Enterprise Systems - Moving to the Cloud

HPTS, Oct 2009

Rainer Brendle
SAP AG

with many thanks to
Shel Finkelstein, Thomas Heinzel, Hui Ding, Dean Jacobs, Kaj van de Loo, Albert Zedlitz, Gunther Liebich and many others

Enterprise Systems - Moving to the Cloud
Conclusion #1

Today we most see data centric applications “in the cloud”

- Mail , IM, collaboration, CRM, social networks, data collection, some analytics etc.

Major cost factor for moving complex process-oriented applications to cloud

- Impedance mismatch between SQL, Programming Languages, XML
- Leads to unmanaged memory and cpu consumption
- Takes away agility in development and the adaptivity of the solution
- Effective verticalization and extensibility concepts missing
- Effective coherent caching and distribution of data

Sizes

Some Facts about data sizes in commercial enterprise applications

- Data changed in a typical transaction ~100KB
- Reference data needed : 10MB - 100MB
  (and data and code need to be together in RAM for the system to scale)

Huge volume of reference data are needed to describe the business rules

- Purchase Order -> Product types, product configuration, customer categories, location of customer, tax rules, pricing rules, location of delivery agent, supplier, accounting rules, availability, ...
  (a seemingly never ending list).

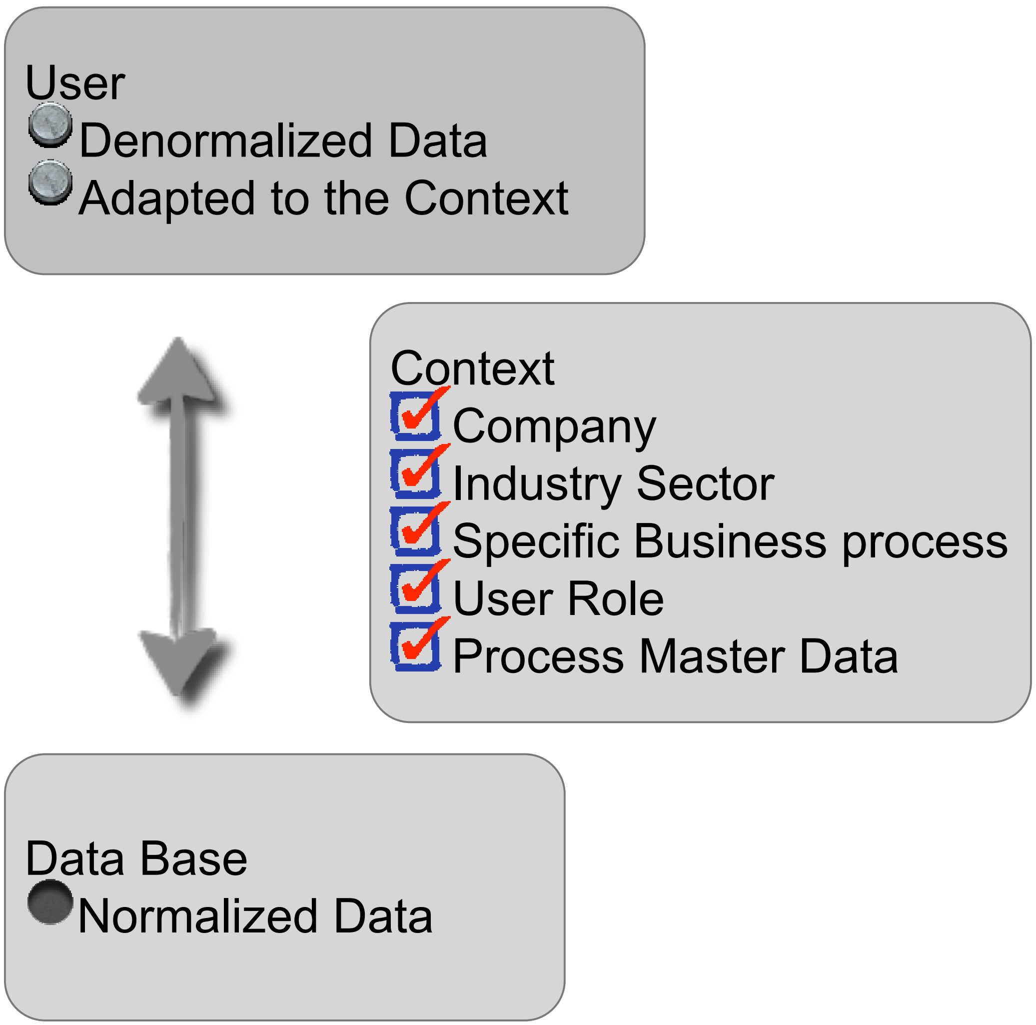
Attributes of Business Objects

- Purchase Orders attributes used in global trade : > 100,000 attributes
- SAP supports about 10,000 of them
- A single company typically uses less then 1,000
  (Customization, verticalization and extensibility is essential)

