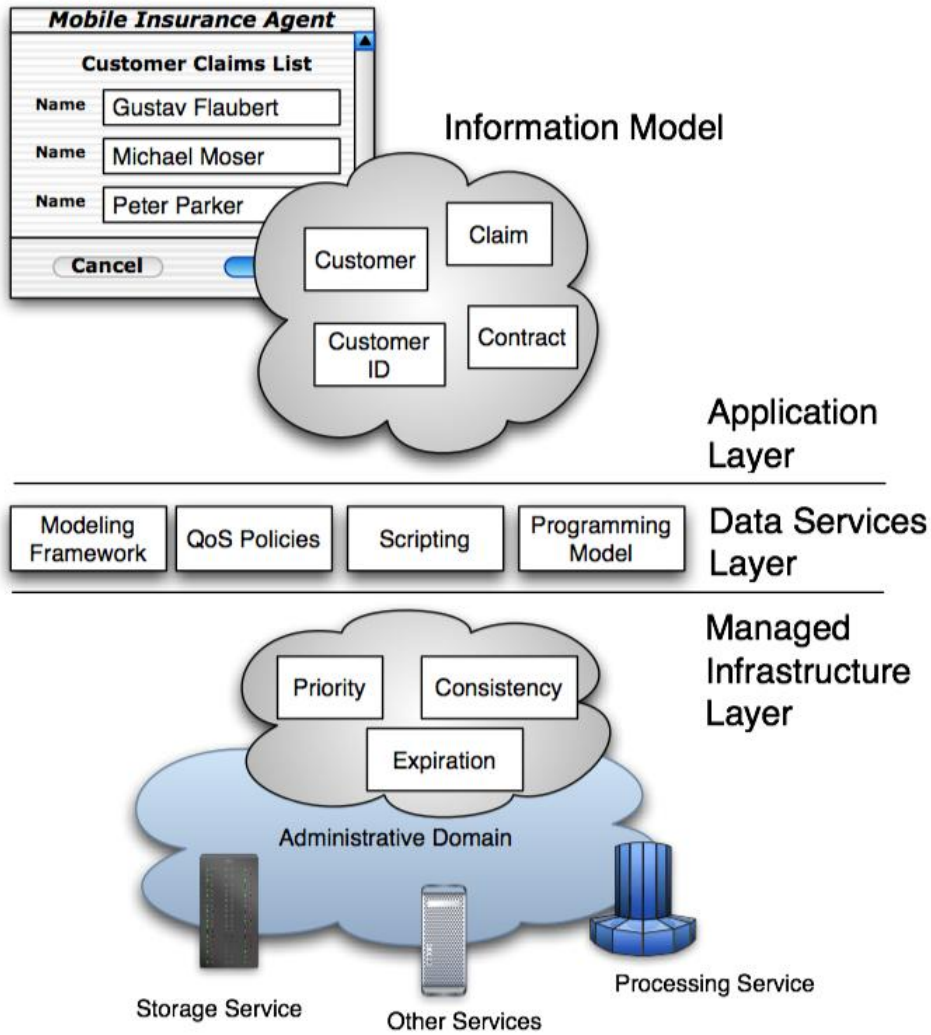
Network Resources

Presentation Overview

- Model-based replication framework
- Service Data Objects Synchronization
- Status and Future Work

Data Services Layer

- Broker between multiple information models and storage models
  - Application-level data requirements
  - System-level QoS
- Support for multiple update semantics
  - Synchronization
  - Messaging
  - Coordination
- Application-level declarative policies
- Applicable to Multiple Client types: e.g., Browser, WCT



