# XL-AOF - Design and Implementation of an eXtensible Lightweight Aspect-Oriented Framework for Distributed Applications

Technische Universität Wien
Institut für Computersprachen
Programmiersprachen und Übersetzer
Ao. Univ.-Prof. Dipl.-Ing. Dr. eva Kühn

Magisterstudium:
Software Engineering & Internet Computing

Gerald Fessl

## XL-AOF

**XL-AOF** - e**X**tensible **L**ightweight **A**spect-**O**riented **F**ramework - offers a lightweight and extensible framework based on the aspect-oriented paradigma without any need for neither a special compiler weaving the aspect programs with the business program at compile time nor a new virtual machine, an exhaustive configuration mechanism or the need to learn a new language to be able to integrate the desired functionality into the implemented software system. This framework is extensible with new modules in a simple way, whereas those modules need not be restricted on distributing concerns.

Although this aspect-oriented framework is applicable in different application areas, the target domain is the distribution of systems using the virtual shared memory approach VSM.

## Virtual Shared Memory (VSM)

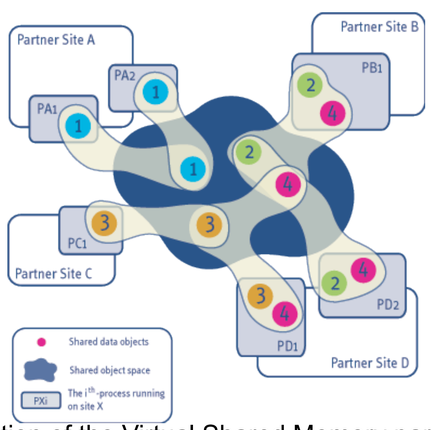



Illustration of the Virtual Shared Memory paradigm

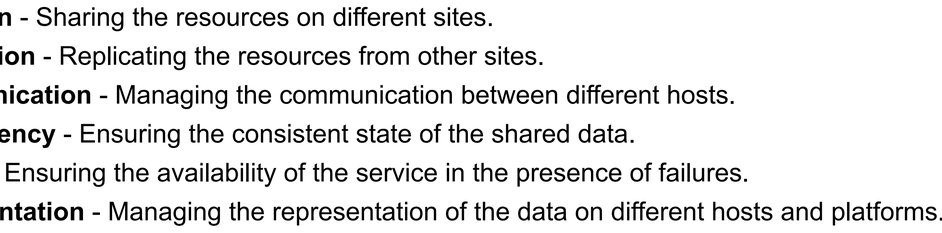
The virtual shared memory approach uses a virtual space which can be shared among several systems. To distribute some data, this data is written into and read out of this shared space. By using a suitable middleware, the following issues can be simplified:

- **Location** - Retrieving some information on different sites using names.
- **Migration** - Sharing the resources on different sites.
- **Replication** - Replicating the resources from other sites.
- **Communication** - Managing the communication between different hosts.
- **Concurrency** - Ensuring the consistent state of the shared data.
- **Failure** - Ensuring the availability of the service in the presence of failures.
- **Representation** - Managing the representation of the data on different hosts and platforms.
- **Load balancing** - Managing the load balance on different hosts.
- **Persistency** - Ensuring the survival of data in the presence of crashes.
- **Notification** - Notifying other processes about changes in shared data objects.

But there still remains a high degree of complexity when systems are distributed as several concerns - the so-called crosscutting concerns - are scattering and tangling the code, thus making the software development difficult.

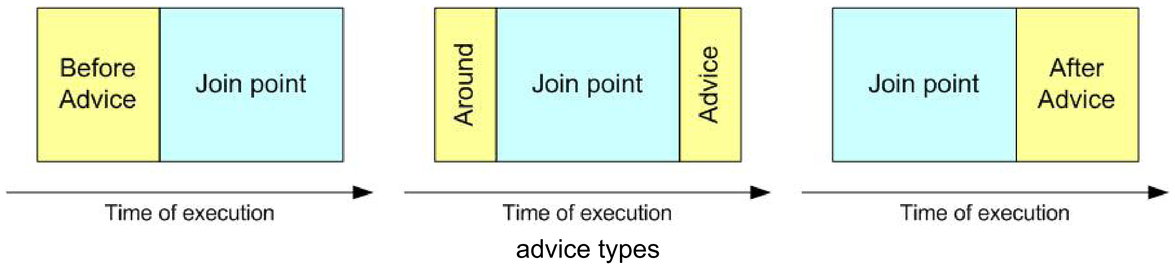
## Aspect-Oriented Programming (AOP)

