AOSA - Betriebssystemkomponenten und der Aspektmoderatoransatz

Wasif Gilani

wasif@informatik.uni-erlangen.de

Introduction

- Operating system design issues
- Aspect-oriented programming
- Architectural issues
- Aspect Moderator Framework
- Aspect-oriented Framework
- Summary and Conclusions

Operating System Design Issues

- Hardware oriented
  - Physical Networks, Communication protocol design
  - Physical clock synchronization
  - Storage
  - System components

Operating System Design Issues

- Software-oriented
  - Distributed algorithms
  - Naming, resource allocation
  - Distributed operating systems
  - Reliability tools and languages
  - Real-time systems and performance measurement

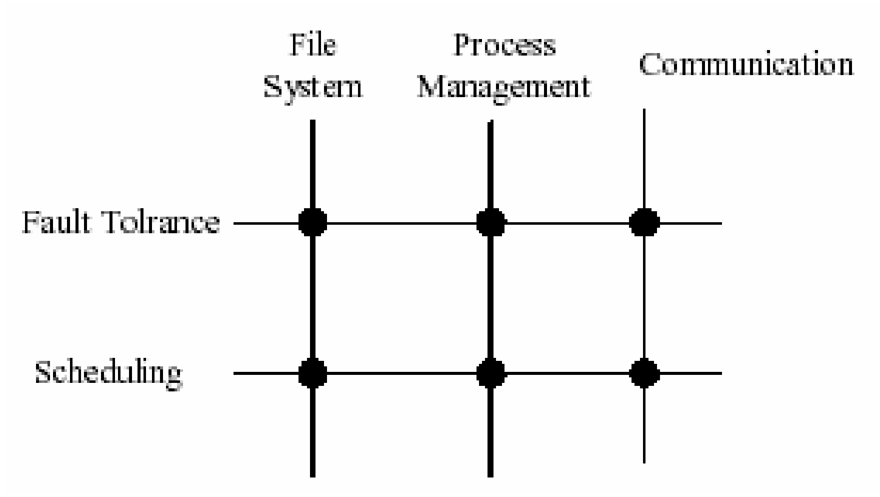
Problems

- Separation of concerns
  - No universally accepted methodology
- Functional decomposition
  - Achieved along one dimension, not able to address complete separation of concerns
- OOP suffers from cross-cutting code for scheduling, synchronization, fault tolerance, etc
  - Distributed and concurrent systems

Aspect-oriented programming

- **Aspects:**

Properties of a system that do not necessarily align with the system`s functional components bu tend to cut across group of functional component



