22. The San Francisco (SF) Framework for Business Applications

Prof. Dr. U. Aßmann

Chair for Software Engineering

Faculty of Informatics

Dresden University of Technology

12-1.0, 1/12/13

1) Architecture of SF
2) Extensibility Mechanisms
3) Special SF Patterns

San Francisco – Obligatory Literature

- K.A. Bohrer: Architecture of the San Francisco frameworks
  http://researchweb.watson.ibm.com/journal/sj/372/bohrer.html

San Francisco – Non-Obl. Literature

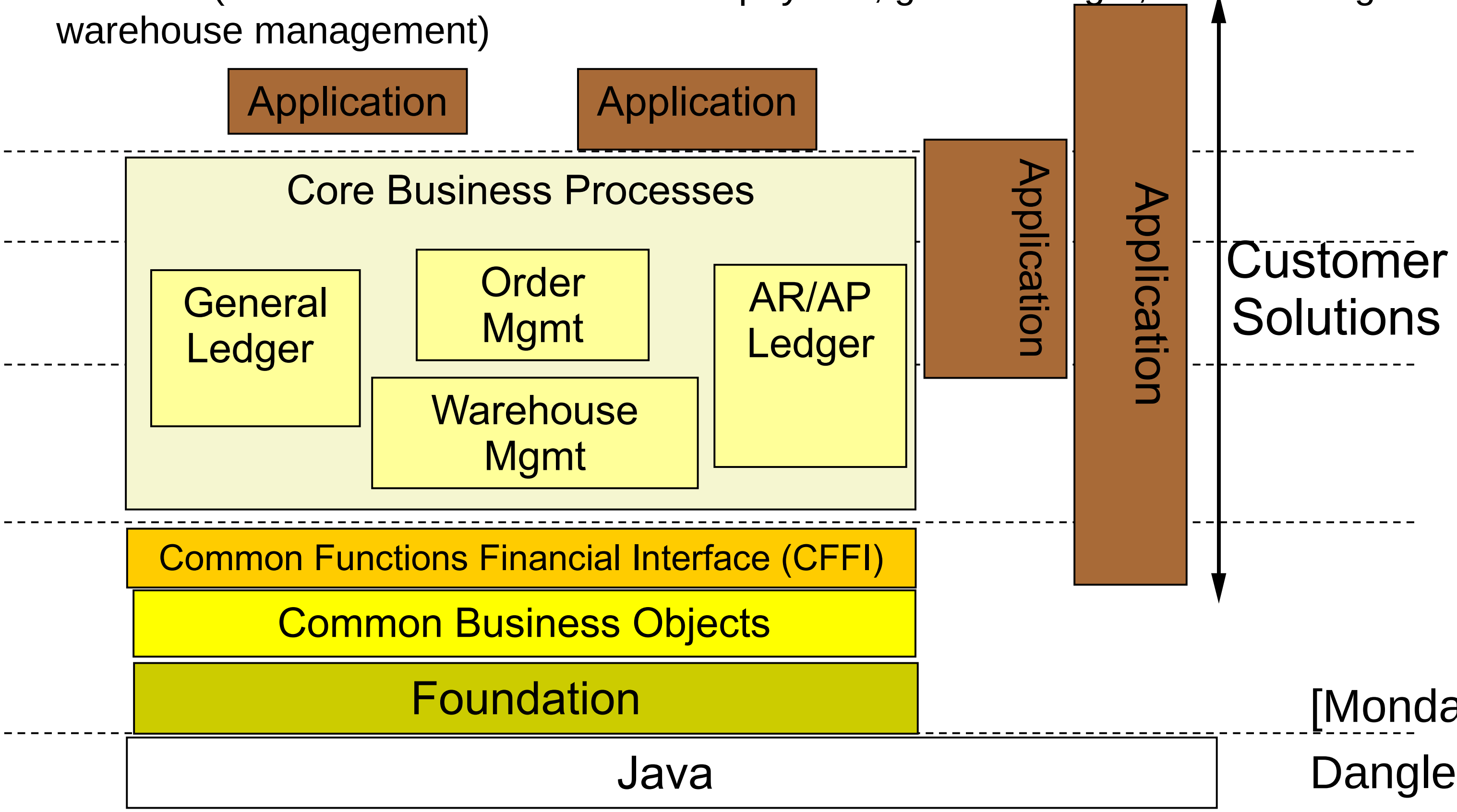
- P. Monday, J. Carey, M. Dangler. SanFrancisco Component Framework: an introduction. Addison-Wesley, 2000. Overview on San Francisco and its layered architecture.
- J. Carey et al.: SanFrancisco Design Patterns: blueprints for business software. Addison-Wesley, 2000.
- Carey, Carlson, "Framework Process Patterns: Lessons Learned Developing Application Frameworks", Addison-Wesley, 2002
- Carey, Carlson, Graser, "SanFrancisco Design Patterns: Blueprints for Business Patterns", Addison-Wesley, 2000.
- IBM SanFrancisco Documentation Entry http://csiserv01.centerprise.com/techdoc/SF/doc_en/ibmsf.sf.FS_DocumentationEntry.html

