An Introduction to Filter Objects

Rushikesh K. Joshi
Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

Introduction

Filter Objects are first class objects. They can be passed and returned as parameters, they have identities, and they can be created dynamically. They are modular: they are instance of classes. Filter objects have special capabilities to transparently intercept message sent to server objects related to them. Filter objects, themselves being instances of classes, can also receive direct messages just as other first class objects do. They can organize their independent activities cohesively. It is also possible to network filter objects for a collaborative filtering activity. In this document, we discuss filter constructs in a language independent way. Filter constructs have been implemented for C++ [1], JAVA [2] and MICO CORBA [3]. These implementations have specific syntactic manifestation resulted in a bid to homogeneously integrate them in their respective parent programming languages.

The filter object constructs are primarily composed of an *interclass filter relationship, filter member specifications,* and a *filter binding mechanism.* The interclass filter relationship makes it possible to specify filter classes and eventually filter objects. A filter class specifies filtering member functions which filter their respective server member functions when a filter instance is plugged to a server object at run time. Filter binding mechanism facilitates dynamic plugging and unplugging of filter instances (filter objects) to their respective filterable servers. There are other extended filter object constructs which provide additional capabilities to filter objects. For example, filter objects can be layered, they can be made to filter a group of server objects. It may also be possible to switch filter member function implementations at runtime. A Filter object implementation integrated in an object oriented programming language is required to support the primary constructs. Extended constructs enhance the filtering facility.

The constructs described below are not exact syntactic representations in programming languages. The respective constructs in C++, JAVA and over MICO CORBA are close to these, with parent language specific variations.

Interclass Filter Relationship

Filter objects are instances of filter classes. A filter class is a class in filter relationship with another class. If class F filters class C, instances of class F can be made to transparently intercept instances of class C.

A Server class Description	*A Filter class Description*	*Code that creates server and filter objects and binds the filter with server instance. Method invocations on the server instance go through the plugged filter object*

class C {
...description of class C
}
	class F filters C {
	.... description of class F
	}
		...
		aServer <-- new C;
		aFilter <-- new F;
		plug aFilter to aServer;
		aServer.m()
		...

Figure 1

Filter Member Function Specification

A Filter class is like any other class, plus the filtering members. (At a meta level, one can think of a class FilterClass as a subclass of class Class.) Objects knowing the identity of a filter object may invoke public member functions on it. The filter member functions of a filter object are invoked by the runtime execution environment as demanded by filter bindings.

Upward and Downward filtering

Filter members may be specified to filter messages in upward and downward direction. An upfilter member function filters a server member function when the message in on its way to the server for an invocation. A downfilter member function filters a return result from an invocation of a member function at the server.

An Upfilter Member

An example specification of an upfilter class is given below:

class C {
private ...
public m(...) {.....}
public ...
}
	class F filters C {
	private ...
	public ...
	upfilter f filters m (...)
	{....}
	}

Figure 2

In the above code, filter class F defines an upfilter member function f to intercept invocations of member function m on instance of server class C. Note that F is specified to be a filter class of C. The *automatic* invocation of f on a plugged instance of F such as aFilter in Figure 1, receives all the input arguments intended for the invocation on its server instance. These arguments may be modified by the intercepting upfilter member function.

pass and bounce actions

An upfilter member function may decide to pass on the invocation to the intended server or bounce a result to the caller. These are called *pass* and *bounce* actions. An upfilter member function either invokes a pass (invocation) action, or a bounce (result) action. The following code demonstrates two filter members, each acting in one of the two modes.

<table>
<tr>
<td><pre>class F filters C {
private ...
public ...
upfilter f filters m (...) {
      .... take intermediate actions
      ...... possibly modify arguments
      pass;
}</pre></td>
<td><pre>class H filters C {
private ...
public ...
upfilter f filters m (...) {
   .... take intermediate action
   if (some condition) pass;
   else {
        compute a result to be bounced;
        bounce (result);
}</pre></td>
</tr>
<tr>
<td>A filter member that passes on the invocation</td>
<td>A filter member that dynamically chooses between pass and bounce actions</td>
</tr>
</table>

Figure 3

A Downfilter Member

A downward filter member specification is given below:

```
class F filters C {
....
downfilter g filters m (..) {....}

}
```

Figure 4

A downfilter member is invoked by the runtime environment once the intended server member function invocation returns a result and before the result is passed on to the client. A downfilter may modify the return value.

