Abstract

This practitioners’ report describes Credit Suisse’s new client-centric price management platform, with the aim of replacing over 20 heterogeneous older applications with an integrated solution. We plan to improve business agility and to enable new ways of creating, controlling, and governing prices across all the bank’s businesses.

We have decided to employ the federated architecture pattern as an enabler for a stepwise inside-out renewal. This approach leads to a highly distributed platform, with customizable components for specific business needs. As a result, the platform is designed and managed as a software product line.

A software product line leverages reuse potential and helps to enforce required commonalities—but a product line also requires a dedicated design and management approach. We’ll present our approach, which is based on model-driven engineering and domain-driven design, and will show how a clean separation and customizability of both domain-dependent and domain-independent aspects are supported.
Enable innovative price models by building an integrated IT platform for client-centric price management

- Brokerage
- Payments
- Custody
- ...  

- CRM
- Pricing
- Regulatory Support
- ...  

But: The matrix of core banking processes and client-centric banking processes must be carefully organized!
The disconnected and highly heterogenous structure of legacy pricing functionality has to be overcome.

- **Standard Tariffs**: 40 services; 200 variants
- **Special Terms & Tariffs**: 50 product groups; 130 products
- **Integrated Pricing Platform**
  - 20 Product Areas
  - 500 revenue-related types of posting records; 90 consignments

*All numbers approximate*

But: A monolithic, centralized architecture is not an option! Efficient Business/IT rollouts have to be enabled; product area specialities and operational independence have to facilitated.

While more complex, a Federated Architecture is required to balance client-centric, integrated price management vs. feeder variety/independence and rollout complexity.

But: The Federated Architecture implies a high degree of componentization, resulting in the risk of high SW cost and conceptual/operational fractionation.
The Software Product Line approach provides the tools to enforce required commonalities and to leverage reuse potential

- Core Asset Development
  - Developing components and accompanying artifacts
- Product Development
  - Developing customized products by composing customized components
- Organizational and Technical Management
  - Keeping everything in check

But: Effective reuse can only be achieved by a careful and differentiated analysis of all conceptual and technical aspects of the architecture.

Example: The value of a proper and properly executed global IT reuse strategy

Strategy: "Evolutionary"

- \( x \) mCHF
- 6 countries + CH
- one common code base for all locations & CH

2006-2008

Source: IT Plan; classification of projects not validated w/ providers

Strategy: "Disruptive"

- \( 3^x \) mCHF
- 1+3+1 countries + CH
- four separate and irreconcilable code bases

2008-today

Reorganization / Strategy change
The XPP reuse strategy is based on differentiated approaches to price model management and price calculation.

Applications:
- Service & Contract Management
- Product & STT Management
- Price Calculation

Business Objects:
- ServiceArea, ServiceGroup, ServiceAssignment (Contract)
- ProductCombinations, Products, Tariffs, STTs

getTariff

- Universal
- Feeder-specific

The common concepts for structuring integrated price models across all product areas:

SMWB:
- Investment Services
- Discretionary Mandates
- Benchmark oriented Mandates
- ExclusiveSelection
- Brokerage

TMWB:
- Equities, Warrants, etc.
- 1.6% of Transaction Value

XPP BOM (simplified):

- Service Group
- Service Category
- Service Area
- Service
- Product Area
- Product
- Tariff

"John Doe" holds an Exclusive Selection Mandate

"John Doe" holds a Special Tariff Terms of "John Doe" (if applicable)

0% (no Transaction Fees for Exclusive Selection)
Structural Variability: Generic price model structures can be specialized for product areas/feeders

If we support variety in the data models – how to efficiently implement the business layer for managing the data?

Compositional CRUD: A balanced approach for complex data management

- Arbitrarily complex graph transformations can be represented as composition of basic transformations
- Compositional CRUD: The client composes complex operations from primitive operations. The composition can be executed with one service call
The Economics of MDE

<table>
<thead>
<tr>
<th></th>
<th>XPP TMWB Brokerage</th>
<th>FrontNet FNAPPL</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDM</td>
<td>27</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>LDM-H</td>
<td>50</td>
<td>149</td>
<td></td>
</tr>
</tbody>
</table>

**Generated Artifacts**

<table>
<thead>
<tr>
<th></th>
<th>kLoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransferObjects</td>
<td>11</td>
</tr>
<tr>
<td>PersistentObjects</td>
<td>2</td>
</tr>
<tr>
<td>Services</td>
<td>6</td>
</tr>
</tbody>
</table>

**Generators and Framework**

<table>
<thead>
<tr>
<th></th>
<th>kLoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2M</td>
<td>2.5</td>
</tr>
<tr>
<td>TransferObjects</td>
<td>1.5</td>
</tr>
<tr>
<td>PersistentObjects</td>
<td>0.1</td>
</tr>
<tr>
<td>Services</td>
<td>0.5</td>
</tr>
<tr>
<td>Runtime Framework</td>
<td>3*</td>
</tr>
</tbody>
</table>

* further refinements steps pending
Step-wise refinement and variability of pricing logic

XPP Core Domain Logic

+ Specific Rules Formalism

+ Brokerage Specific Logic

+ Brokerage Pricable

Production Infrastructure (Persistence Unit, Oracle, JMS/MQ)