What is San Francisco (SF)?

- Business framework of IBM, to support the building of business applications
  - started in March 1995, initial release Aug 1997, stopped in 1999
- Arranged as layered frameworks
    - Supporting distributed applications
- Based on business-specific Design Patterns
- Design goals
  - flexibility by using object-oriented framework technology
  - Dynamic extensibility
  - 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

- **Foundation:** infrastructure and services (transactions, collections, administration, conflict control, installation), hides differences in underlying technology
- **Common Business Objects:** implementations of business objects that are common to more than one domain
- **Core Business Processes:** business objects and default business logic for selected vertical domains (accounts receivable/accounts payable, general ledger, order management warehouse management)



[Monday, Carey, Dangler]

Common Business Objects (from the Domain Model)

- General business objects:
    - Value objects: Address, currency, natural calendar
  - Company
  - Business partner, customer
  - Decimal structure of numbers, number series generator
  - Document location
  - Fiscal calendar
  - Initials
  - Payment method and payment terms
  - Unit of measure
- Financial business objects
  - Value objects: Money, currency gain
  - Account, loss account
- Generalized mechanisms
  - Cached balances
  - Classification
  - Keys and Keyables

Component Model of SF: User-Defined *Entity* (Dynamic Classes)

- **Entities**: ***Dynamically extensible components*** in SF
  - *materials*, also persistent
  - with global identifiers (*handles, guids)*
    - Created via factories, entered into *containers*
    - Split into interface class and implementation class
- Entities are similar to *Java Entity Beans.*
    - Hence, IBM started a move to port onto EJB, but this was very difficult

