The San Francisco Framework for Business Applications

Prof. Dr. U. Aßmann
Chair for Software Engineering
Faculty of Informatics
Dresden University of Technology
09-0.1, 12/31/09
San Francisco – Obligatory Literature

- K.A. Bohrer: Architecture of the San Francisco frameworks
San Francisco – Secondary Literature

- IBM SanFrancisco Documentation Entry
What is San Francisco (SF)?

- Business framework of IBM, to support the building of business applications
  - started in March 1995, initial release Aug 1997
- Arranged as layered frameworks
- Supporting distributed applications
- Based on business-specific Design Patterns
- Design goals
  - flexibility by using object-oriented framework technology
  - maximal reuse
  - isolation from underlying technology
  - focus on the core, provide the common tasks of every business application
  - rapidly building quality applications
  - integration with existing systems
San Francisco Architecture (1)

Three layers:

- **bottom**: *Foundation* provides infrastructure and services (transactions, collections, administration, conflict control, installation), hides differences in underlying technology

- **middle**: *Common Business Objects* provides implementations of business objects that are common to more than one domain

- **top**: *Core Business Processes* provides business objects and default business logic for selected vertical domains (accounts receivable, accounts payable, general ledger, order management warehouse management)
San Francisco Architecture (2)

- Java Virtual Machine
- Foundation
- Common Business Objects
- Core Business Processes
- General Ledger
- Order Mgmt
- Warehouse Mgmt
- AR/AP Ledger
- Common Functions Financial Interface (CFFI)
- Application
- Application
- Application
- Application
- Customer Solutions

IBM San Francisco
Predefined Business Objects (from the Domain Model)

- General business objects
  - Address, currency
  - Company
  - Business partner, customer
  - Decimal structure of numbers, number series generator
  - Document location
  - Fiscal calendar
  - Initials
  - Natural calendar
  - Payment method and payment terms
  - Unit of measure

- Financial business objects
  - Money, account, currency gain, loss account

- Generalized mechanisms
  - Cached balances
  - Classification
  - Keys and Keyables
Component Model of SF: User-Defined *Entities*

- **Entities**: standard components in SF
  - *materials*, perhaps persistent
  - Have global identifiers (*handles*, *guids*)
  - Created via factories, entered into *containers*
  - Split into interface class and implementation class
  - Dynamically extensible!!

- Entities are similar to *Java Entity Beans*.
  - Hence, IBM started a move to port onto EJB, but this was very difficult

- **Functions**:
  - constructor (factory method). Calls a global factory
  - initialize
  - getters and setters
  - set ownership of an entity (to an entity container)
  - destroy
  - `externalizeToStream`
  - `internalizeFromStream`

- **Global functions**:
  - begin, commit, rollback transaction
  - Manage *work area* for a thread
Business Processes

- Common Function Financial Interface (CFFI): common functionality used by other business processes
- Warehouse management
  - Stock movements
  - Quality control
- Order management (sales, purchase)
  - Order data interchange planning
  - Pricing, discounts, order acknowledgment
- Accounts payable (AP), Accounts receivable (AR)
  - Payment process
  - Business task transfer to other partners
- General ledger
  - Journaling (creating, validating, maintaining journals)
  - Closing at the end of a financial year
Classes can be marked as *extension points*
  - Naming scheme `E<number>_<name>`

Business objects are extensible by *subclassing* (white-box extension)

Subclasses of class `PropertyContainer` are extensible via a special Design Pattern
  - New attributes (properties) can be added dynamically, without recompilation. Access works via hash tables

Policy classes implement business rules
  - *Strategy* (TemplateClass) as extension points
  - *ChainOfResponsibility* as extension points (for multiple policy objects and multiple business rules), e.g., for specific rules of product, system, company, globally
  - *Composite* as extension points: Policies may be added that search for policies (higher-order policies) in composite data structures

Dynamic identifiers for extending value ranges of business value domains
Lifecycle of Business Objects (Business Workflow, Process)

► A business workflow in SanFrancisco is described by an *extensible* statechart
  - However, in the form of a state transition *and* decision table
  - The table rows contain conditions and actions (CA-Rules) and change the state of the process

► The statechart can be extended dynamically with new paths
  - As an action, a transition can extend the statechart (or shrink it)
San Francisco Design Patterns (1)

- San Francisco uses both GOF-Pattern and new business-related Design Patterns
- Special patterns developed that meet particular problems of business applications
  - analyzing typical business applications and developing generic solutions for recurring problems
  - encourage object-oriented implementation of business software
  - several patterns for several aspects of business tasks
SF Design Patterns (2)

Foundational Patterns:
- Class Replacement
- Special Class Factory
- Property Container (extensible class)
- Business Process Command

Process Patterns:
- Cached Aggregate
- Keyed Attribute Retrieval
- List Generation

Behavioral Patterns:
- Simple Policy
- Chain of Responsibility-Driven Policy
- Token-Driven Policy

Structural Patterns:
- Controller
- Key/Keyable
- Generic Interface

Dynamic Behavioral Patterns:
- Extensible Item
- Hierarchical Extensible Item
- Business Entity Lifecycle
- Hierarchy Information
- Decoupled Processes
Selected SF Patterns: Class Replacement

- **Intent**: change the behavior without changing the class or application logic. Provides a kind of *super factory*, a factory delivering factories

- **Motivation**: replace provided business objects with others that have been tailored for a specific application

- **Related Patterns**: Abstract Factory and Factory Method

```
BaseFactory

DomainClassFactory
  creates

SpecializedDomainClassFactory
  creates after class replacement

DomainClass

SpecializedDomainClass
```
Selected Patterns: Property Container

- **Intent:** dynamically extend an instance of a business object with new properties (attributes)
- **Motivation:** adding dynamically new data, properties or capabilities to specific instances of business objects
- **Related Patterns:** Chain of Responsibility, Controller
**Selected Patterns: Business Process Command**

- **Intent:** a logical business object is implemented as multiple physical objects and support one business process
- **Motivation:** encapsulating a business process (a *tool*) in a command, thus a logical object combines a group of physical objects
- **Related Patterns:** Command, Template Method, Facade
Selected Patterns: Simple Policy

- **Intent:** encapsulate business rule as a set of methods in an object, make them interchangeable and produce independence from affected business objects
- **Motivation:** different versions of an algorithm are required dependent on the specific situation in a company
- **Related Patterns:** Simple Policy is a Strategy. Additionally, the strategy method implements a method in the domain business objects with the same name (method factoring). Hence, the BO delegates the computation of the business rule to the strategy
Selected Patterns: Chain-Of-Responsibility-Policy

- **Intent:** encapsulate business rule(s) as a chain-of-responsibility
- **Motivation:** many rules are available for a business case and must be exchanged dynamically.
- **Related Patterns:** A typical 1-TH-pattern. COR-Policy is a Chain, combined with a Strategy. The Chain is searched for appropriate rules that apply to the current state of business.
  - Search order can be changed by higher-order policies
What Have We Learned?

► Big business frameworks are structured according to the principles of variability and extensibility we have studied in the course.
► IBM San Francisco manages extension points and types them with certain framework hook patterns, e.g., Strategy/Policy, or Chain.
The End