Design Patterns and Frameworks	**Exercise Sheet No. 13 with Solutions**
Dr.-Ing. Max Leuthäuser	Software Technology Group
INF 2081	Institute for SMT
`http://st.inf.tu-dresden.de/teaching/dpf`	Department of Computer Science
	Technische Universität Dresden
	01062 Dresden

Patterns in Business

In this exercise we will look at patterns which are particularly relevant in the context of business applications. The first task is on the analysis level and looks at *analysis patterns*, while the later two tasks look at design patters from a business application framework.

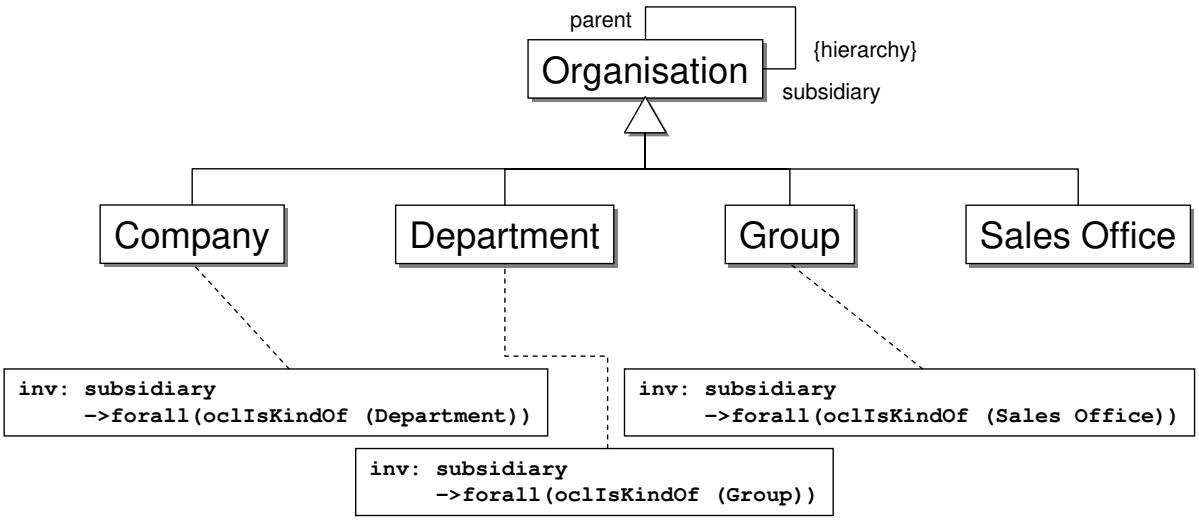
Task 13.1: Accountable Organizations

Organisations typically have a hierarchic structure, where sub-organisations report to bigger parts of the organisation above. This structure must be represented at analysis time in order to develop a correct model of the organisation.

1a) **Task:**

How can we represent an organisation with a single hierarchy? How can we model the rules governing such a hierarchy?

Solution: This can essentially be done using the COMPOSITE pattern. Notice that, because this is analysis, we do not care about implementation issues, so we do not need to distinguish between leaf and composite nodes.



As can be seen, we have attached constraints to the specific subclasses of `Organisation` to indicate the structure of the hierarchy.

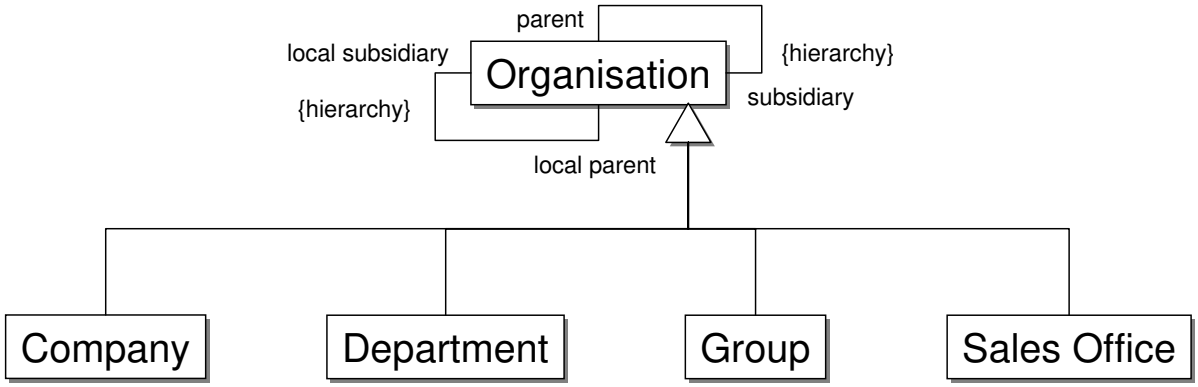
1b) **Task:**

Many organisations use different, overlapping hierarchies. For example, the Boston-based sub-group of IT-management reports to the IT-management group, which reports to Technical Infrastructure, which

