

DYNAMIC FRAMED ASPECTS FOR POLICY DRIVEN AUTO-ADAPTIVE SYSTEMS

Philip Greenwood and Lynne Blair

Computing Department, InfoLab21, Lancaster University, Lancaster, UK, LA1 4WA.

Key words to describe the work: Dynamic AOP, Frame Technology, Reflection, Adaptive Systems, Policies, Parameterisation

Key Results: A framework is described that can be applied retrospectively to a system to provide auto-adaptive behaviour, by using parameterised dynamic aspects to encapsulate the adaptations necessary and using policies to describe the desired behaviour. It also improves specification of system behaviour with the inclusion of relationship definition.

How does the work advance the state-of-the-art?: Parameterising the adaptations to be applied to a system improves their reuse and, by delaying the generation of the concrete code to run-time, allows the code to be customised to the run-time conditions of the system.

Motivation (problems addressed): To improve the reuse and flexibility of adaptations used and allow correct system behaviour to be clearly defined.

Introduction

Modern applications often have to operate in environments that are susceptible to change. There are numerous properties that could vary during an applications run-time, from the available hardware resources to the user requirements. For a system to continue operating optimally, it is vital that the system adapts to suit the current conditions.

A traditional approach in adapting a system [2] is to stop the system, apply the changes and then restart the system. Critically, any stopping of the system is undesirable for systems that require high levels of availability such as safety critical systems.

Recently focus has shifted on to *auto-adaptive* systems. These are systems that are able to perform the adaptations automatically by monitoring the system conditions and deciding themselves when the most appropriate adaptation should occur.

Aspect-Oriented Programming

AOP is now a common approach to system adaptability due to its ability to allow code to be woven retrospectively. AOP [3] extends Object-Oriented Programming (OOP) by encapsulating concerns that would normally crosscut several objects. These crosscutting concerns are encapsulated in units of code called aspects and rely on constructs called joinpoints to define where the aspect code interacts with the OO code. The aspects are applied to the OO code using a process called *weaving*. Dynamic AOP can weave the aspect code at run-time – a useful property for adapting system behaviour dynamically.

Our Approach

The key focus of this work is applying flexible auto-adaptive behaviour to a wide range of systems retrospectively. We are proposing to develop a framework that uses dynamic AOP and parameterisation via frame technology [1] to achieve this. We believe that this solution will allow us to overcome some of the limitations encountered in earlier work such as reuse and customisation, whilst still achieving high degrees of flexibility and dynamic capabilities. By implementing the solution as a framework adaptive behaviour can be applied easily to a wide-range of systems.

Framed Aspects

Framed aspects are the amalgamation of frame technology [1] with aspect-oriented programming. Frame technology was originally conceived in the late 1970s to provide a mechanism for creating reusable segments of code using a combination of meta-variables, code templates, conditional compilation and parameterisation for use in software product lines.

Figure 1 shows the three elements required to generate a customised aspect:

- Framed Aspect – contains the generalised aspect code using the elements mentioned earlier, such as conditional compilation, parameterisation etc,
- Composition Rules – defines possible legal aspect feature compositions, combinations and constraints. These rules are responsible for controlling how the aspects are bound together,

- Specification – contains the developer’s customisation specification. This is used to set the values of the meta-variables and select the variation options available in the framed aspect.

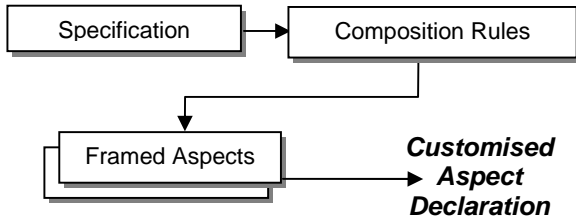


Fig. 1. Framed aspect composition

Traditionally framed aspects have been statically used in that they have been generated and processed at compile time. Our approach involves delaying the creation of the customised aspect declaration to run-time whereby using reflection and other system probes we can gather run-time data regarding the system status. This data can then be used to create the specification to generate a concrete aspect declaration that will be specifically customised for the current run-time conditions of the system.

Using framed aspects in this way allows the system to be adapted in a very flexible manner. For example using conditional compilation the current CPU usage could be used to select the most appropriate image compression algorithm to be included in the generated aspect. Or, using parameterisation, the amount of free memory could be used to create a cache of the most optimum size. We can generate code customised to the specific conditions of the system from generalised code that is highly reusable.

Policies

In order to apply the customised aspects to the correct points, at the correct times, it is necessary to define this behaviour.

Our solution to this is the use of policies based on Event-Condition-Action rules. In our policies the *event* is some joinpoint being reached. Each time an event is triggered the *conditions* associated with it are evaluated; if they evaluate to true then the *actions* are executed. There are a number of system attributes that can be used to make up the conditions ranging from hardware attributes such as memory usage or network bandwidth to system specific attributes like field or parameter values. The actions defined in the policies will always involve the weaving of some framed aspect or the removal of a previously woven aspect. Figure 2 shows an

example policy which defines the behaviour that specifies when the execution time of the method with the signature `autoadaptive.Client.get(String)` exceeds 500ms, an aspect should be generated from the Cache framed aspect. To prevent the cache from having a negative impact on the system the policy also specifies the aspect that should be removed if the amount of free memory ever drops below 50Mb.

```

<policy name="ResponseTime">
  <condition attribute="ExecutionTime" condition="more-
  than" value="500"/>
  <points expression=" execution(*
  autoadaptive.Client.get(String))"/>
  <aspect-type type="Cache" frame="true"/>
  <remove-condition attribute="FreeMemory"
  condition="less-than" value="50"/>
</policy>
  
```

Fig. 2. Policy example

Each type of framed aspect also needs a helper class to specify the system attributes that should be used to make up the specification used to generate the concrete aspect. Figure 3 shows how the amount of free memory can be bound to the `CACHE_SIZE` variable used to generate the cache aspect. Once generated, the caching aspect can be woven dynamically to affect the system behaviour immediately.

```

set= objFactory.createFrameTypeSetType();
set.setVar("CACHE_SIZE");
int size= MemoryInformation.getFreeMemory();
set.setValue(size);
setList.add(set);
  
```

Fig. 3. Code to bind the amount of free memory to the `CACHE_SIZE`

Summary

This paper has briefly described our approach for applying auto-adaptive behaviour retrospectively. The use of framed aspects allows us to reuse code that adapts system behaviour, and also allows us to generate customised code to suit the current run-time conditions of the system they are being woven to. To apply the framed aspects at the correct points and times we use ECA based policies to define this behaviour.

References

1. Basset, P. “Framing Software Reuse – Lessons from Real World”, Yourdon Press Prentice Hall, 1997.
2. Gupta, D., et al, “A Formal Framework for On-Line Software Version Change”, IEEE Transactions on Software Engineering Vol. 22 No. 2, 1996.
3. Kiczales, G., et al, “Aspect-Oriented Programming”, Proceedings of ECOOP ’97 pp. 220-242, 1997.