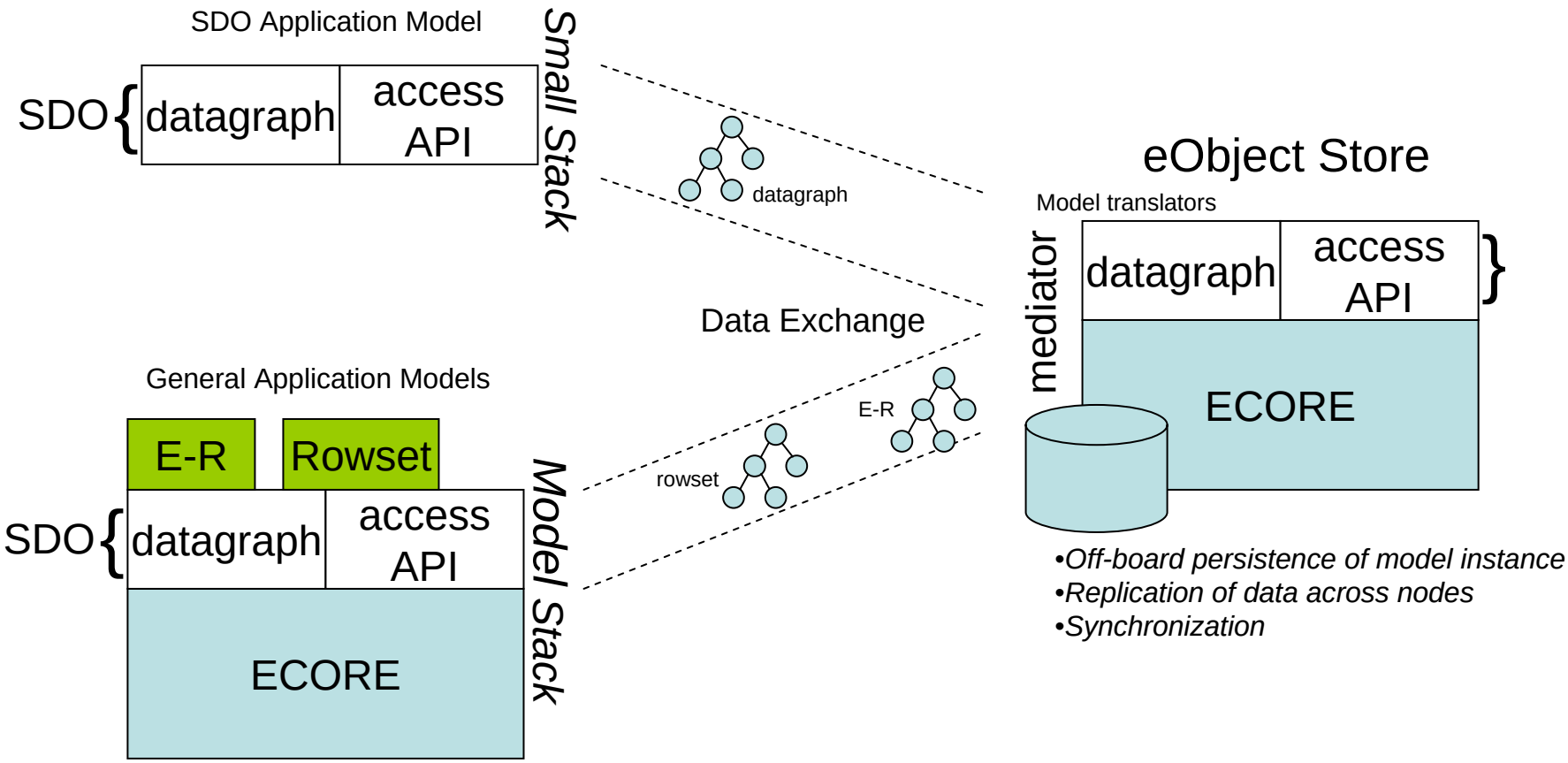
Two Main Goals

- Re-usable infrastructure that can support many different model-based approaches
  - Client and server store abstractions
  - Performance
    - Young codebase
    - Overhead of current sync mechanisims is potentially high for an application design point
- Support for a model-based approach to federation and replication
  - Focus on customer applications that used heterogeneous data sources
  - Easy (enough) programming model