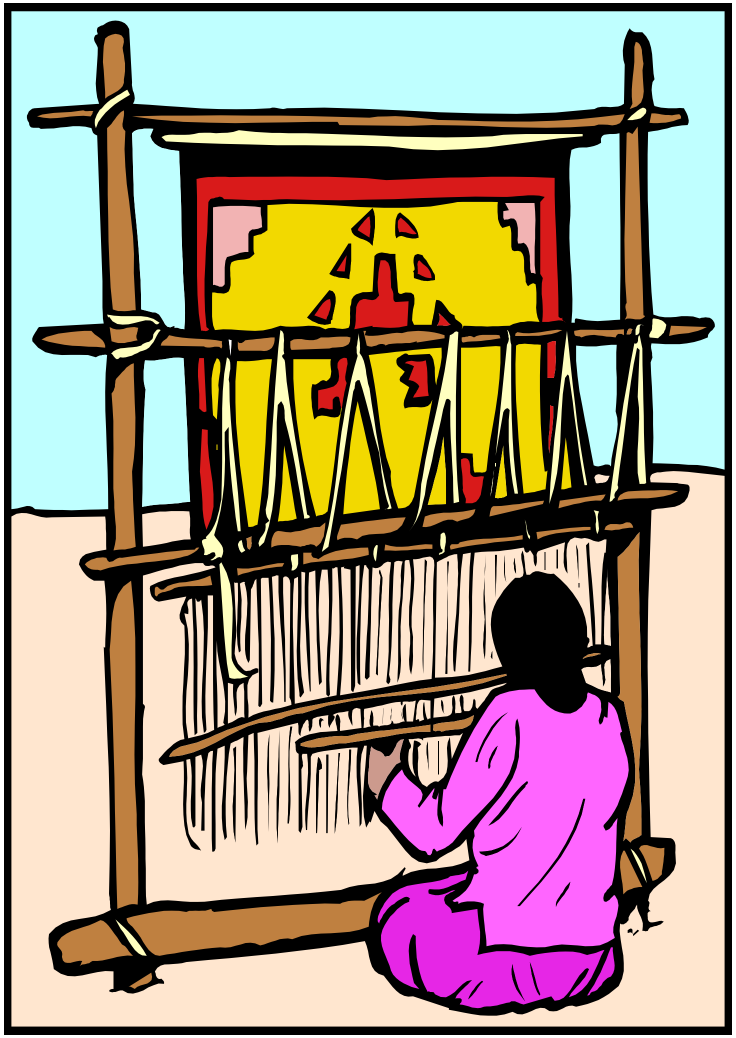
Aspect-oriented programming

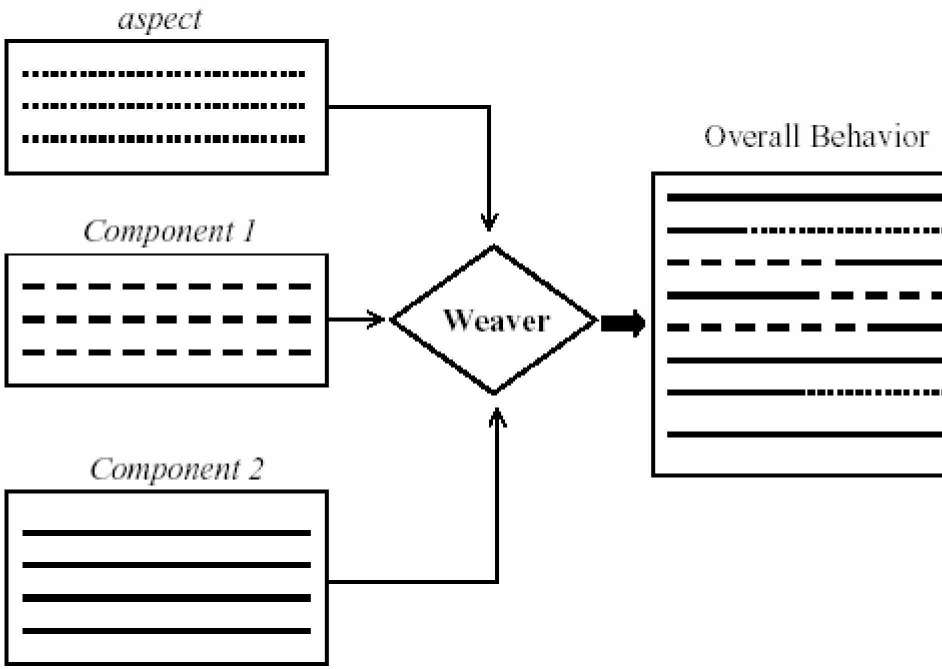
- Not bound to OOP, aspect-oriented programming retains the advantages of OOP and aims at achieving a better separation of concerns.
- Idea is to separate the component code from so-called aspect code
- Aspectual decomposition manages to achieve two dimensional separation of concerns
- At the implementation phase, aspects and components are combined together to form overall system

Architectural Issues

- Language support
- Static (automatic weaver) and dynamic weaving (reflective technologies)
- Code transformation
- Level of weaving
    - Pre-compile
    - Compile-time
- Open and closed implementations