- Standard 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

Core Business Processes

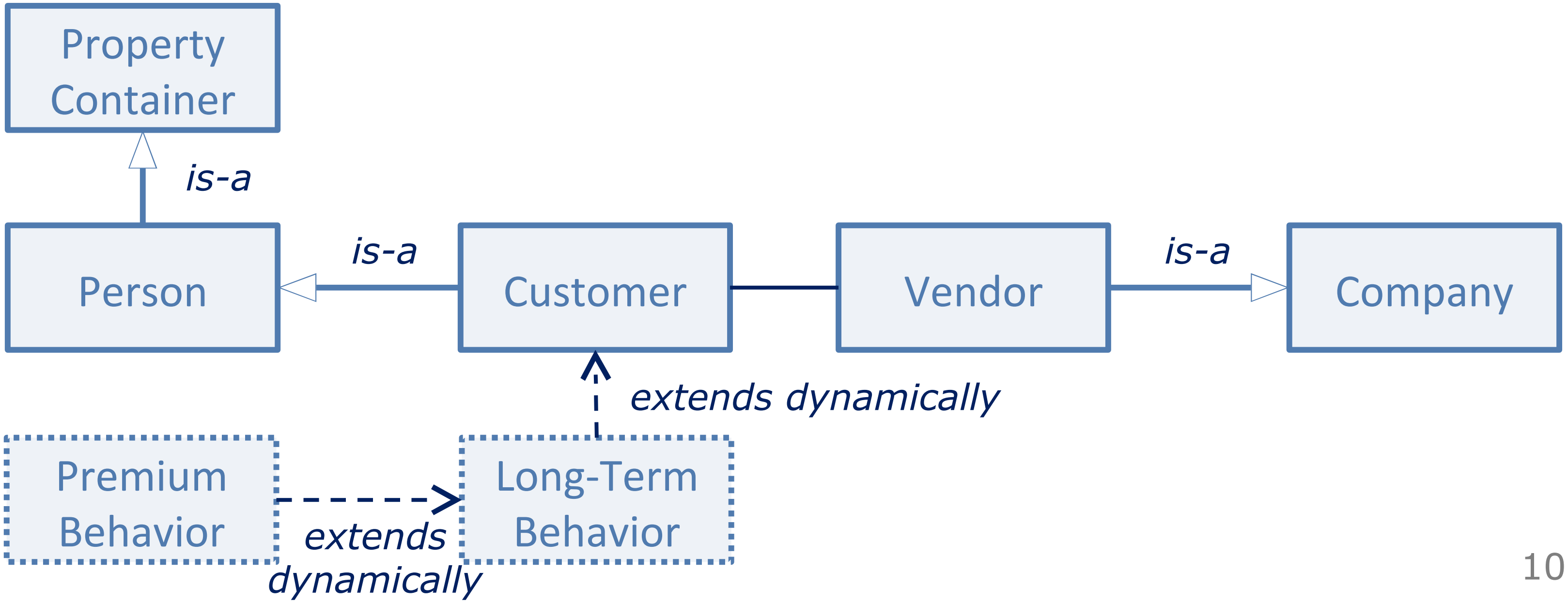
- Common Function Financial Interface (CFFI): common functionality used by other business processes
- Warehouse management
  - Stock movements
  - Quality control
- Order management (sales, purchase)
  - Order data interchange planning
  - Pricing, discounts, order acknowledgment
- Accounts payable (AP), 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

22.1 Extending San Francisco

- Dynamic Extension of
  - Classes by dynamic subclassing
  - Object life cycles by state maschine extension
  - Business rules

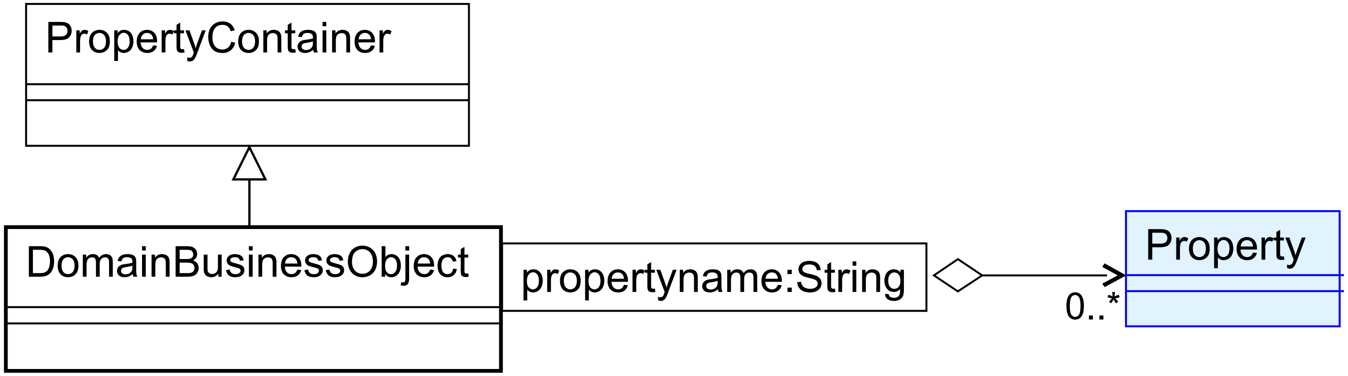
22.2.1. Extending Classes by Dynamic Subclassing