eventually reports to Executive Board. At the same time, the Boston-based sub-group of IT-management is responsible for managing the company's IT systems in Boston, so it reports to Boston branch's technical director, who reports to Boston branch's board of management, who report to Executive Board. Thus, the same organisational unit is involved in at least two organisational hierarchies in this company.

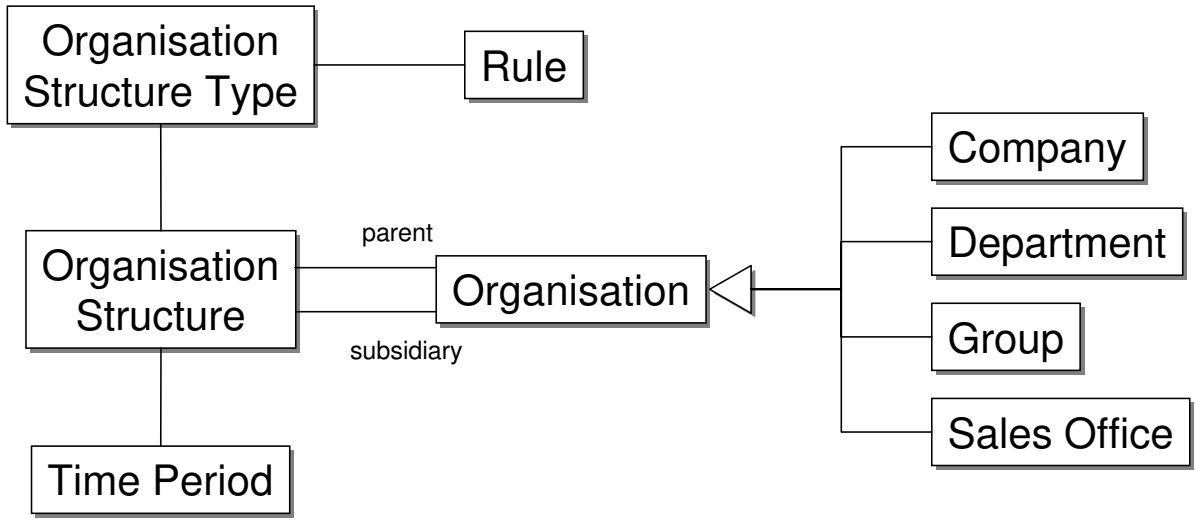
How can we model multiple hierarchies in an organisation? Where do the rules go?

Solution: The easiest way seems to be to simply add another association to the ORGANISATION from the last task:



This approach is fine, as long as there are not too many hierarchies and the constraints structuring each hierarchy do not become too complex. Keep in mind that we have left out the hierarchy constraints from the above figure. Of course we would need constraints similar to the ones we had in the first subtask, but this time for both hierarchies.

If there are more (or more complex) hierarchies, or if it looks like there might be, we can introduce an explicit type to represent the `Organisation Structure`. For this, we split the information on organisation structure into an `Organisation Structure Type`, representing general information about one particular type of organisation structure (e.g., line management as discussed in the first subtask), and an `Organisation Structure`, instances of which represent actual relations of a certain organisation-structure type. This also allows us to represent additional information about each such link—for example, the time period in which the link is to be valid. This solution is discussed as the ORGANIZATION STRUCTURE pattern in [1, Sect. 2.3].



