# Flexible Software Architectures for Presentation Layers demonstrated on Medical Documentation with Episodes and Inclusion of Topological Report

Jens Bohl <info@jens-bohl.de>
Torsten Kunze <zone3@gmx.de>
Christian Heller <christian.heller@tu-ilmenau.de>
Ilka Philippow <ilka.philippow@tu-ilmenau.de>

Technical University of Ilmenau
Faculty for Computer Science and Automation
Institute for Theoretical and Technical Informatics
PF 100565, Max-Planck-Ring 14, 98693 Ilmenau, Germany
http://www.tu-ilmenau.de, fon: +49-(0)3677-69-1230, fax: +49-(0)3677-69-1220

**Abstract.** This document describes how existing design patterns can be combined to merge their advantages into one domain- independent software framework. This framework, called *Cybernetics Oriented Programming* (CYBOP) [Hel03], is characterized by flexibility and extensibility. Further, the concept of *Ontology* is used to structure the software architecture as well as to keep it maintainable. A *Component Lifecycle* ensures the proper startup and shutdown of any systems built on top of CYBOP.
The practical proof of these new concepts was accomplished within the diploma thesis of Jens Bohl [Boh03] which consisted of designing and developing a module called *Record*, of the *Open Source Software* (OSS) project *Res Medicinae*. The major task for this module is to provide a user interface for creating medical documentation. New structure models such as *Episodes* were considered and implemented. In this context, the integration of a graphical tool for *Topological Documentation* was also highly demanded. The tool allows documentation with the help of anatomical images and the setting of markers for pathological findings.

**Keywords.** Design Pattern, Framework, Component Lifecycle, Ontology, CYBOP, Res Medicinae, Episode Based Documentation, Topological Documentation

## 1 Introduction

Quality of software is often defined by its maintainability, extensibility and flexibility. *Object Oriented Programming* (OOP) should help to achieve these goals – but this wasn't possible by only introducing another programming paradigm.
So, major research objectives are to find concepts and principles to increase the reusability of software architectures and the resulting code. *Frameworks* shall prevent code duplication and development efforts. Recognizing recurring structures means finding *Design Patterns* for application on similar problems. These two concepts – frameworks and design patterns – depend on each other and provide a higher flexibility of software components [Pre94].
The aim of this work was to find suitable combinations of design patterns to compose a framework that is characterized by a strict hierarchical architecture. Everything in universe is organized within a hierarchy of elements – the human body for example consists of organs, organs consist of regions, regions consist of cells and so on. This very simple idea can also be mapped on software architectures – and basically, this is what this document is about.
What kind of techniques to realize such a concept of strict hierarchy does software engineering provide? The following chapters first introduce common design patterns and the lifecycle of components as templates for own ideas and then show how the resulting framework *Cybernetics Oriented Programming* (CYBOP) [Hel02] is designed.

## 2 Design Principles

### 2.1 Essential Design Patterns [EG96]

Design patterns are elements of reusable software. They can be used for solving recurrent design problems and are recommendations on how to build software in an elegant way. With the help of these patterns, software shall be more extensible, flexible and easy to maintain in respect to further enhancements. The following patterns are essential within CYBOP.

**Composite** This design pattern allows creating tree-like object structures. One object is child of another

object and has exactly one parent. This pattern is often used to realize *whole-part* relations: one object is *part* of another one.

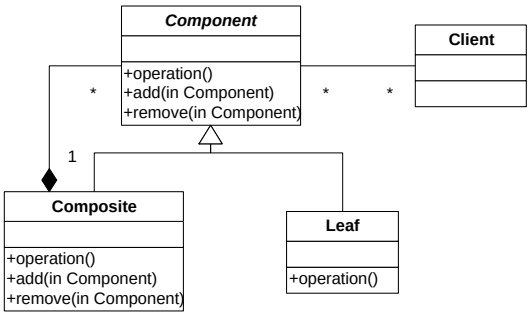


**Fig. 1.** Composite Pattern

**Layers** With the help of this pattern, software can be organized in horizontal layers. Modules and applications can be separated into logical levels, whereby these levels should be as independent from each other as possible, to ensure a high substitutability.