- Business objects are extensible by *subclassing* (white-box extension)
- Classes can be marked as *extension points* inheriting from *Entity*
  - Naming scheme E<number>_<name>
- 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
- *Dynamic identifiers* for extending value ranges of business value domains



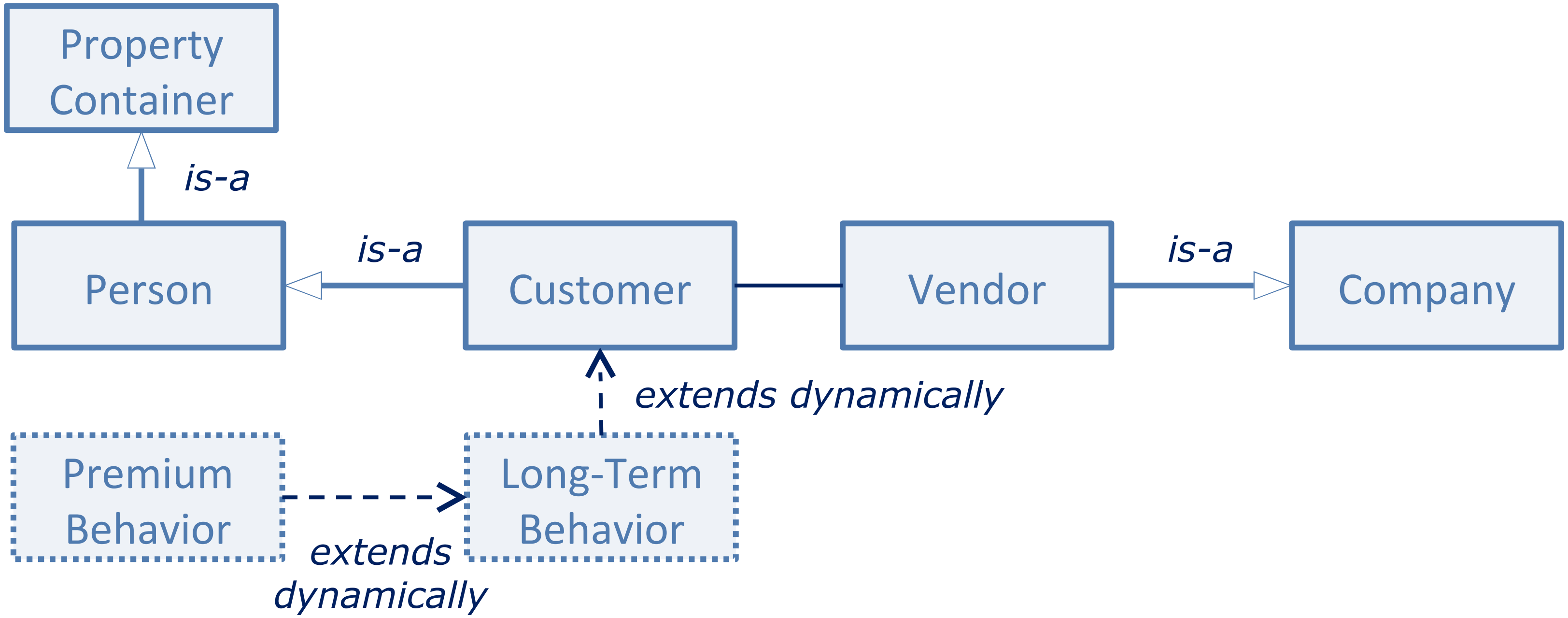
Dynamic Class Extension by Pattern "Property Container"