Architectural Issues

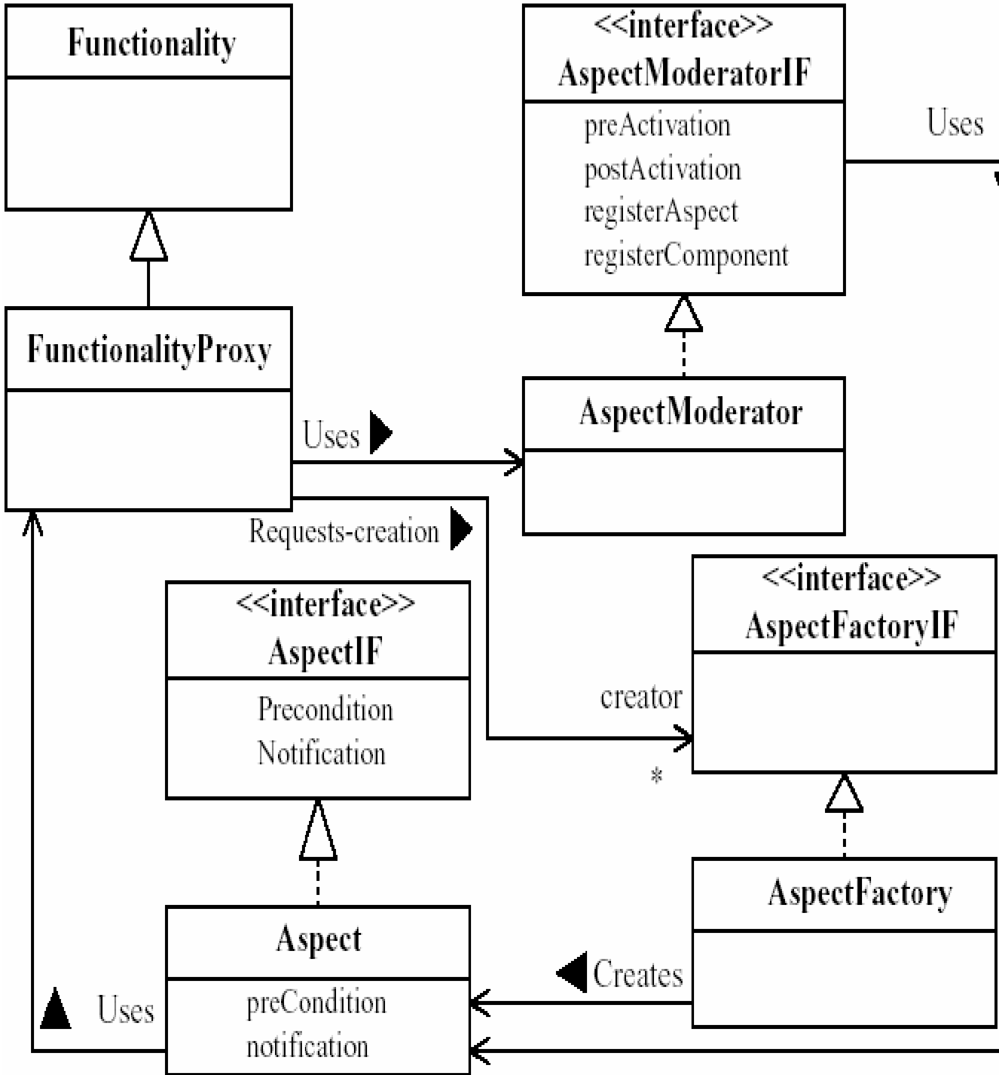




Weaving

Aspect Moderator Framework

- Proxy object controls access to functionality class
- Aspects are created using factory method pattern
- Proxy uses moderator object to evaluate the aspects for every method of functionaliy class



Aspect-oriented Framework

- Support separation of components and aspects from each other in different layers

- Three dimension model for system design

  - Components – basic functionality modules
  - Aspects – cross-cutting entities
  - Layers – components and aspects decomposed into more manageable sub-problems