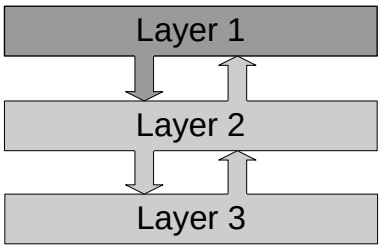


**Fig. 2.** Layer Pattern

**Chain Of Responsibility** Messages initiated by a particular object can be sent over a chain of instances to the receiving object. So, either the message will be transmitted over a bunch of objects or evaluated immediately by the target object.

**Model-View-Controller** Dividing the presentation layers into the logical components *Model*, *View* and *Controller*, is a very approved way for designing software for user interfaces. The model encapsulates the data presented by the view and manipulated by the *controller*.

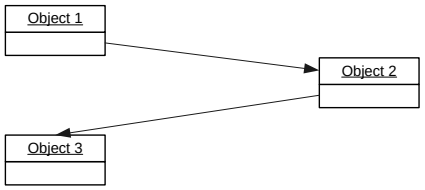


**Fig. 3.** Chain Of Responsibility Pattern

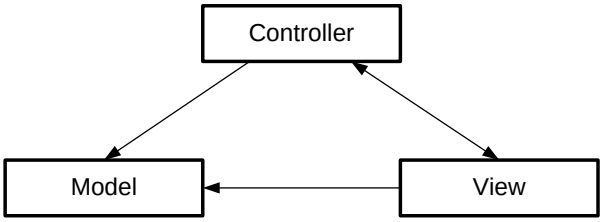


**Fig. 4.** Model-View-Controller Pattern

**Hierarchical Model-View-Controller [JC01]** The Hierarchical Model-View-Controller combines the essential design patterns *Composite*, *Layers*, and *Chain of Responsibility* into one conceptual architecture. It consists of hierarchical layers containing *MVC-Triads*. These triads conventionally separate the presentation layer into model, view, and controller. Triads communicate with each other by relating over the controller object. Here is a short explanation of this concept, using a practical example: The upper-most triad could represent a dialog, the middle one a container such as a panel. In this container, a third triad, for example a button, can be held.
All Triads communicate with each other by using the controller component. The basic idea behind this concept is to divide the presentation layer into hierarchical sections.

### 2.2 Component Lifecycle [ava02]

Each *Component* lives in a system that is responsible for the component's creation, destruction etc. When talking about components, this article sticks to the definition of Apache-Jakarta-Avalon [ava02], which considers components to be *a passive entity that performs a specific role*.
A component has a number of methods which need to be called in a certain order. The order of method calls is what is known as *Component Lifecycle*. An outside, active entity is responsible for calling the lifecycle methods in the right order. In other words, such an entity or *Component Container* can control

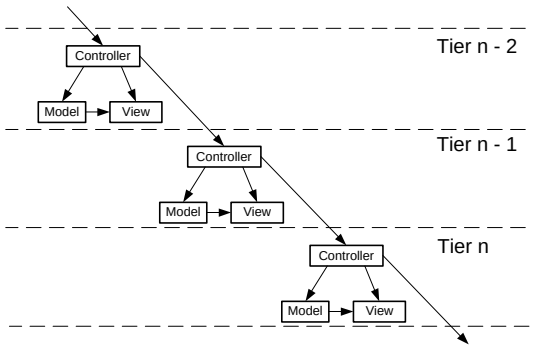


**Fig. 5.** Hierarchical Model-View-Controller Pattern

and use the component. The Avalon documentation [ava02] says:

> "It is up to each container to indicate which lifecycle methods it will honor. This should be clearly documented together with the description of the container."

## 3 An Extended Component Lifecycle

The CYBOP lifecycle of components is an extension of the lifecycle idea of Apache – basically the same idea but another background and realization.
All *whole-part associations* between objects were organized under the rules of the component lifecycle. Analogous to the lifecycle of organic cells, the relations were created and destroyed in a sequence of lifecycle steps. These steps are realized as method calls on the components (see figure 6).