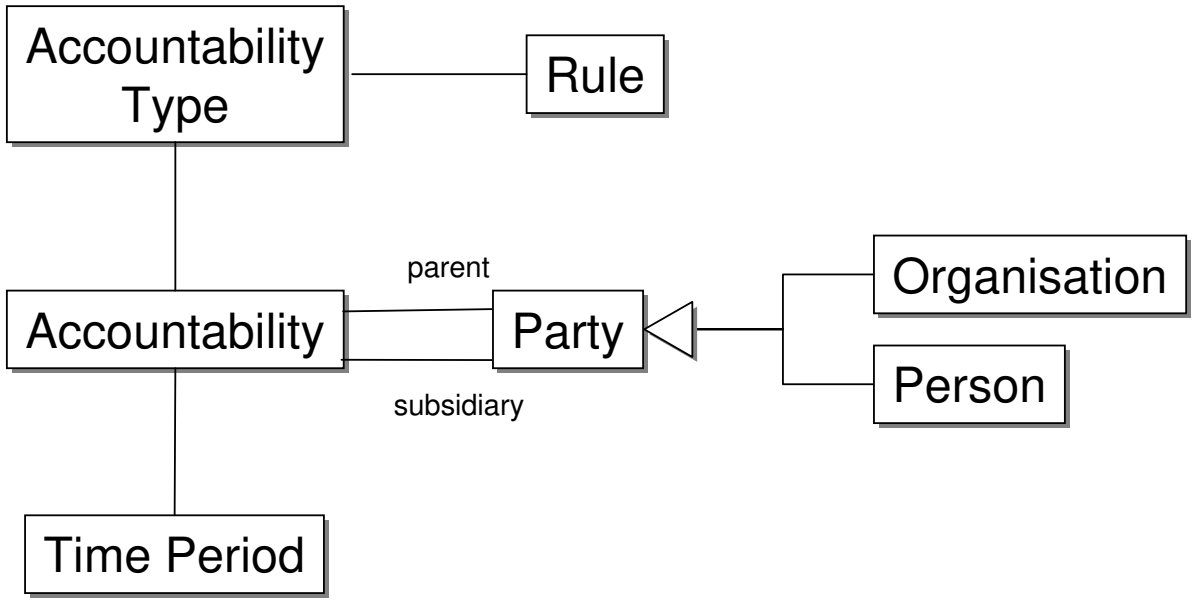
An interesting aspect of this analysis model is that it makes most sense to place the hierarchy rules with the organisation structure type now. This means all rules concerning one type of organisation structure

will all be collected in one place, so that changing the rules for one type of hierarchy is easy. Note that this structure may get problematic as soon as changes to the actual organisation units happen often as opposed to changes to the organisation hierarchies.

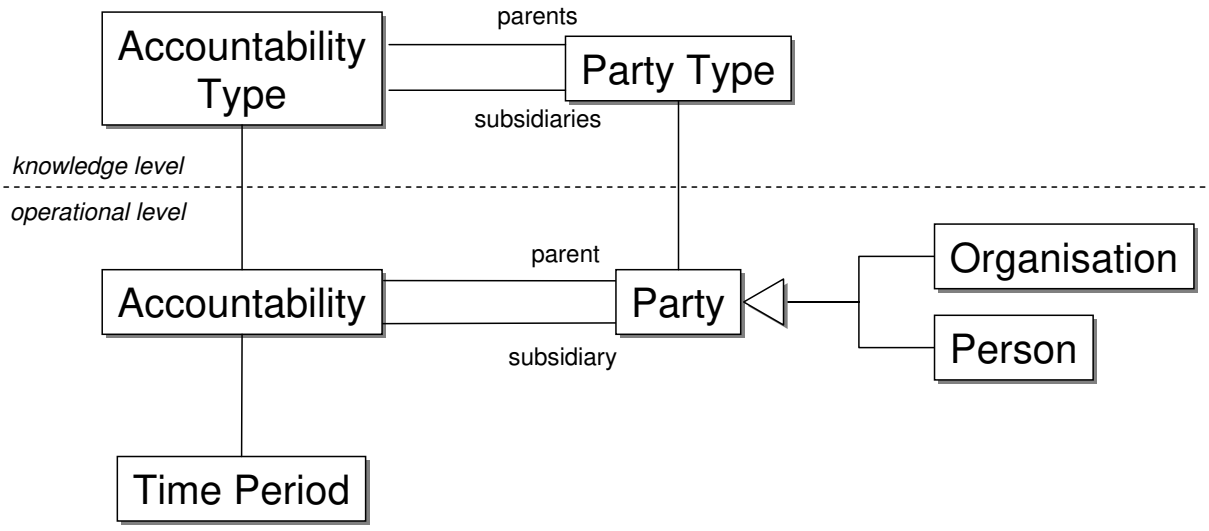
1c) **Task:**

Can the arguments and solutions given above be extended to persons dealing with organisations (or other persons)? If yes, how?