Reference data are needed everywhere

- Database, Application Servers, Client

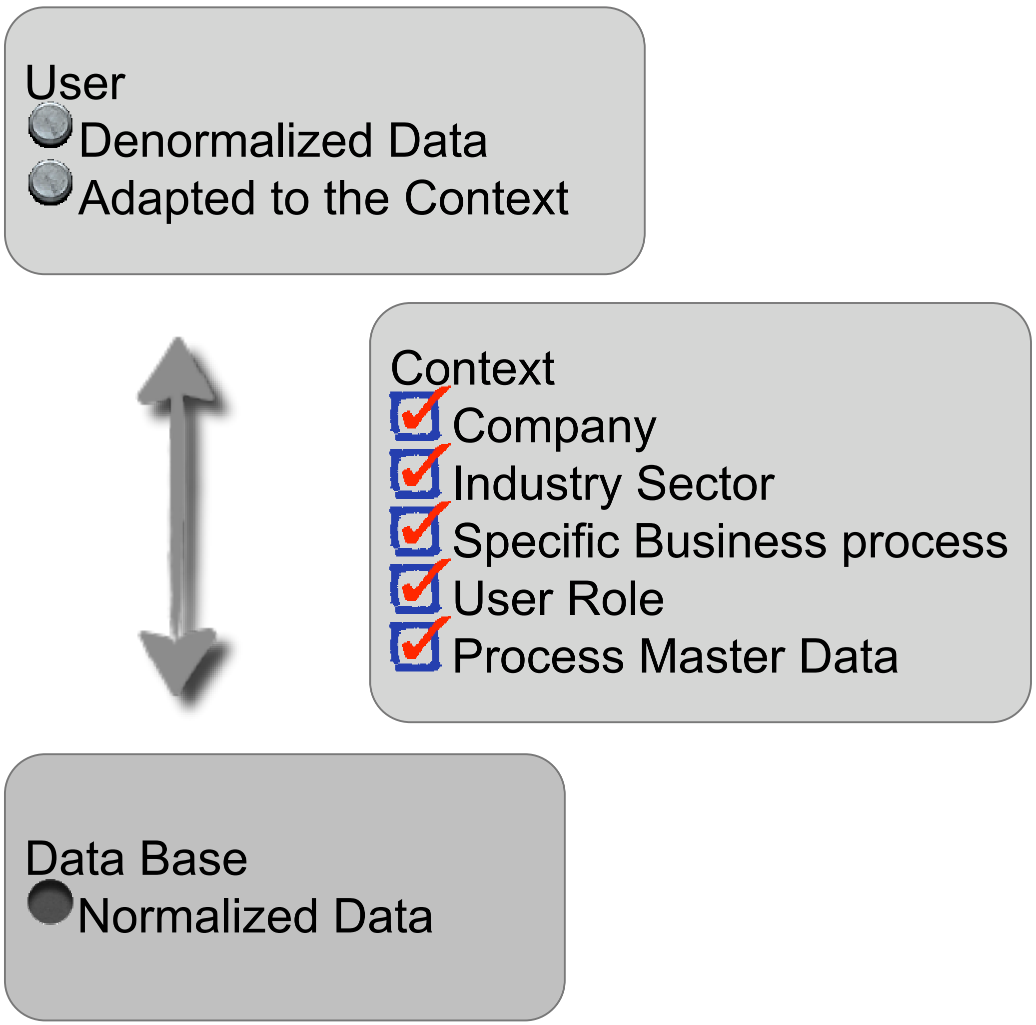
The “Object” Problem and The “Database” Problem



The “Object” problem

- Data get transformed from a normalized database to a user-friendly representation
- Driven by various contextual parameters
- Database -> OR-Mapper -> Configuration, -> Process, ..
- Every layer adds a copy
- Every layer adds maintenance costs
- Unmanaged memory consumption
- OO-style programming forces to materialize the “views” on every layer