Architecture of the Framework

- Base framework
- Application framework

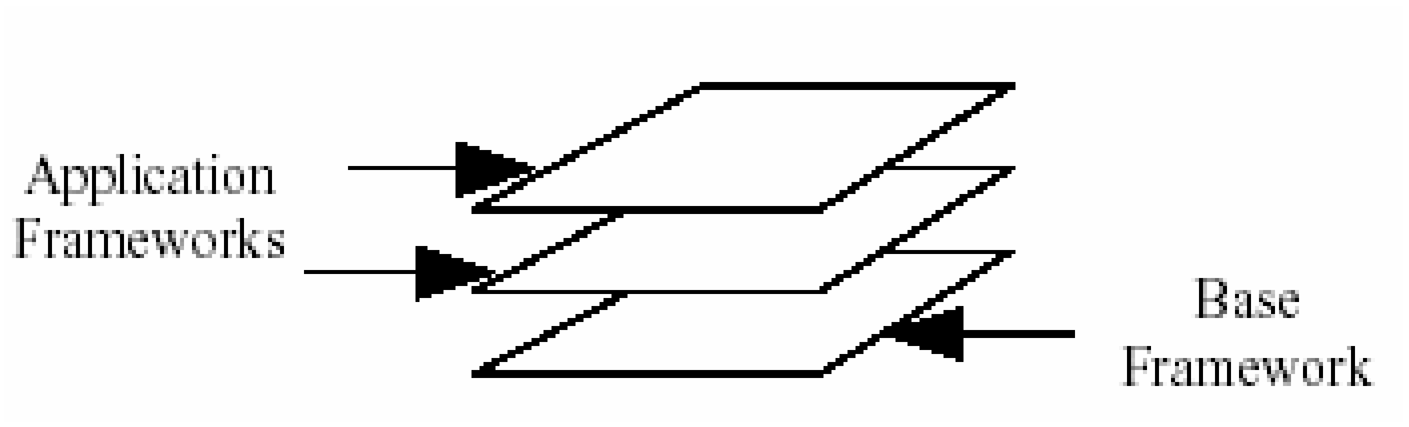


Figure – Aspect-oriented Design Framework

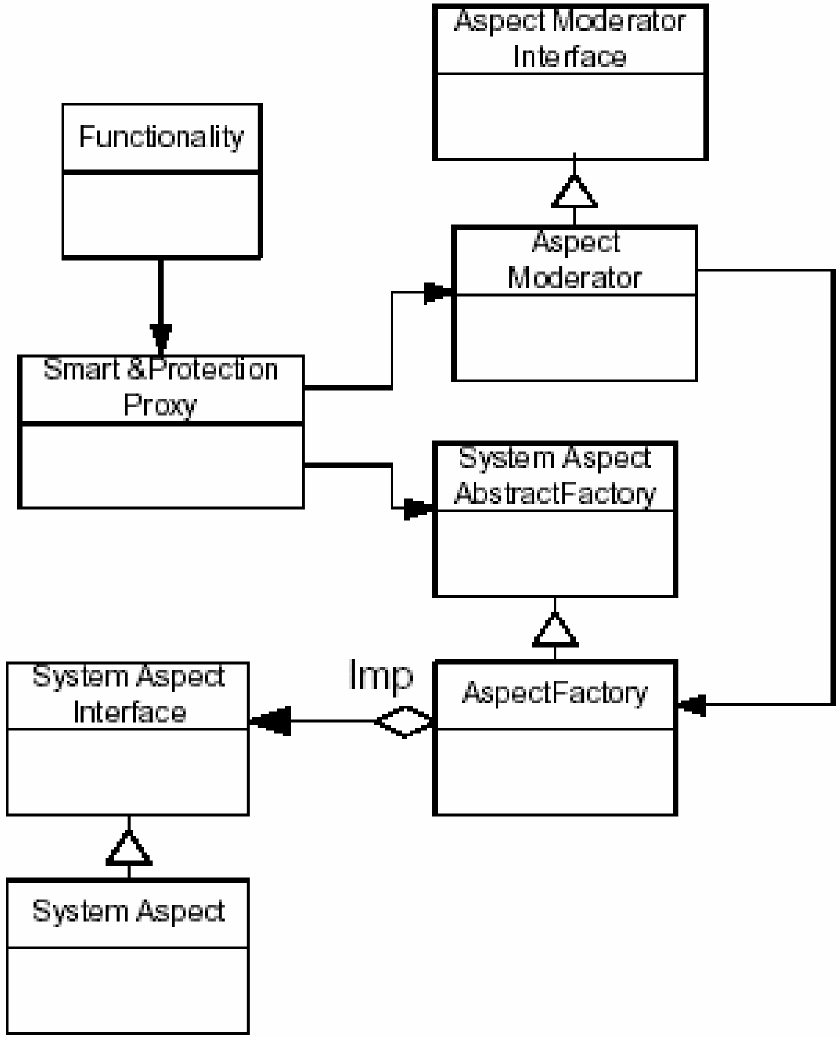
Architecture of the Framework

- **Abstract factory isolates aspects frm implementation classes**
- **Bridge pattern avoids a permanent binding between an abstraction and its implementation**
- **Smart protection proxy controls access to the aspects**