- Intent: dynamically extend an instance of class (a business object class) with new properties (dynamically new attributes)
- Motivation: adding dynamically new data, properties or capabilities to specific instances of business objects
  - Qualified association with key "propertyname:String"
- Related Patterns: Chain of Responsibility, Controller



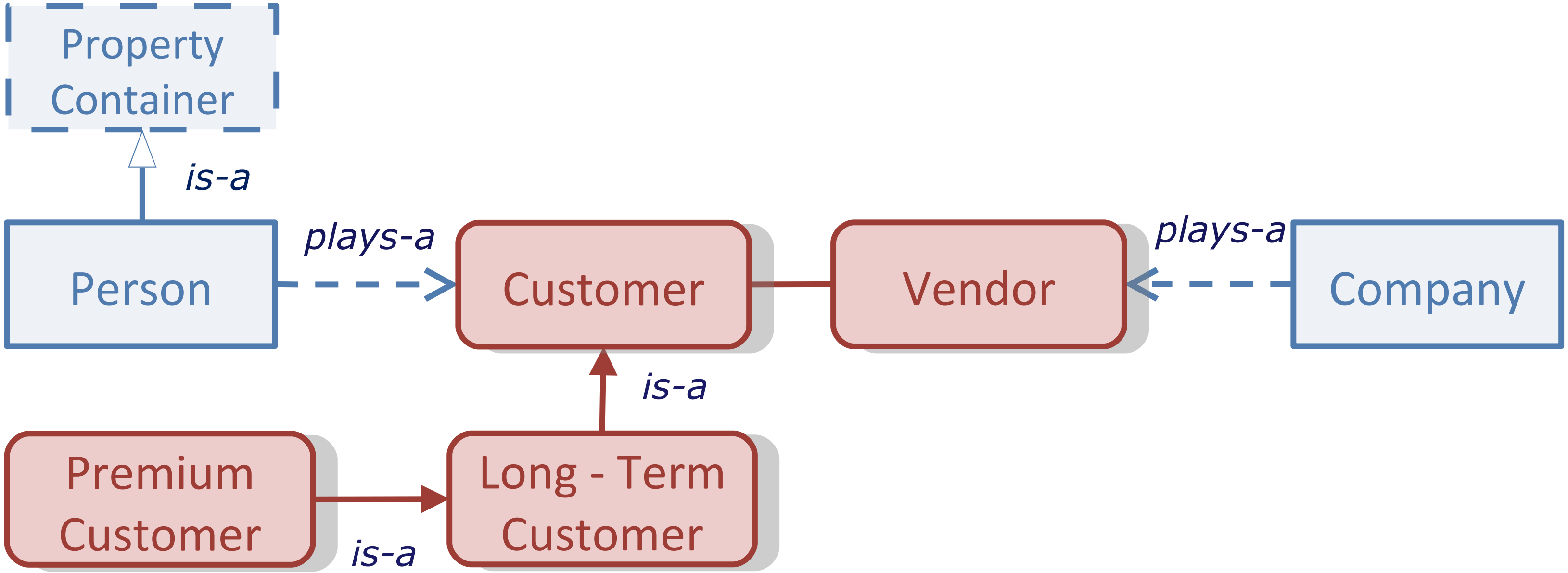
How it Should have Been: Dynamic Extension by Roles

- Class modeling does not distinguish **roles (context-based und non-rigid knowledge)**
- Roles separate the **functional core** iof an object of the **context-specific (foun** und **temporary (non-rigid)** features