The “Object” Problem and The “Database” Problem

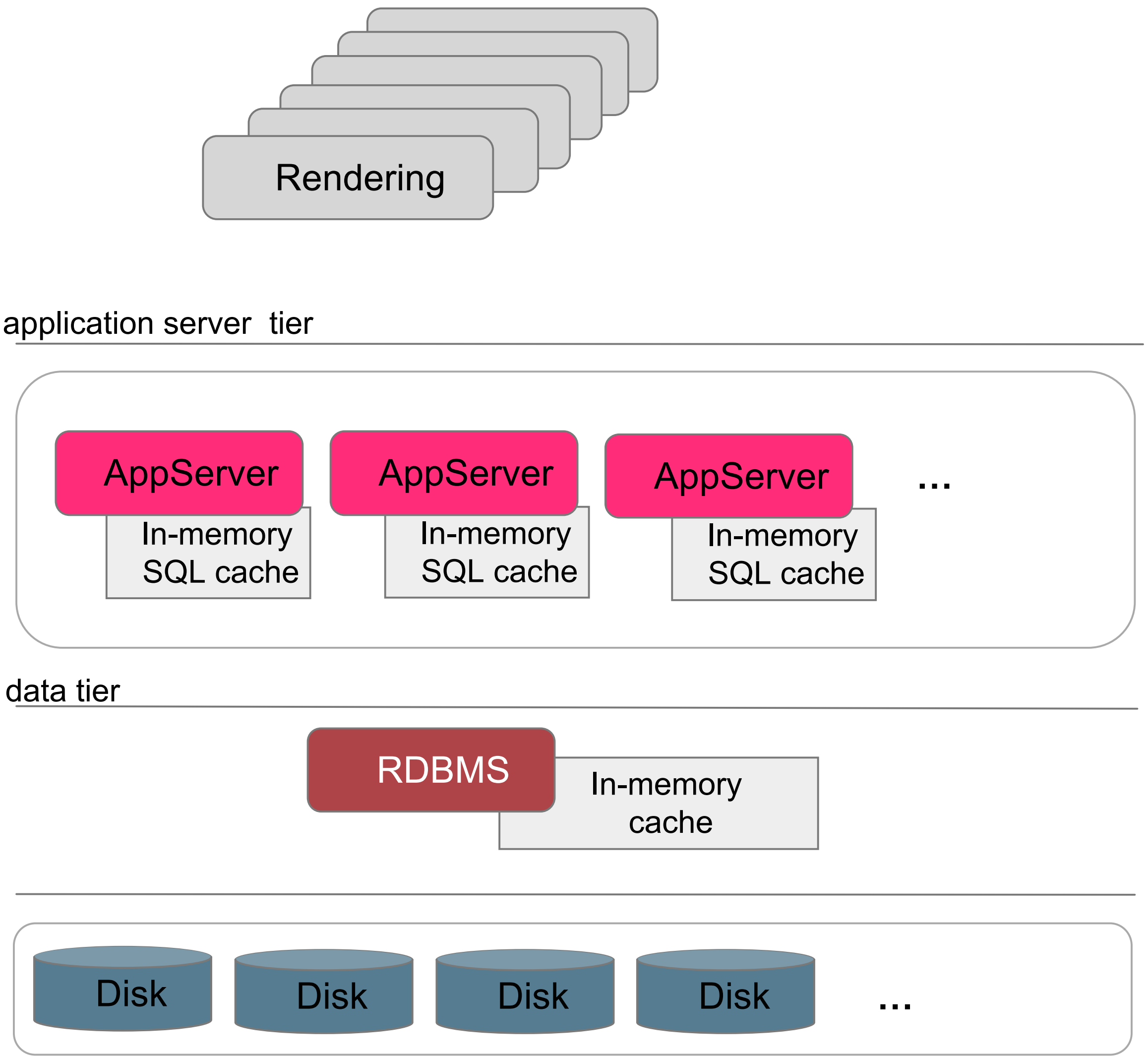


The “Database” problem

- Schemas are not flexible
- Operation costs are dominated by schema management (the “upgrade” problem)
- We need context-specific extension and specialization mechanisms
- Adding attributes at no costs
- Unused attributes at no costs
- We need a better and semantic way to flexibly describe associations between things
- Joins are everywhere, they need to scale
- OR impedance mismatch is a mess

From Servers ...

