Data Filter Architecture Pattern

Robert Flanders and Eduardo B. Fernandez

Dept. of Computer Science and Eng,.
Florida Atlantic University
Boca Raton, FL 33431

Abstract

Given the potentially enormous amount and variety of data available in the Internet and other systems, it becomes necessary to filter out part of it for institutional or legislative reasons. We present a filtering architecture that can apply different filtering policies.

Intent

Filters the contents of client requests in a distributed system, according to predefined policies. Filtering can occur locally or remotely.

Motivation

Problem

In many distributed systems, e.g., the Internet, requests for services or data need to be filtered according to institution policies, legislative restrictions, privacy needs, etc.

Context

We consider a distributed system where clients send requests for data or services. Examples include CORBA or DCOM-based systems where users may ask for database items, the Internet, where browsers request web pages, specific types of Intranets, etc.

Forces

The following forces define this pattern:

- Data requested by clients needs to be filtered for different reasons.
- Filtering should not affect the normal operation of the system, i.e., the only noticed effect will be that some data or services are not available to some requests.
- The architecture should be scalable and modifiable with respect to the number and type of policies.
- The filter should be independent of the conten t provider structure and the network protocol.

Solution

Figure 1 shows the basic architecture for the proposed solution.

The Data Filter Architecture pattern partitions the middle tier of a three-tier architecture into a Filter Service administration component and multiple distributed remote Filter Agent objects executing on either the content providers data server or the remote clients server. This Filter service is based on reconfigurable Filter components that distributes a smart pipe (Policy Applicator) evaluation mechanism to optimize network utilization. Filter Components provide a repository for persistent local user-defined Filter Policies for each client. The Distributed Data Filter Architecture Pattern is independent of the content providers data and Network protocol.

The Collaborators are the Data source that delivers input to the processing pipeline, the Data Sink of consumer of data, the Filter Policy that performs a functional transformation and produces output and the Pipe that synchronizes active neighbors and transfers data. The Smart pipe combines the concept of pipe and Filter Policy applicator. The Smart pipe synchronizes active neighbors, transfers filtered data and performs discrete functional screening (filters) or format transformation to generate a modified stream.

The Distributed Data Filter allows the active content provider data source to be tailored to a particular client's specification without having to maintain a large client database or a large distributed software application. The Filter Agent Database maintains only the necessary Data Policies needed to tailor the Data Source for a particular Activate Policy of a Client. The Client receives a data stream, which contains only the data, which passes predefined Filter Policies or is a format based on the client’s criteria or format specification. The Filter Policy can contain a format or translation algorithm for specific types of data. The Filter Service provides centralized access control and persistent database for E-commence and re-programmability for the distributed data clients. The re-programmability of both individual Filter Policy and the Filter Policy evaluation Algorithm (rules based engine) provides an open standard for various behavior flexibly which is key to an Architecture that generalizes complex data manipulation and representations.

