

# Is Composition of Metaobjects = Aspect Oriented Programming

*Charlotte Pii Lunau*

*Department of Computer Science, Aalborg University  
Fredrik Bajers vej 7E, 9220 Aalborg, Denmark  
lunau@cs.auc.dk*

## **ABSTRACT**

This paper proposes to implement aspects as metaobjects and using run time aspect weaving. The metaobjects are metaobjects as defined in computational reflection and can be exchanged dynamically at run time. In contrast to computational reflection, where only one metaobject exist, we have extended the architecture to allow multiple metaobjects and composition.

## **1 INTRODUCTION**

This paper presents extensions to computation reflection, which redefines the causal connection and allows an object to have multiple metaobjects. The metaobjects are composed and they can be invoked either before or after method invocation in their base level objects. The architecture was originally developed to allow computational reflection in process control applications, and is further described in [Lunau 97]. The question is can this architecture be used in aspect oriented programming. ? First, the extended architecture is presented followed by a discussion of it use in aspect oriented programming.

## **2 AN EXTENDED COMPUTATIONAL REFLECTIVE ARCHITECTURE**

Traditional computational reflection [Maes 87] separates the base level of a computation from the metalevel. The metalevel is monitoring and influencing the base level. This separation is used to obtain a structured architecture with a clear separation of concerns. In traditional computational reflection, a base level object has at most one metaobject and there is a close relation between the object and its metaobject. The objects are co-ordinated through a causal connection. A causal connection implies that any changes made in a metaobject are immediately reflected in the base level object's state and behaviour.

## 2.1 COMPOSITION OF METAOBJECTS

Traditional reflective architectures allow each base object to have one metaobject attached. In process control applications, several independent aspects of behaviour need to be monitored simultaneously, such as fault detection, logging of values, and reconfiguration of the process. The monitoring can be performed by having a large set of metaobjects which contain combinations of the different aspects to be monitored. However, this will soon be incomprehensible. Composing metaobjects are a much more structured approach, where each metaobject is responsible for one aspect. Composition of metaobjects is implemented by attaching one system metaobject to a base object. The system metaobject administrates the composed metaobjects, and invoke them in turn. The system metaobject contains two lists. On the pre list are all metaobjects that should be notified before the method is performed in the base object. The post list contains the metaobjects that need to be notified after the performance of the method in the base object. A metaobject may be in both lists at the same time. The architecture is shown in Figure 1.

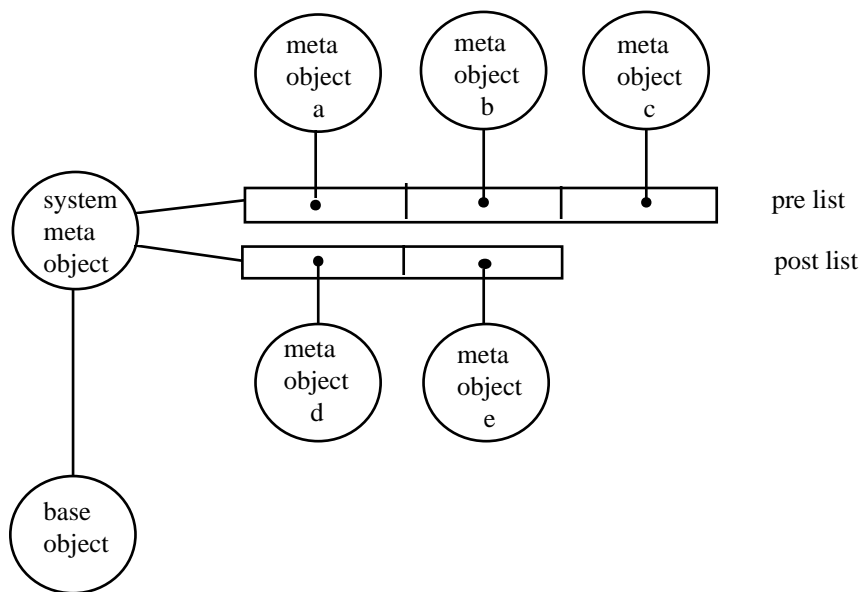


Figure 1 A meta architecture with composition of metaobjects

The system metaobject is invisible to the application. Metaobjects are attached and removed using the following statements:

```
[aBaseObject insertPreMetaObject: aMetaObject]
[aBaseObject insertPostMetaObject: aMetaObject]
[aBaseObject withdrawPreMetaObject: aMetaObject]
[aBaseObject withdrawPostMetaObject: aMetaObject]
```

When a metaobject is attached to a base level object, all messages send to the base object are intercepted. A metaobject decides which messages to handle and metaobjects attached to the same base object can handle different messages.

## **2.2 CAUSAL CONNECTION**

In process control applications, it does not make sense to talk about a causal connection between a base-level object and a metaobject. The directly monitored base-level objects are sensor objects and their is no way to dynamically change their behaviour. However, if we look at a physical system in a greater perspective, we are monitoring the entire process by selecting a set of sensor objects and we are affecting its behaviour through actuators. The way the physical process is affected is dependent on the metaobjects.

## **2.3 ARCHITECTURE SUMMARY**

The proposed architecture is highly dynamic. The metaobjects may exchange themselves with other metaobjects. Furthermore, it is possible to develop new metaobjects or to correct errors in existing metaobjects and to install them on the fly. For a process control application, this is an important feature because it is usually expensive to close down the system for software updates.

The possibility for composing metaobjects makes the architecture extendible. New metaobjects can be developed and installed without affecting already existing metaobjects.

## **3 ASPECT ORIENTED PROGRAMMING**

This section shows how aspect oriented programming can be mapped onto the extended architecture.

Metaobjects seem to be a natural place to implement aspects, because they are separated from the basic application and can be attached dynamically to any object in the application.

The causal connection as defined in computational reflection is neither needed in process control application nor in aspect oriented programming. In aspect oriented programming the affect of an aspect is dependent on the aspect itself. An aspect contains behaviour that cross-cut the functionality of an application. This behaviour may put constrains on the application, it may monitor its execution, or simple provide behaviour that otherwise would be scattered all over the application.

In the proposed architecture aspect weaving takes place at run time when messages are sent to an application level object, which has one or more aspects attached.

## 4 CONCLUSION

This paper has shown that aspects oriented programming can be mapped onto the extended computational reflective architecture. The architecture is dynamic and extensible because metaobjects may be defined, attached, and exchanged at runtime. However, the aspect weaving process is constrained to take place at runtime when messages are sent.

## REFERENCES

- [Lunau 97] Charlotte Pii Lunau:  
A Reflective Architecture for Process Control Applications  
Proceedings of ECOOP '97 p. 170 - 189  
*Lecture Notes in Computer Science, no. 1241*  
Springer-Verlag 1997
- [Maes 87] Pattie Maes:  
*Computational Reflection*  
Ph D. Thesis Technical Report 87-2  
Artificial Intelligence Laboratory  
Vrije University Brussel, 1987