- **Adapter pattern allows aspect factory to either convert the interface of existing aspect or create a new aspect**

Execution Flow in Base Framework

Initialization phase

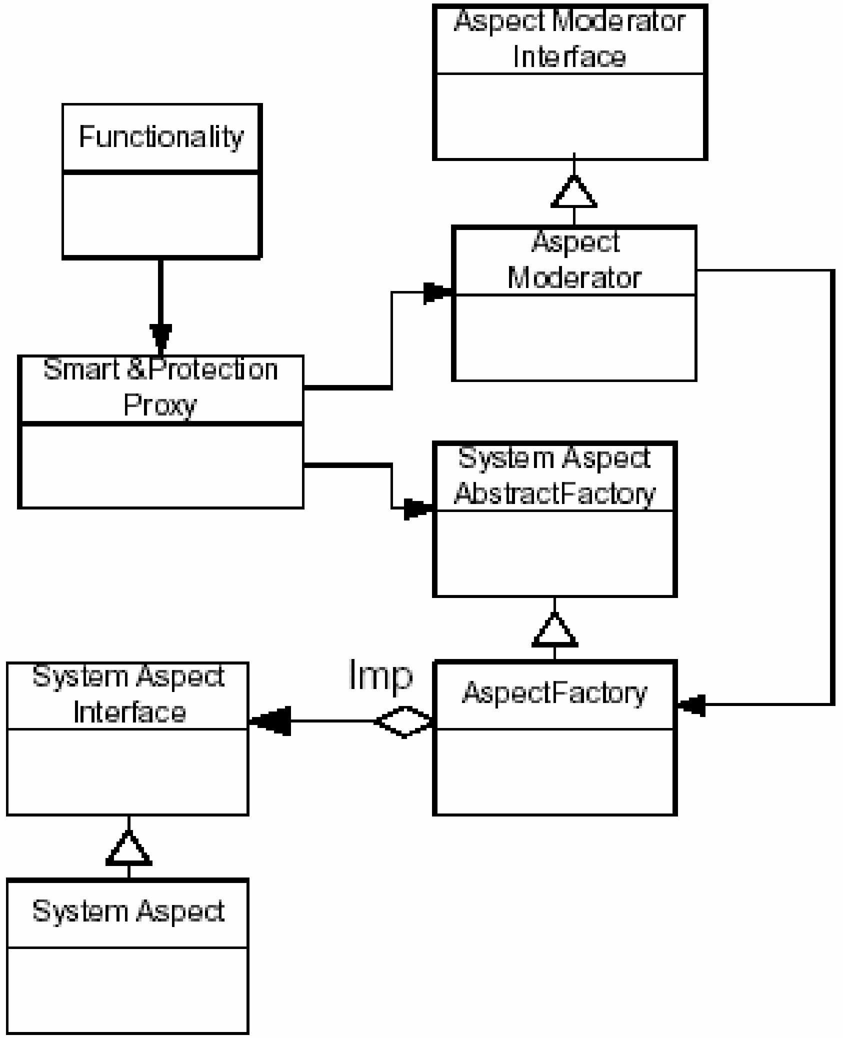
- Proxy forward request for aspect creation to AspectModerator object to find out if this aspect does not already exists.
- After verification proxy will call Aspectfactory to create the interface definition and the class definition of that aspect.
- Proxy will register both with AspectModerator