Solution: Of course, people have relations with companies. These relations (e.g., contracts) are typically about responsibility and accountability (and not necessarily about organisation structure only). They may only exist for a certain period of time. Thus, we can transfer the concepts from above:



This is discussed in [1] under the heading of ACCOUNTABILITY. The complexity of this model has increased, though, because there are many more types of accountability than of organisation structure, only. To maintain overview of these concepts, it may be worthwhile to add a `Party Type` which can be used to model which `Parties Accountability` instances of a certain `Accountability Type` may connect. This essentially introduces a meta-level into our model.



Bibliography

1. Martin Fowler. *Analysis Patterns – Reusable Object Models.* Object Technology Series, Addison-Wesley, 1997.

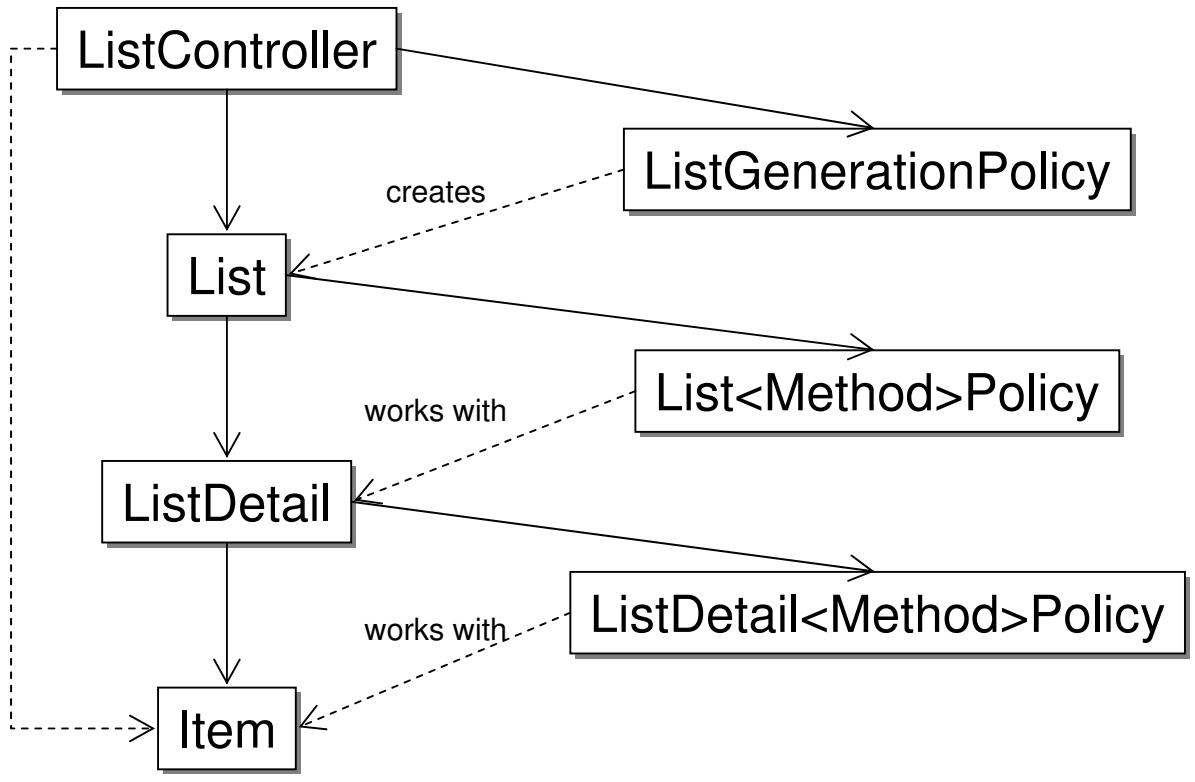
Task 13.2: Generating Complex Lists

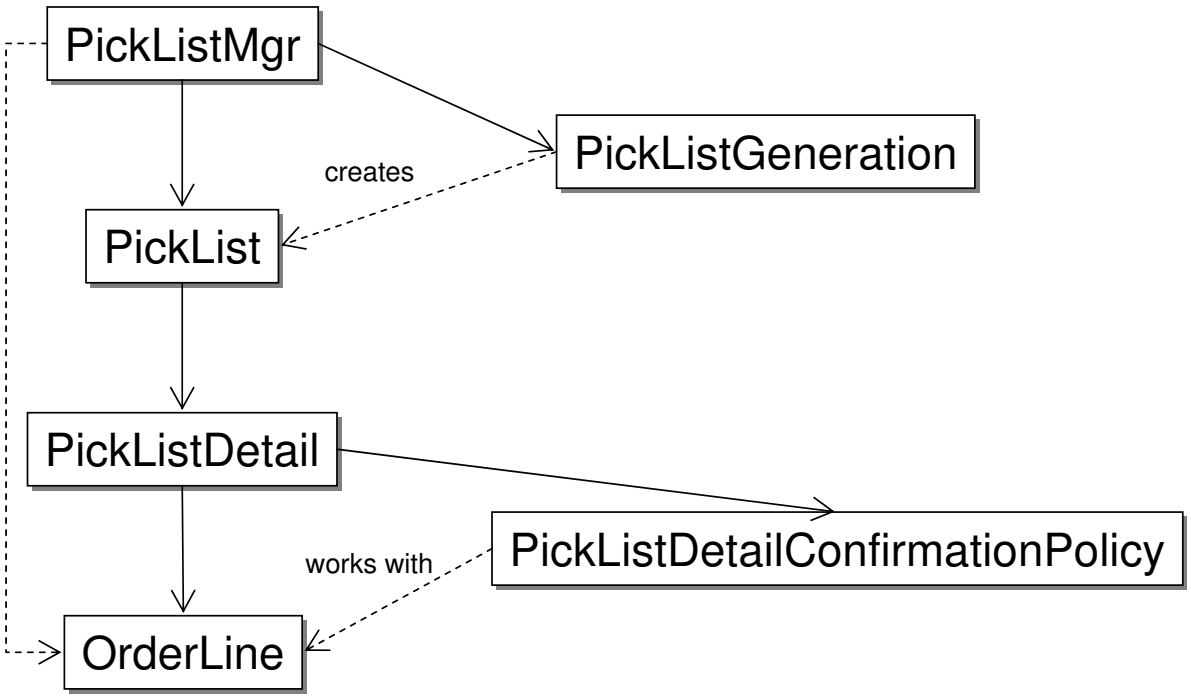
In many business applications it is necessary to generate lists from multiple data sources for varying purposes. Often, for every item in the list it must be possible to determine the original data source, so that actions on the list can be reflected into actions on the original data. It is necessary to be able to vary the algorithm for list creation (including the number of lists to generate) as well as the algorithms to be applied on individual list items or complete lists.