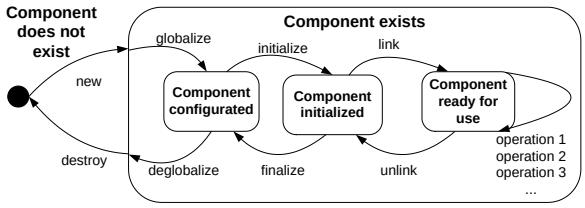


**Fig. 6.** State Diagram of CYBOB's Component Lifecycle

## 4 Ontologies

An ontology is a catalogue of types that are depending on each other in hierarchical order. It is a formal specification concerning a particular objective. CYBOP consists of three such ontologies:

- Basic Ontology
- Model Ontology
- System Ontology

Figure 7 shows the model ontology. The layer super types are `Record`, `Unit`, `Heading` and `Description`. These classes are super types of all classes in a particular ontological level.
The right side shows a concrete implementation of the model ontology – the *Electronic Health Record* [ope02]. This data structure contains all information concerning a particular patient. The figure shows `Problem` types in level `Unit`. These consist of episodes containing instances of `PartialContact`. In level `Heading`, the structural elements of a partial contact can be found – `Subjective`, `Objective`, `Assessment` and `Plan`. Therapeutical issues are placed in level `Description` – such as `Medication` with particular dose.
As shown, the concept of ontology can be used to

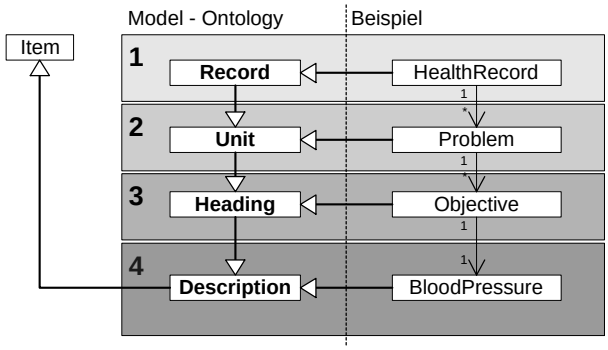


**Fig. 7.** Model Ontology

organize data structures in a hierarchical order by defining logical layers with super types.

## 5 CYBOP

Section two introduced essential design patterns that represent the main structure of the CYBOP framework. Section three explained the Component Lifecycle and section four the well-known idea of ontology. Now these design principles and conceptual architectures will be combined to comprise their advantages and to increase the demanded quality characteristics: high flexibility and maintainability.
*Structure by Hierarchy* – this is the basic idea behind CYBOP. Extending the concept of Hierarchical Model-View-Controller to whole software architectures, CYBOP was designed to be the domain-independent backbone for information systems of any

kind. Originally designed for medical purposes, it should also be usable for insurance, financial or any other standard applications in future.

### 5.1 Class Item

As shown, tree-like structures can be realized by the Composite pattern. In CYBOP, this pattern can be found in class `Item` which is super type of all other classes. References, respectively relations to child elements are held within a hashmap. No attributes were used except of this hashmap. Every element of the map can be accessed by a special key value. So, no particular `get-` or `set`-methods were needed for attributes.

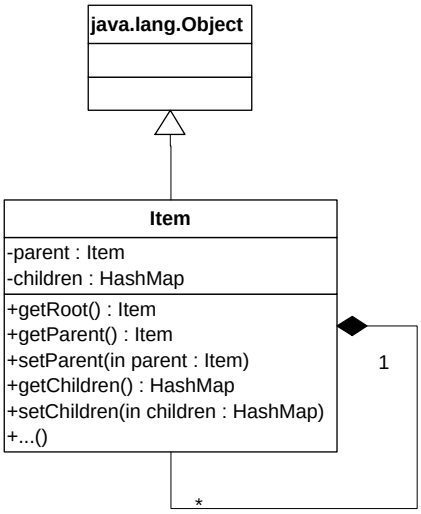


**Fig. 8.** Class Item

### 5.2 Basic Structure

Comprising the design patterns *Composite*, *Layers*, and *Chain of Responsibility*, the framework CYBOP is comparable to a big tree containing objects organized in different levels. Figure 9 shows the object tree and the different levels of granularity.

## 6 Record – An EHR Module

The practical background for the application of CYBOP is *Res Medicinae*. A modern clinical information system is the aim of all efforts in this project. In future, it shall serve medical documentation, archiving,

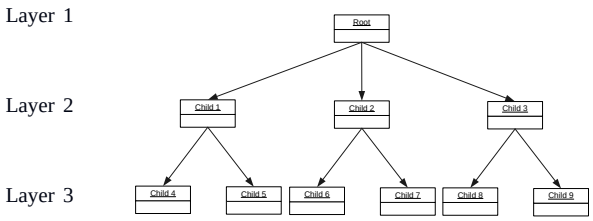


**Fig. 9.** Basic Structure

laboratory work etc. *Res Medicinae* is separated into single modules depending on different tasks.
One of these modules is *Record* – an application for documenting medical information (see figure 10). In addition to new documentation models, it also contains a tool for topological documentation.
Starting from an over-all view of the human body, it is possible to reach every organ or region of the body in detail (see figure 11).