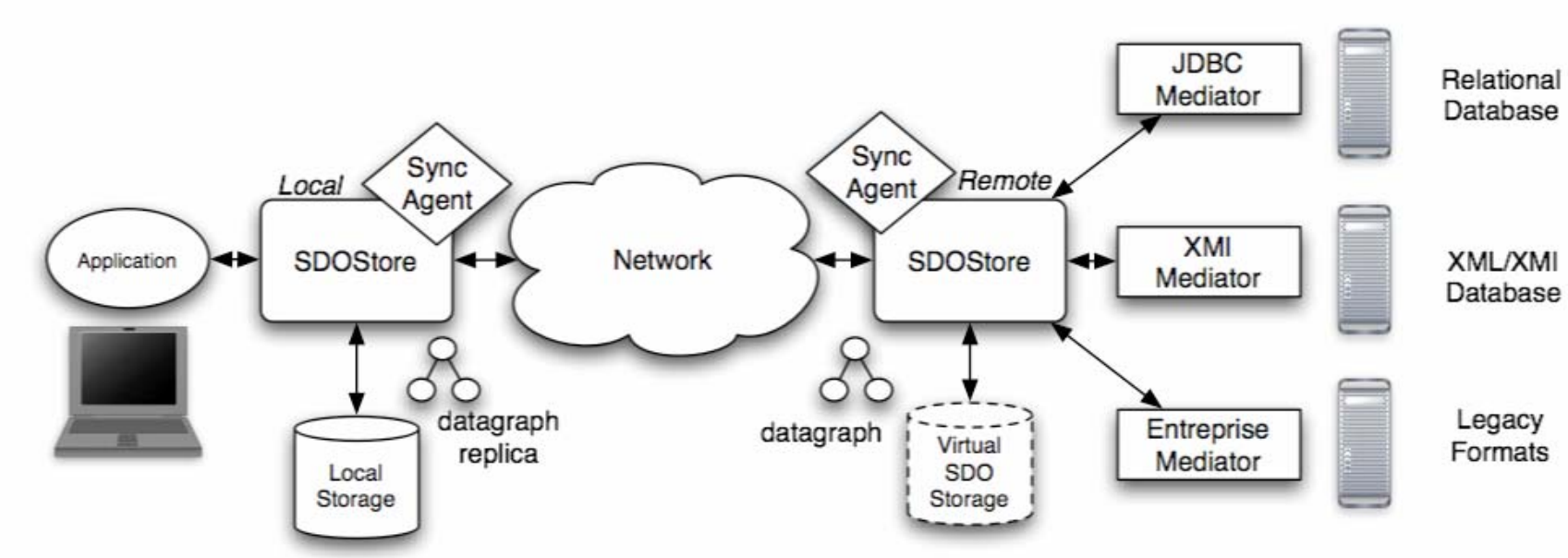
Example Model – Service Data Objects

- Web Services standard first proposed by IBM, BEA
- Object model of data
  - Objects contain typed attributes, references, and lists/sequences
  - Collection of objects form a datagraph
  - Datagraphs have “closure” property
- Standard API defined by SDO specification
  - XPath-like querying of datagraph
  - Java API to manipulate datagraph
  - Creation of SDO datagraphs defined by separate mediator specification
- Implementation of version 1.0 closely tied to Eclipse Modeling Framework
  - Tool for defining models and automatically generation code
  - Meta-models built using EMF/MOF
  - SDO is really just a bit-flip away

Re-usable Model Distribution Stack



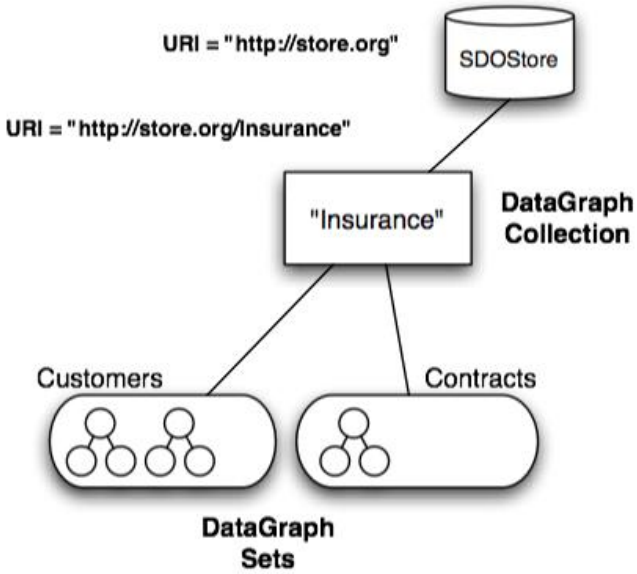
Current Prototype: SDOSync



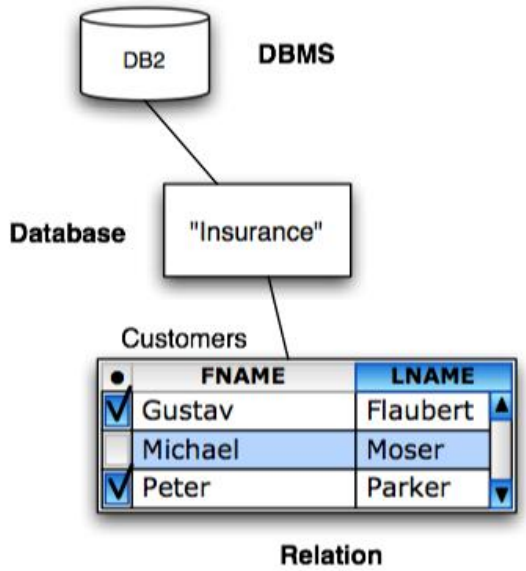
Internal Organization of SDOStore

- Organization of models as SDO datagraphs
  - Datagraph (Model) set
  - Datagraph (Model) collection
- Store and retrieval primitives using XPath-like expressions
- Elementary algebra for simple data manipulation (select, join)