Test Infrastructure (Persistence Unit, JavaDB, Test Orders in XML)

Brokerage PRICE getPrice() Service

Unit Test

Behavioural Variability: Logic of Price calculation can be specialized as well by applying "Stacked Domain Driven Design"

Generic Domain Model

Brokerage Domain Model

Payments Domain Model

Custody Domain Model

Brokerage Infrastructure Binding

Payments Infrastructure Binding

Custody Infrastructure Binding
Distribution of Cost in XPP Architectural Release 1

<table>
<thead>
<tr>
<th>XPP Platform Components</th>
<th>Specific for first Feeder</th>
<th>Reusable Base Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMWB</td>
<td>10%</td>
<td>90%</td>
</tr>
<tr>
<td>TMWB</td>
<td>20%</td>
<td>80%</td>
</tr>
<tr>
<td>PRICE</td>
<td>60%</td>
<td>40%</td>
</tr>
<tr>
<td>Feeder Integration</td>
<td>70%</td>
<td>30%</td>
</tr>
<tr>
<td>Price Model Migration</td>
<td>80%</td>
<td>20%</td>
</tr>
</tbody>
</table>

45% (33% of platform) 55% (66% of platform)

Indicative Reuse Base plus Project Cost & Savings

The more feeders integrated...
- the higher the base of reusability
- the lower the cost of every subsequent feeder (higher cumulated savings)
- the higher the relative impact of integration and migration cost ('renewal')
Federated Architecture and Software Product Line: The right strategies to achieve client-centric pricing

- Integrated, client-centric pricing only feasible with
  - efficient rollout/migration paths
  - proper balancing of harmonization of pricing concepts vs. business variability
  - proper balancing of pricing integration vs. operational independence

- A federated architecture enables balanced designs

- The software product line approach is highly suitable to implement a federated architecture

- Differently characterized parts of the architecture require different reuse/variability concepts

- Every senior developer trained to be able to *technically* setup a new feeder implementation within one day

Content

- The Extended Pricing Platform and it’s Federated Architecture

- Federated Architecture in the wider Context of Credit Suisse
Creating the Federated Architecture (FA) Vision

analyzing the requirements and answering questions

- Why is it so difficult to implement a comprehensive system streamlining requirements from many business areas? Why could COTS (replacement approach) not be integrated?
- Are there other ways to solve such problems than a central, monolithic approach which dramatically reduces agility? (even the agility to find a solution)
- How to reduce the conceptual and technical complexity?

Federated Architecture Pattern explained

comprehensive models which can be enhanced to local needs

- The Federated Architecture Pattern describes an approach to enterprise architecture that allows interoperability, control of and information sharing between semi-autonomous de-centrally organized lines of business and information systems.

  The pattern emphasizes on organizations and systems which require a controlled sharing of information and behavior among autonomous components and is especially useful for very large organizations and information systems.

- The Federated Architecture Foundation (FAF) defines what has to be shared and centrally controlled. It defines common concepts and reusable components, standards and principles, a comprehensive BOM which can be enhanced by distributed components of different business areas to their local needs. This allows for central control of defined aspects of behavior and syndication of information.

- Goals

  Reduce complexity: Complexity is usually the result of too many local particularities which are tried to be managed in a central, one fits all approach. This is the reason why highest possible autonomy shall be given to the different cooperating components.

  Higher agility: The result is a higher degree of flexibility and a larger solution space — which at the end means, taking local particularities seriously and solve local problems locally whenever possible.

  Implementation: To be able to manage different lifecycles, migration plans and continuous enhancements and changes of functionality of the participating components, Federated Architecture should be developed as a product line, with a common technical architecture empowered by MDE.

*for a more detailed description of the pattern, see http://en.wikipedia.org/wiki/Federated_Architecture
Federated Architecture Pattern

a strategy pattern enabling the evolutionary-coexistent migration process*

- The pattern solves the functional and non-functional requirements as well as IT architecture standards. This is the reason, the FA pattern is regarded as a key architecture strategy pattern.

*For more details about the approach see Managed Evolution by Stephan Murer, et al.

Federated Architecture applied

Credit-Suisse is in the midst of an evolutionary-coexistent, stepwise & risk minimized migration process

- Service & Contract are delivering the syndicated comprehensive view.

- Especially for the federated core banking objects: master data like product but also transactional objects like orders & positions (accounts) the FA pattern looks promising.
Federated Enterprise Architecture applied

With the help of the FA pattern, IT enables efficiently the harmonization of requirements from different business areas. IT efficiency is the value proposition of IT architecture for the business.

Product definition, order execution and booking are product area specific & vertical (federated) but service & contract management as well as position keeping expand many business and product areas (asset classes) are client centric & horizontal (syndicated) processes and views.

Appendix
The stepwise refinement of CRUD to Compositional CRUD w/ Historization/Versioning and Change Management

- Isolated Objects
- Object References
- Objects w/ Components
- Managed Objects
- Versioned Objects
- Specific Application Policies

- simple objects w/ primitive properties
- unidirectional references as special properties
- special containment-references
- designated "changeset" objects, that control modifiability of owned objects
- designated versioned/version objects

Available MDE tools commonly work on model snapshots. But DB change management requires knowledge about deltas!