Execution Flow in Base Framework

Invocation phase

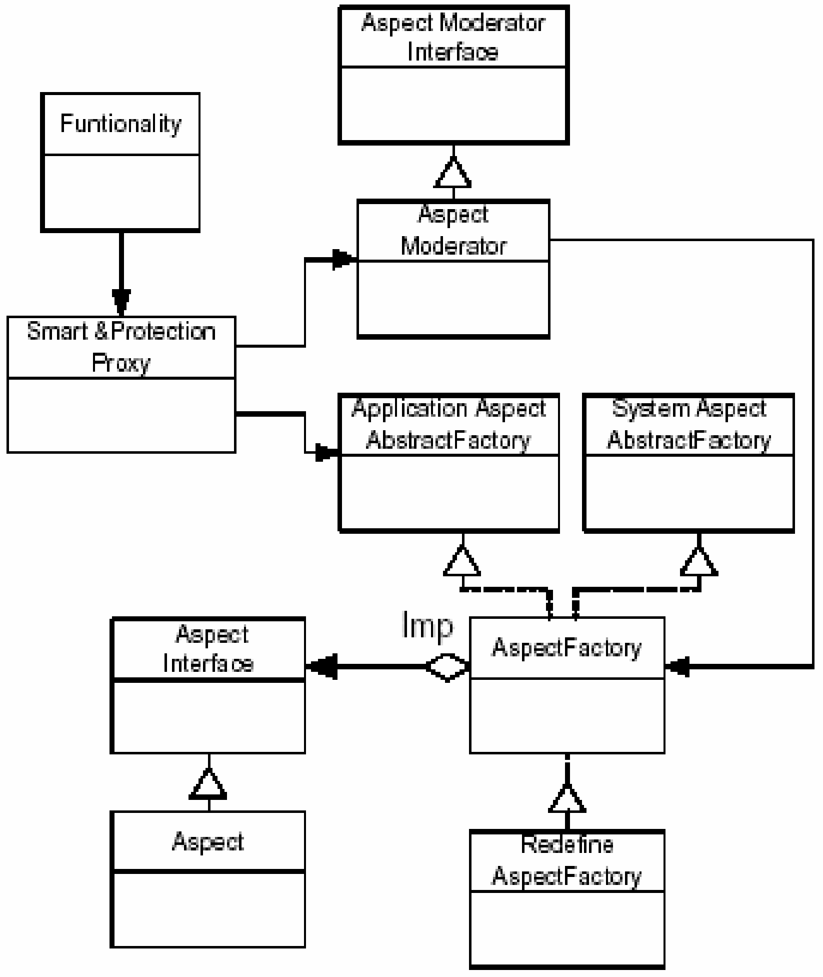
- Proxy checks whether an aspect that describes method‘s constraints is already registered with AspectModerator object
- AspectModerator will validate the constraints of the invocation method
- AspectModerator will activate the method of the aspect object and return control to the proxy.