An Example

The following class diagram depicts three classes. Class Dictionary is a server class for class Client. Class Cache filters class Dictionary. An invocation on an object of class Dictionary passes through an instance of class Cache plugged to the instance of class Dictionary. The client code remains unaware of the intermediate filtering action. The class diagram uses an extended modeling language to include notations for representing interclass filter relationship and upfilter and downfilter interfaces.

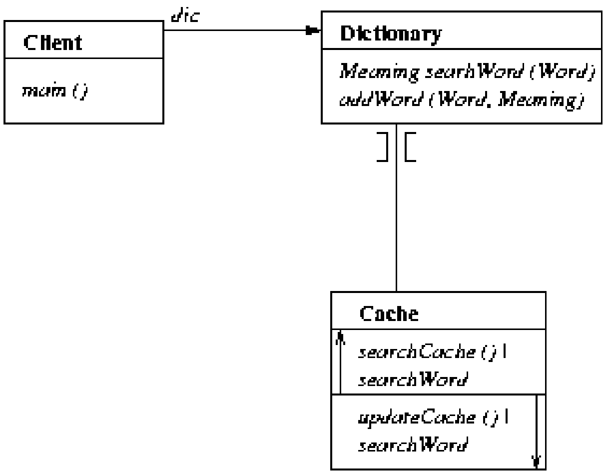

Figure 5

In the above example, whenever client code invokes Dictionary::searchWord(), an intermediate invocation of Cache::searchCache() is triggered on the instance of Cache that is plugged as a filter to *dic*, an instance of Dictionary.
Similarly Cache::updateCache() is called on the plugged filter instance when searchWord() returns from instance *dic* with a result. For modeling of Object Diagrams and Interaction Diagrams, the reader is referred to [4] and [5].

Related Work

Examples of well-known filter applications are process filters found on Unix systems, transparent proxy caches on Internet, packet filters in communication, mail filters, security filters, transparent event loggers and online transparent virus scanners. Integrations of filtering mechanisms into programming languages, products and technologies have started emerging. An earliest filter implementation in a programming language is Composition Filters due to Aksit et al. for language Sina [6]. CORBA products such as ORBIX [7] support various filtering abilities. Today, the term *Aspect Oriented programming* [8] identifies several initiatives on *separation of concern methodologies*. Filter objects can be viewed as an Aspect Oriented Programming paradigm, integrated in object oriented programming frameworks.

References

[1] Rushikesh K. Joshi, N. Vivekananda, D. Janaki Ram, Message Filters for Object Oriented Systems, Software Practice and Experience, June 1997, Vol. 27, N.6, pp. 677-700.

[2] Maureen M., Rushikesh K. Joshi, Filter Objects for JAVA Programming Language.

[3] G. Srirami Reddy, Rushikesh K. Joshi, Filter Objects for Distributed Object Systems, Journal of Object Oriented Programming, January 2001.

[4] Rushikesh K. Joshi, Filter Configurations for Transparent Interactions in Distributed Object Systems, Journal of Object Oriented Programming, June 2001, pp. 10-16.

[5] Rushikesh K. Joshi, Modeling with Filter Objects in Distributed Systems, Proceedings of the 2nd Workshop on Engineering Distributed Objects, (EDO 2000), UC Davis, Nov 2-3 2000, Lecture Notes in Computer Science, Vol. 1999, pp. 182-187, Springer-Verlag.

[6] Lodewijk M. L. Bergmans, The Composition-Filters Object Model, Ph.D thesis, University of Twente, June 1994.

[7] IONA Technologies Ltd., Orbix Advanced Programmer’s Guide, 1995.

[8] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, Aspect Oriented Programming, Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241. June 1997.