For example, in a warehouse application, at some point, sets of current orders must be transformed into lists of items to be picked from the warehouse. We want to optimise these lists so that for each product only one line occurs in the list, with an amount corresponding to the accumulated amount for this product from all orders.

What design patterns can we use for this?

Solution: We can use Strategy to enable variation of list generation and list modification algorithms. In [1] the authors describe this pattern (called List Generation) using so-called Policies, which are no more than extended Strategies.





Bibliography

1. James Carey, Brent Carlson, and Tim Graser. *San Francisco Design Patterns – Blueprints for Business Software.* Addison-Wesley, 2000.

Task 13.3: Dynamic Life Cycle

Business objects often have a life cycle the individual elements of which stay the same, but whose arrangement may change depending on context. For example, in a warehouse, an order needs to be treated differently depending on whether it is an Internet order or a direct sales in a factory outlet. In the former case, the order is accepted, planned (determining which storage is to serve the order), prepared for picking, picked, shipped, and invoiced. In the latter case, it is accepted, prepared for picking, picked, and invoiced. Planning and shipping are not necessary, because the customer is right there in the outlet shop and picking can only sensibly happen in the warehouse associated to the outlet.

3a) **Task:**

How can we realise the individual states in the life-cycle of the order? Remember that each state may require its own interface operations and data.