Today's Scaling Model

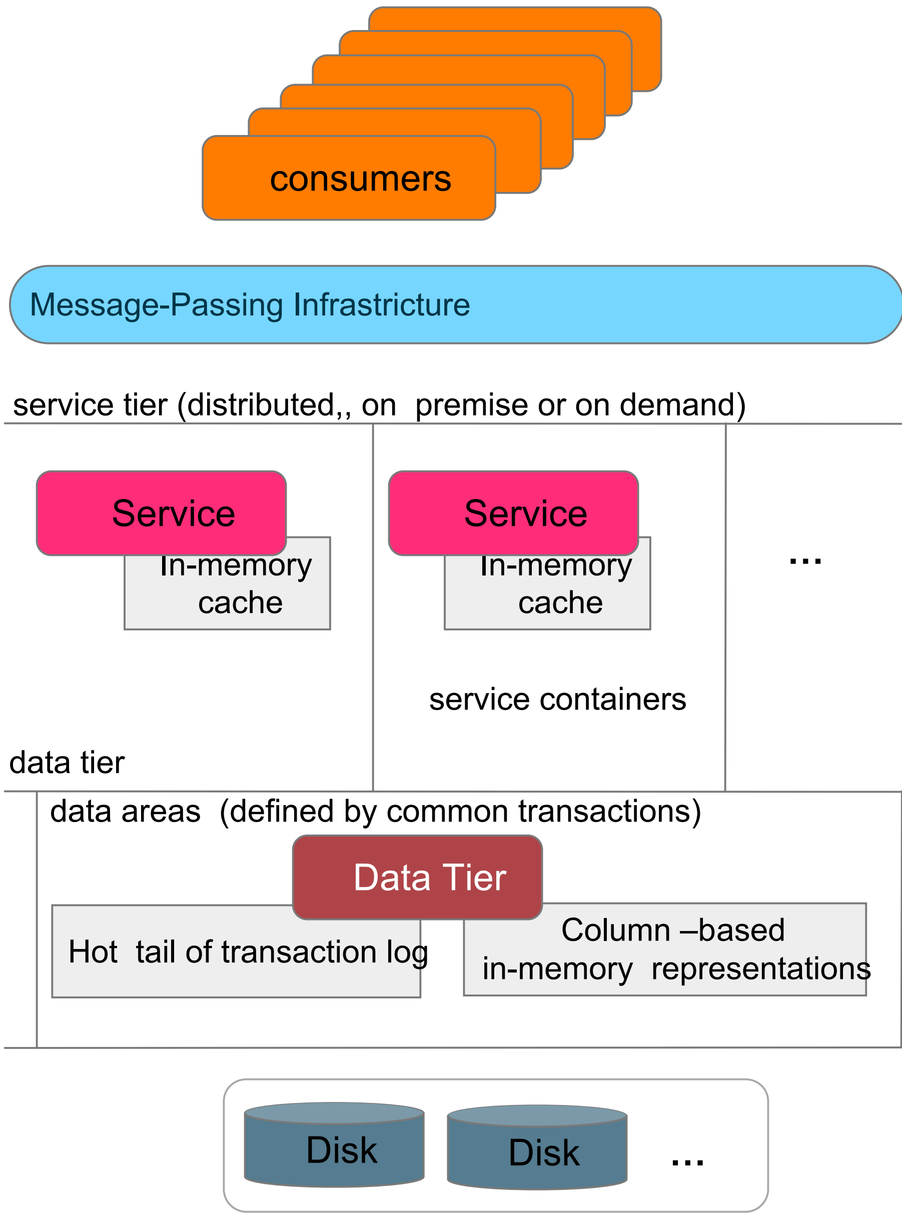


In-Memory hit rate for SQL data access today

80-90 %

>98 %

... to Services



Local Data Availability
with the needed cache coherence
in the format as needed locally

Managing Latency and Addressing
Queueing-up requests, load-balancing,
addressing resources cross firewalls,

Scalable and distributed Hot Data Media
Service should have a cache hit rate close to 100% for object access, Views and Joins on cached data, Data cached in the context of application code, no copying, cache coherence allows to keep also changing data local

Central Hot Data Medium
Data hit rate close to 100%,
Insert-only transactions
search, aggregation, complex joining on data
(classical RDBMS-style access moves to service tier)