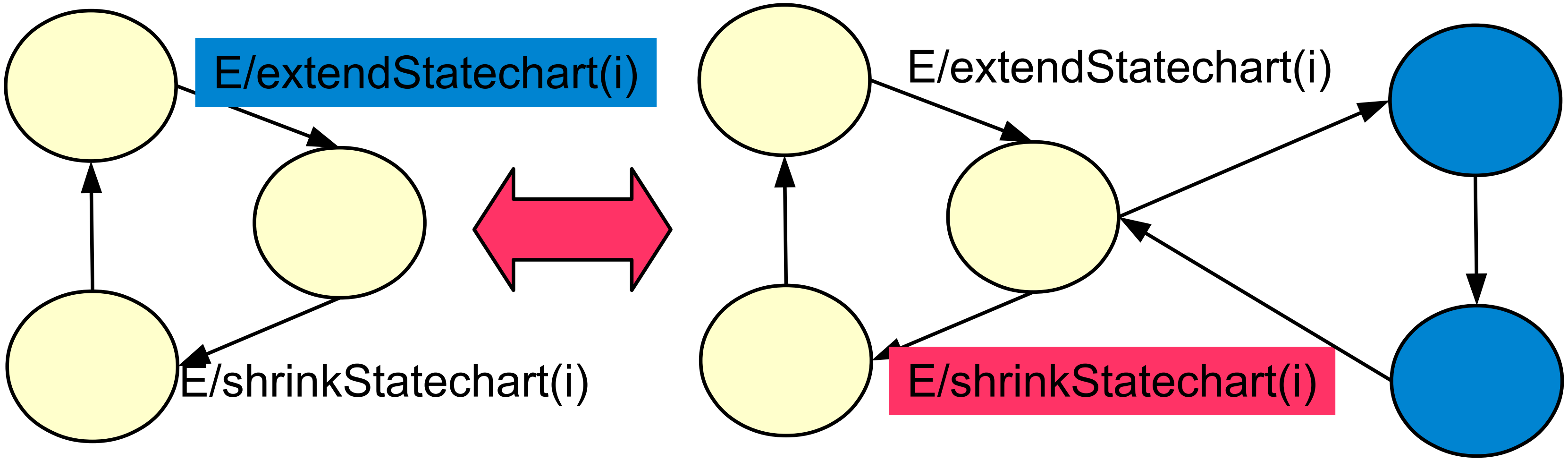
How it Should have Been: Dynamic Extension by Roles

- Property Container is not necessary, because roles add properties to core objects
- Dynamic class inheritance is replaced by <<plays-a>>



22.2.2 Lifecycle of Business Objects (Business Workflow, Process)

- A business workflow in SanFrancisco is described by an *extensible* state machine (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)