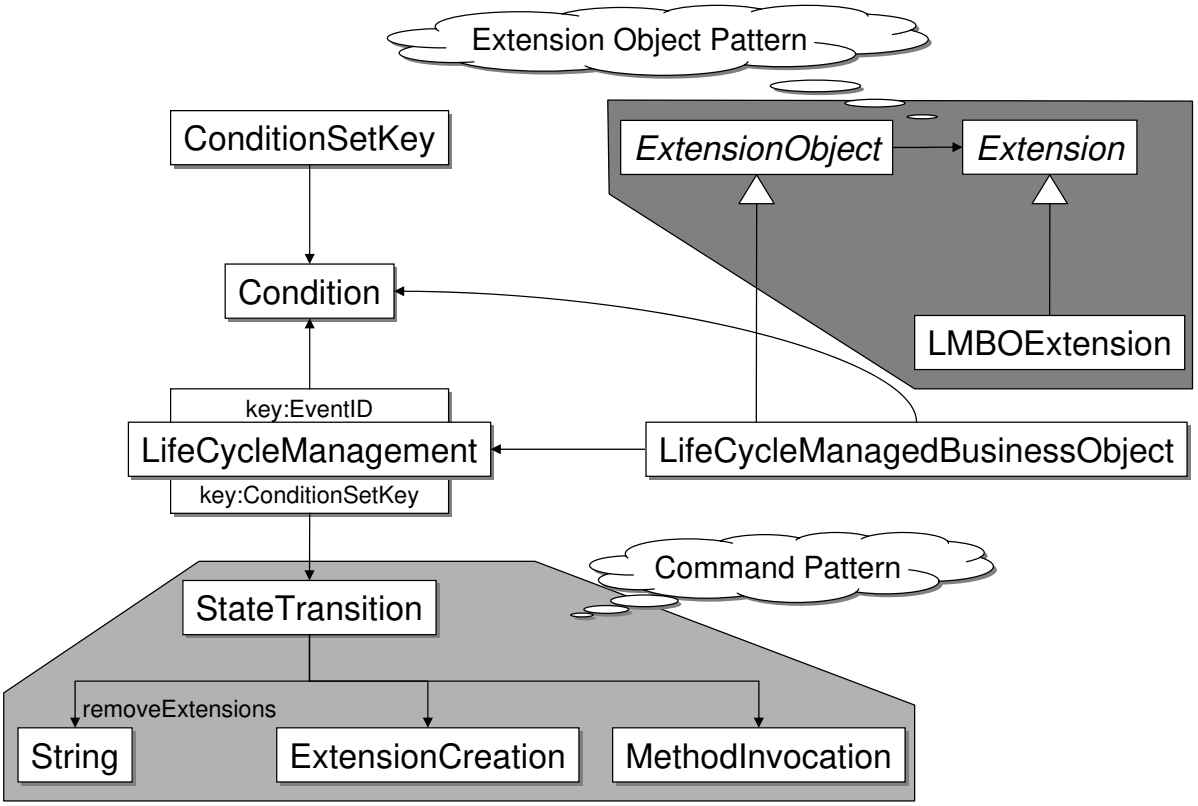
Solution: We can use the Extension Object pattern, realising each state as an `Extension` object providing the additional operations and data elements. A state change then occurs by removing on extension object and adding another.

3b) **Task:**

How can we realise the different life cycles while reusing as much of the order code as possible?

Solution: We must externalise the life-cycle management from the order code. We do this by adding a `LifeCycleManagement` class that has knowledge of the structure of a certain life cycle.

But, how do we implement the mapping between actions on the business object and state transitions in the life cycle? Obviously, life-cycle state changes are triggered by certain messages sent to the business object. On the other hand, not every message triggers a life-cycle state change. Deciding which messages trigger which state change is the responsibility of the `LifeCycleManagement`. So, both the order and the life-cycle management must be involved in this decision. But the order object must not know about life-cycle state changes. We solve this by introducing a double mapping: First, the order sends an event whenever it receives a message, finishes handling a message, or at any other significant moment to the life cycle management object. That object maintains a mapping from events to conditions, where each condition (possibly together with the set of previously activated conditions) demands a certain life-cycle state to be active. The life cycle management object asks the order object to maintain a list of all conditions activated during its life time. Additionally, the life cycle management object determines if a state change must be performed and if so start any activities required.



This pattern is treated as Business Entity Lifecycle in [1].

Bibliography

1. James Carey, Brent Carlson, and Tim Graser. *San Francisco Design Patterns – Blueprints for Business Software.* Addison-Wesley, 2000.