SDOStore

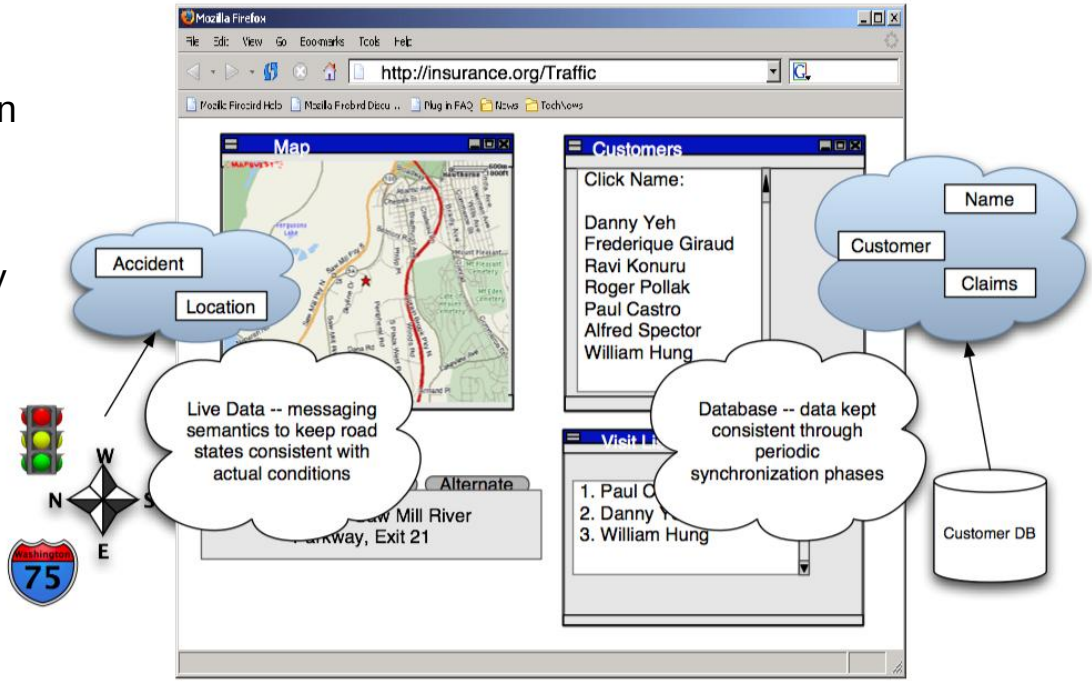


RDBMS



Consistency over Models

- Support for multiple consistency models
  - Traditional optimistic consistency using explicit synchronization sessions
  - Weaker approach to synchronization using messaging semantics
- Declarative consistency policy files
  - Synchronization agent consults policy files to determine consistency requirements
  - Change consistency model based on semantics of the data or the performance requirements of systems
  - Currently using XML formatted file and playing with “Priority” and “Granularity”



Declarative Consistency Policies

```
<syncpolicy>
    <copyset name='custrecords'>
        /CompanyData/CustomerRecords/
    </copyset name='agentinfo'>
    <copyset>
        /CompanyData/AgentInfo/
    </copyset>
    <copylaw role='owner'
                priority=10
                granularity='property'
               name='custrecords'/>
    <copylaw role='assistant'
                priority=1
                granularity='datagraph'
                name='agentinfo'/>
    <acl>
        <role name='owner'>
                <userid>Polly</userid>
                <userid>Anna</userid>
        </role>
        <role name = 'assistant'>
                <userid>Alfred</userid>
        </role>
    </acl>
</syncpolicy>
```

- Policy-based synchronization between SDOStores
- Policy file formatted in XML
- Exploring some consistency degrees of freedom
  - Priority
  - Granularity
  - Still evaluating primitives: perhaps only 2-3 consistency modes needed?
- Point-to-Point and Multipoint synchronization policies

Conclusions

- Status – working SDO Sync prototype for insurance application
- Building better support for declarative consistency policies
- Understanding performance requirements
- Related Work at IBM Watson
  - EJB Sync (preserve transactional semantics for disconnected applications)
  - Live Data (message-based semantics for keeping data synchronized, e.g. soft-state)