Using the **A**spect-**O**riented **P**rogramming (AOP) approach which was intruduced by Kiczales et al. in 1997, these concerns can be separated by a better way of modularization. If business logic and aspect code have been defined seperately, an Aspect Weaver weaves these parts together.




dynamic subclass-proxy approach - weaving business logic with aspect code

XL-AOF uses the dynamic subclass-proxy approach to weave aspect code into the business logic at runtime. Class definitions can be extended by special intertype declaration mechanisms called Mixins. Otherwise, if certain join points - special points in the execution flow of an application - are reached, the behaviour of an application can be extended by some advice code that can be executed **before**, **around** or **after** the join point.




advice types

## Aspect-Oriented Software Development (AOSD)

To identify all important aspects in a project, it is important to start this task at the beginning of the software development process, when the users requirements are evaluated. Furthermore, special attention on the separation of concerns has to be paid when the application is designed, implemented and tested.

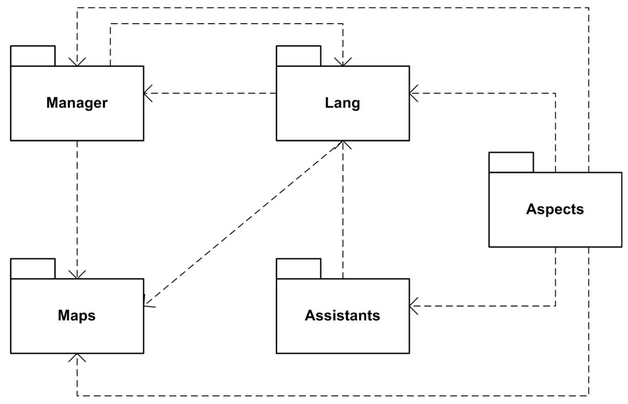
Two approaches are discussed in this work - the Theme Approach from Baniassad and Clarke and the Use-Case Approach from Jacobson and Ng. These approaches provide different methodologies for analysing the problem and designing the application with respect to aspect-oriented programming.

The Use-Case Approach was elected to be used for analysing and designing the new AOPCorso framework. The procedure is the following:

**Analysis:**
- Identification of Use-Cases
- Spezification of Use-Cases
- Modelling of Use-Case Realizations
- Comparison of Use-Case Realizations

**Design:**
- Identification of Design Elements
- Design of Operation Extensions
- Identification of Components and Interfaces
- Refining Design Elements
- Achieving Extensibility

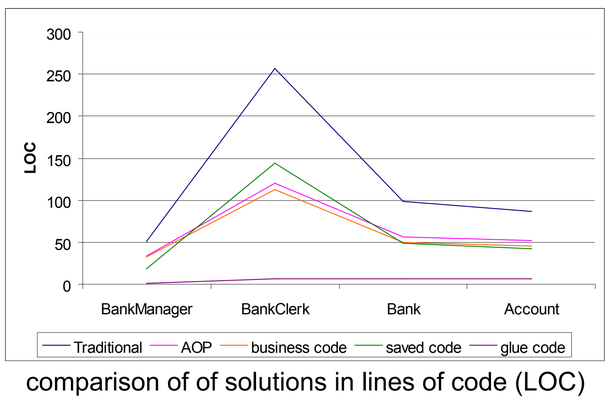
In this approach, some new use-case items were introduced, the element structure (very general view on the packages and components), the use-case slice (a layer comprising the elements of a single concern ; can be put on the element structure like a slide on an overhead projector) and the use-case module (contains each representation of the use-case realizations in the different phases of the software development cycle).




example of an element structure

## Proof of Concept

Finally, a bank accounting system is implemented to compare the traditional way with the aspect-oriented way. This application is made of the entities Bank and Account which can be persisted in the virtually shared space, and the controllers BankManager and BankClerk which are used to perform some operations on these entities.




comparison of of solutions in lines of code (LOC)

Concerning this sample application, a high degree of code reduction could be achieved, proving that the use of aspect-oriented programming leads to a better modularized and structured code, thus improving the software development process.