Cold Medium
Transactional Logging, Archiving, Data and System Recovery

Scaling Data

Partitioning with today’s and tommorow’s hardware

- No problem to run a single business application module for a whole company on a single box even today, but for sure in near future
  - Typical Modules: Financials, Delivery and Logistics, Manufacturing etc,
- Transactions crossing such modules are rare and can be easily avoided
- Defining partitioning: simply use **tenant + module** = “data area”
- Assuming a transaction log per data area
- In most cases good enough

- Essential
  - Provide data into a distributed landscape
    - With appropriate granularity for each layer
    - Need to support projections (views and joins) dynamically
    - Need to support local caching with consistency

Enterprise Systems - Moving to the Cloud
Conclusion #2

Data access layer, which can cross-cut the various layers of applications

- We tried Java-like object-representations (bad idea), we tried XML (bad idea)
- We better rediscover set theory, logic, functional approaches from SQL
- The data layer must support in-memory caching of data at the various layers where needed
- Late materialization of queries
- Functional approaches (define data and transactions by describing functions)

Scaling Data

A little language experiment : define a class like this:

```
US_Customer {
    from Customer
    select firstName
    select lastName
    where ( countryCode = "USA" )
}
```

- Just a collection of functions. (select, where, ...)
- We can extract subsets/views at runtime

```
my_us_customers = us_cstomers.where('firstName == "Tom").
```

- Materialization is optional.
  Classes just collect functions like select, where, from …

Scaling Data

A little language experiment : define a class like this:

```
US_Customer {
    from Customer
    select firstName
    select lastName
    where ( countryCode = "USA" )
}
```

- Just a collection of functions. (select, where, ...)
- We can extract subsets/views at runtime

```
my_us_customers = us_cstomers.where('firstName == "Tom").
```

- Late materialization, further filtering after materialization.
  Classes just collect functions like select, where, from …
- System can provide data from local caches or from database (or both)
- Cache synchronizations concepts become possible
  (versioning, multi-version/insert-only transaction log, shared data)
- Optimization, which “filter” function to apply where and when

Scaling Data

Rethink SQL
(from a distinct datastore layer to application programming)

Basically set theory with propositions (boolean functions, first-order logic)

Can’t we use these concepts to scale-out business data

- also outside the database? As a basis for a distributed service landscape?
- better then with Java-like objects

Scaling Data

Rethink SQL
(from a distinct datastore layer to application programming)

Business data typically describe Business Events

- Stable in time, immutable
- Immutable data are constant (and by this distributable)
- Constants can be described as functions (surprising, but true!)
- Functional approaches become possible
  (parallel execution, late materialization)

Scaling Data

Rethink SQL
(from a distinct datastore layer to application programming)

Context-specific data definition?

```
        Customer {
             has firstName
             has lastName
         }

context UtitiliesIndustry:

         Customer {
              has meterID
         }
```

Enterprise Systems - Moving to the Cloud Conclusion #3

Overcoming RPC

- We are still in an RPC-like world regarding Web Services
- Fixed interfaces, client cannot really request or express, what he wants
- Instead:
  - send filter and transformation functions to read data
  - send flexible events and notifications for transactions
- We would have 100,000 interfaces, if doing this with an RPC-like style

Services must be usable, not reusable

- Client-driven, client must express, what is needed
- Client defines its own data views
- Client-side view definition = adding another “function” to a data feed definition
- Well-defined state transfer and data lifecycle
  - Feeds
  - Conflict-free transactions
  - Express business events and notifications

Enterprise Systems - Moving to the Cloud
Conclusion #3

Services to be based on asynchronous message-passing

- Actor model
- History: IMS DC, SAP, Tuxedo, CICS, TPF (IBM, Sabre, Amadeus, Visa)
- The equivalent in a cloud is a **distributed messaging infrastructure**
  - addressing distributed resources
  - load-balancing
  - crossing firewalls (on demand/on premise, cross company)
  - Active, AMQP opens this world up

Enterprise Systems - Moving to the Cloud
Conclusions (repeated)

Today we most see data centric applications "in the cloud"

Major cost factor for moving complex process-oriented applications to cloud

- Impedance mismatch between SQL, Programming Languages, XML

Data access layer, which can cross-cut the various layers of applications

- We rediscover set theory, logic, functional approaches from SQL
- The data layer must support in-memory caching of data at the various layers where needed
- Late materialization of queries
- Functional approaches (define data and transactions by describing functions)

Services must be useable - not reusable

- Client expresses its intention

Services to be based on asynchronous message passing

- from TP monitors to distributed messaging systems