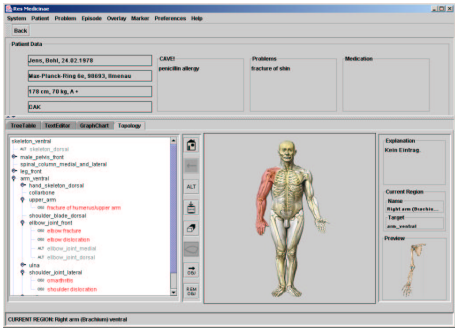

**Fig. 10.** Screenshot of Record [urb]

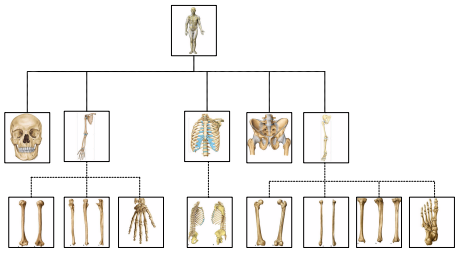

**Fig. 11.** Excerpt from Topological Structure of Human Skeleton [urb]

## 7 Summary

Software design patterns are essential elements of frameworks. They can be combined to comprise their advantages and to realize hierarchical structures.
These structures can be created and destroyed in the lifecycle of components. In that lifecycle, object relations become more transparent and are easier to control and to maintain.
Ontologies can help to model particular domains and to layer software. Every level of these ontologies has a particular super type, whereby these types depend on each other by inheritance. This concept supports the modelling and logical separation of software into hierarchical architectures. The granularity of the ontology (number of ontological levels) can be adapted to particular requests.
By applying the new concepts introduced in this document, the quality of software can be greatly increased. The time for building systems can be reduced to a minimum. The clear architecture avoids common confusion as the systems grow.

## 8 Acknowledgements

Our special thanks go to all Enthusiasts of the Open Source Community who have provided us with a great amount of knowledge through a comprising code base to build on. We'd also like to acknowledge the contributors of *Res Medicinae*, especially all medical doctors who supported us with their analysis work [CH02] and specialised knowledge in our project mailing lists. Further on, great thanks goes to the Urban and Fischer publishing company, for providing anatomical images from their *Sobotta – Atlas der Anatomie*.

## 9 About the Author

Jens Bohl, born in 1978, has been studying computer science from October 1997 until January 2003 at the Technical University of Ilmenau with subsidiary subject *Medical Informatics*. During this period, he worked six months at *software design and management AG*. He is a member of the Java-based *Res Medicinae* project and active developer of the Open Source Community. In March 2003, he started working as a software engineer and consultant in Frankfurt/Main.

## References

[ava02] Apache jakarta avalon framework. http://jakarta.apache.org/avalon/index.html, 2002.

[Boh03] Jens Bohl. *Moeglichkeiten der Gestaltung flexibler Softwarearchitekturen fuer Praesentationsschichten, dargestellt anhand episodenbasierter medizinischer Dokumentation unter Einbeziehung topologischer Befundung.* 2003.

[CH02] Roland Colberg et al. Christian Heller, Karsten Hilbert. *Analysedokument zur Erstellung eines Informationssystems fuer den Einsatz in der Medizin.* 2002.

[EG96] Ralph Johnson et al. Erich Gamma, Richard Helm. *Entwurfsmuster–Elemente wiederverwendbarer objektorientierter Software.* Bonn: Addison-Wesley, 1996.

[Hel02] Christian Heller. Cybernetics oriented programming. http://resmedicinae.sourceforge.net, 2002.

[Hel03] Christian Heller. Cybop-cybernetic oriented programming. http://www.cybop.net, 2003.

[JC01] Gaurav Pal Jason Cai, Ranjit Kapila. Hmvc: The layered pattern for developing strong client tiers. http://www.javaworld.com, 2001.

[ope02] Open ehr. http://www.openehr.org, 2002.

[Pre94] W. Pree. Meta patterns – a means for capturing the essentials of reusable object-oriented design. In *Proceedings of ECOOP '94*, pages 150–162, 1994.

[urb] Urban und fischer verlag, muenchen. http://www.urbanfischer.de/.