Execution Flow in Application Framework

Initialization phase

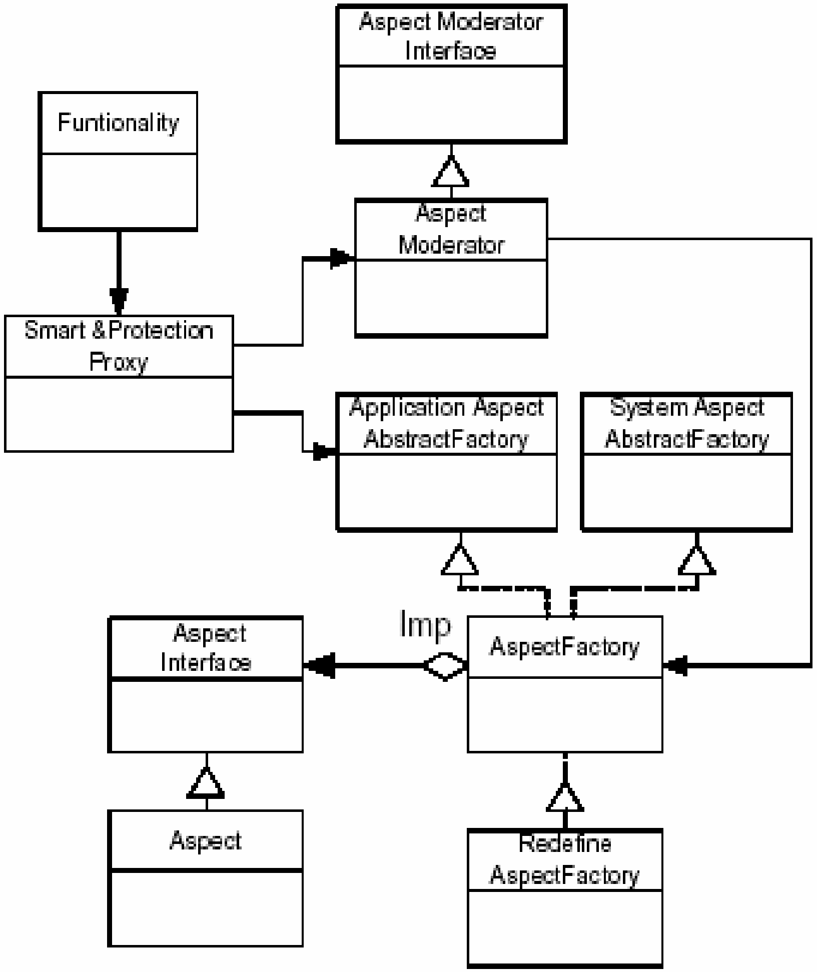
- Proxy recognizes if request is for aspect creation or method invocation
- Checks if aspect is registered with AspectModerator and which aspects in lower layer are included in the Application layer
- If aspect not registered then call Aspectfactory to create one and register with AspectModerator



Execution Flow in Application Framework

Invocation phase

- Proxy will check register at the AspectModerator. In case of no reference it will look up the lower layer.
- In case requested aspect not registered in neither layer, error is returned
- After successfull checking, the AspectModerator will validate the constraints of the method that is invoked and return control to proxy.



Framework Overview

- Three dimensional model
  - Collection of aspects
  - Components form the main functionality of OS
  - Layers are divided into three levels
    - Lower level – OS that provides reusable primitives for intermediate and upper levels
    - Intermediate level – system programming or interface definition
    - Upper level – application and programming level

Advantages from Framework

- Reusability
  - Upper level aspects or components using the lower level aspects or components
- Polymorphism
  - Avoidance of proliferation of functions
  - Provides generality of aspect
  - Makes easy to add new capabilities to an aspect
  - New aspect inherits from or override its super aspec
- Reconfigurability
  - Reconfiguration to appropriate policies

Summary and Conclusion

- Operating system should not be seen as a two dimensional model
- Complete separation of concerns
- Functional components and aspects are designed relatively separately from each other
- Framework provides an adaptable model that allows for open language
- Interactions of newly added aspects is defined by contracts

References

•Netinant P., C. A. Constantinides, T. Elrad, and M. E. Fayad, Supporting the Design of

Adaptable Operating Systems Using Aspect-Oriented Frameworks. Proceedings of the

International Conference of Parallel and Distributed Processing Techniques and Applications (PDPTA), pp.271-278, Las Vegas, NV, June 2000.

•C. A. Constantinides, T. Elrad, and M. E. Fayad, Netinant P., Designing an aspect-oriented framework in object-oriented environment,ACM Computing surveys, MArch 2000