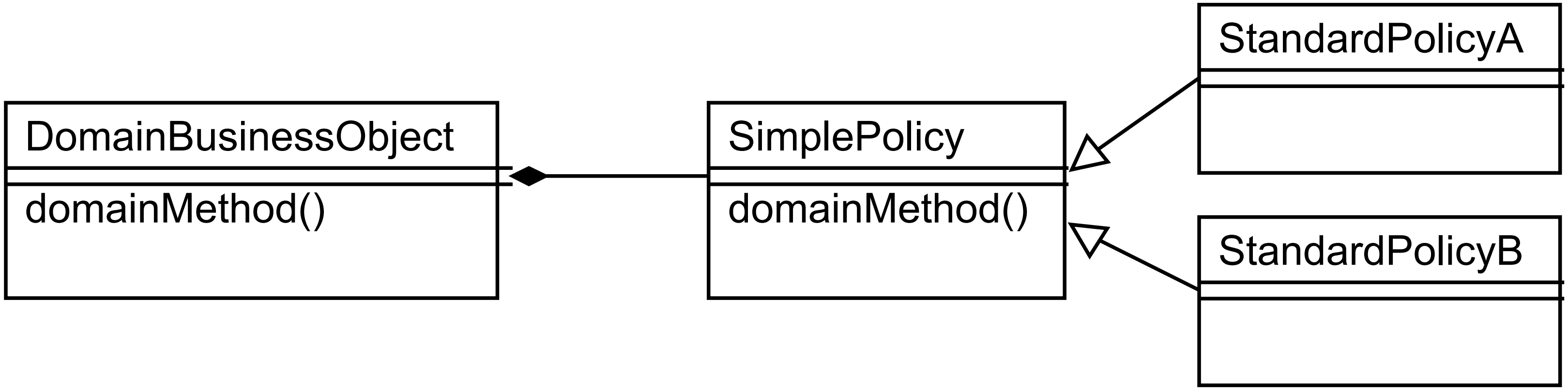


22.2.3. Extending Business rules by Policy Classes

- **Policy Patterns** implement business rules
  - *Policy classes* implement business rules a *Strategy* (TemplateClass) Pattern 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

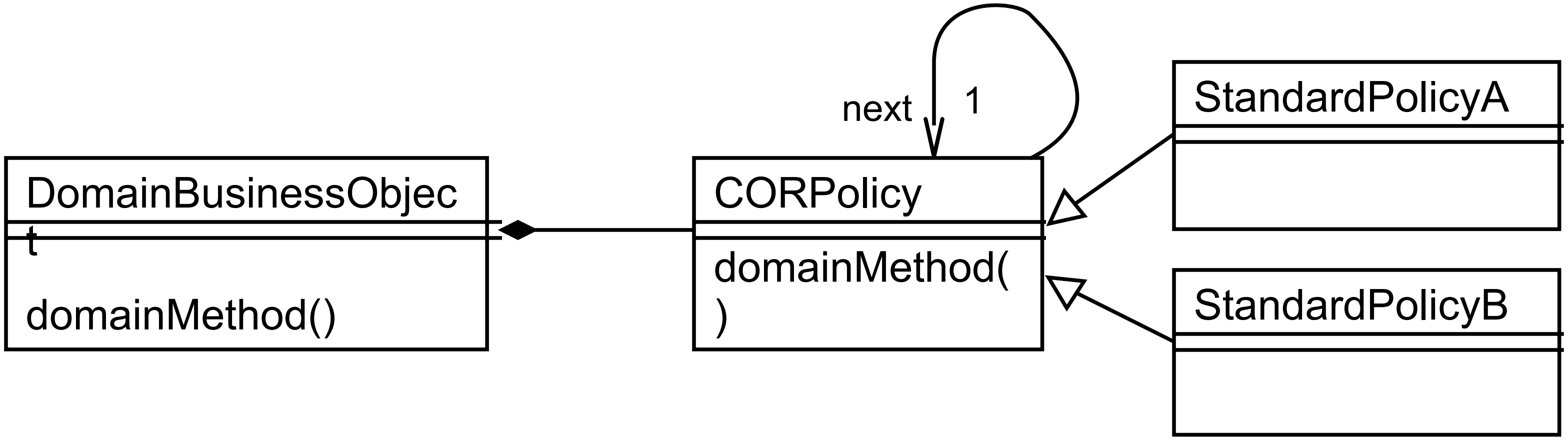
Simple Policy Pattern (Business Rule)

- 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 a 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



Chain-Of-Responsibility-Policy Pattern

- Intent: **encapsulate complex 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



22.3 San Francisco Design Patterns

- San Francisco uses several new business-related Design Patterns meeting 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

Foundational Patterns:

- Dynamic 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:

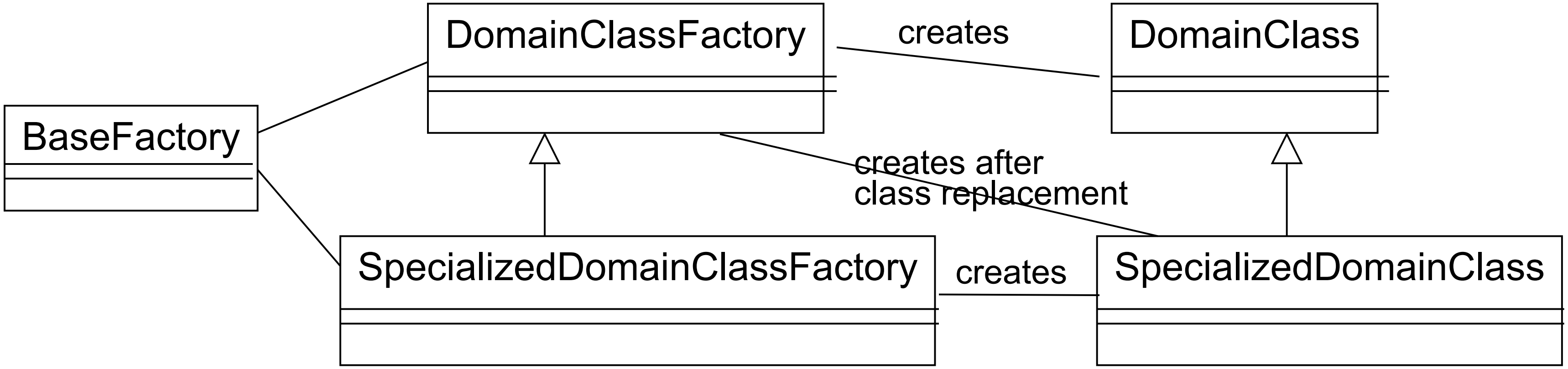
- Controller
- Key/Keyable
- Generic Interface

Dynamic Behavioral Patterns:

- Extensible Item
- Hierarchical Extensible Item
- Business Entity Lifecycle
- Hierarchy Information
- Decoupled Processes

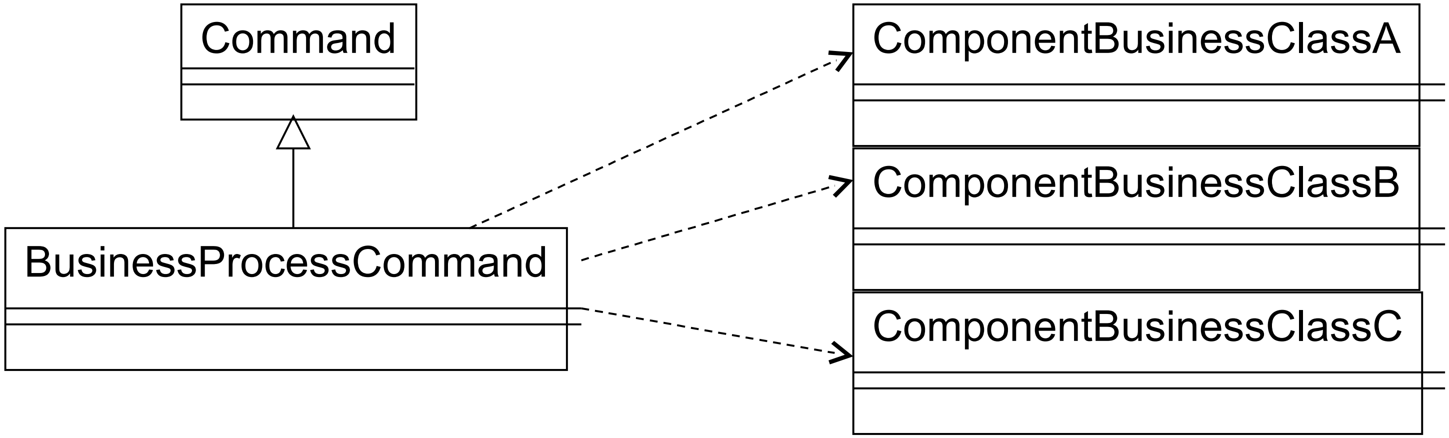
Selected SF Patterns: Dynamic Class Replacement Pattern

- 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



Selected SF 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



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.
- If you ever design a business framework, do it
  - Layered framework
  - Roles for dynamic extension
  - The SF patterns

The End