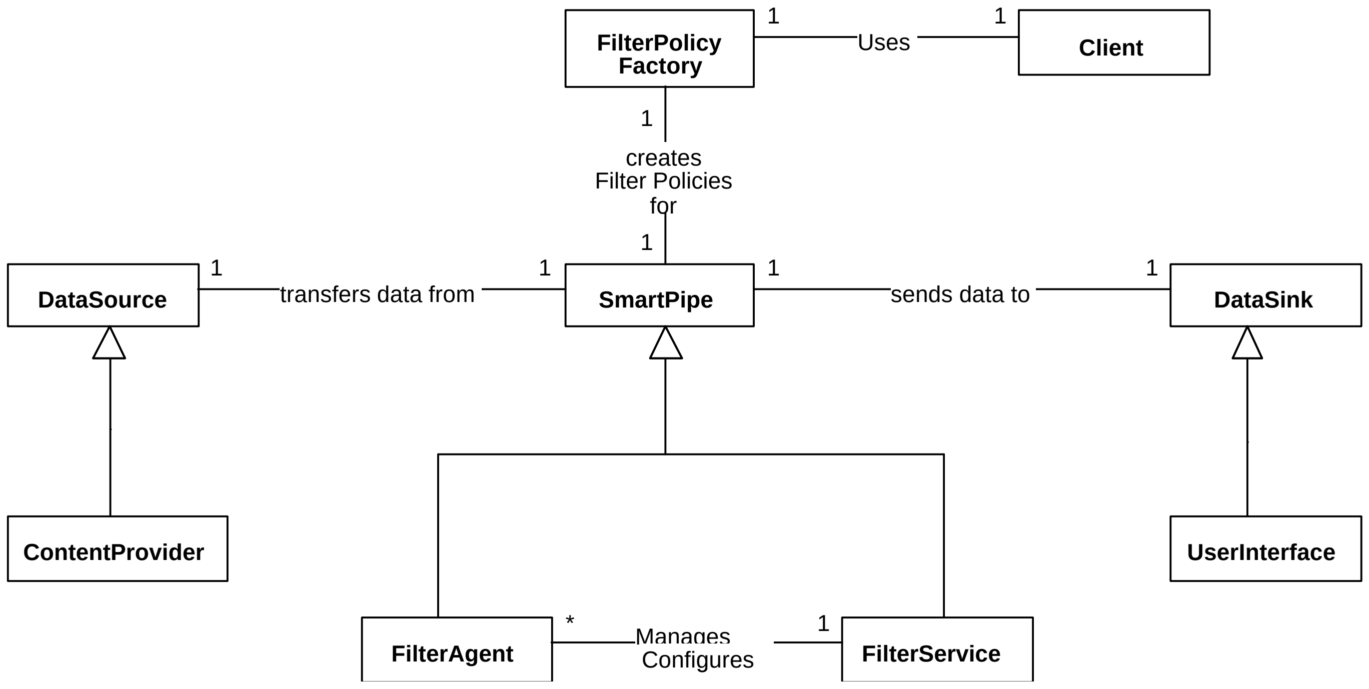


Figure 1 Basic pattern structure

Applicability

Use the Distributed Data Filter Pattern when:

- You are providing a service which needs to provide a client with data retrieved from remote sources and the data stream contains large amounts of unrelated, unwanted or secure data.
- You want to provide to a set of registered clients a tailored data stream from a well-known data service provider.
- You want the flexibility to change the content of the data stream by a reprogrammable strategy..
- You want to Distributed Filter Policy to reduce network traffic, evict unwanted packets or restrict the content of data stream contents available to a client.

Structure

Figure 2 shows the overall Distributed Data Filter Architecture Pattern utilizing a distributed Filter Service and Filter Agents

Filter Service Components can consist of two different types of middle tier objects [3]: a singleton [1] Filter Service Component and multiple Filter Agent components [1]. Each Filter

Service Component contains a reconfigurable database that contains multiple sets of Filter Policy objects that are iterativily evaluated to perform Distributed Data Filtering. The databases interface to the Back End third tier objects [3] may be integrated in the CORBA middle tiers' objects. (The databases maybe encapsulated in either CORBA database components or in Java components that support third-tier operations [3]. The singleton Filter Service Component executes on the Filter Services Server address space and maintains the master Filter Service Database.

The Filter Agent Component executes within the remote server address space or within a content provider’s server and maintains an abbreviated local database. The Filter Agent database contains only Data Policies needed to support specific user interfaces. The Filter Agent is the smart pipe that screens or transforms data supplied by the content provider Data Server Object [2]. The Filter Agent starts a Receive Data Thread and waits on a blocking call returning data or can create a Registered callback object if the Content Provider provides callback services.

The Filter Service Database is the master persistent repository for retaining the Filter Service Clients ID and account information regardless of the Clients Activation State. The Filter Service has the same smart pipe Filtering capabilities as the Filter Agent. The Filter Service is the central point of contact Filter service for the Filter Service Client.

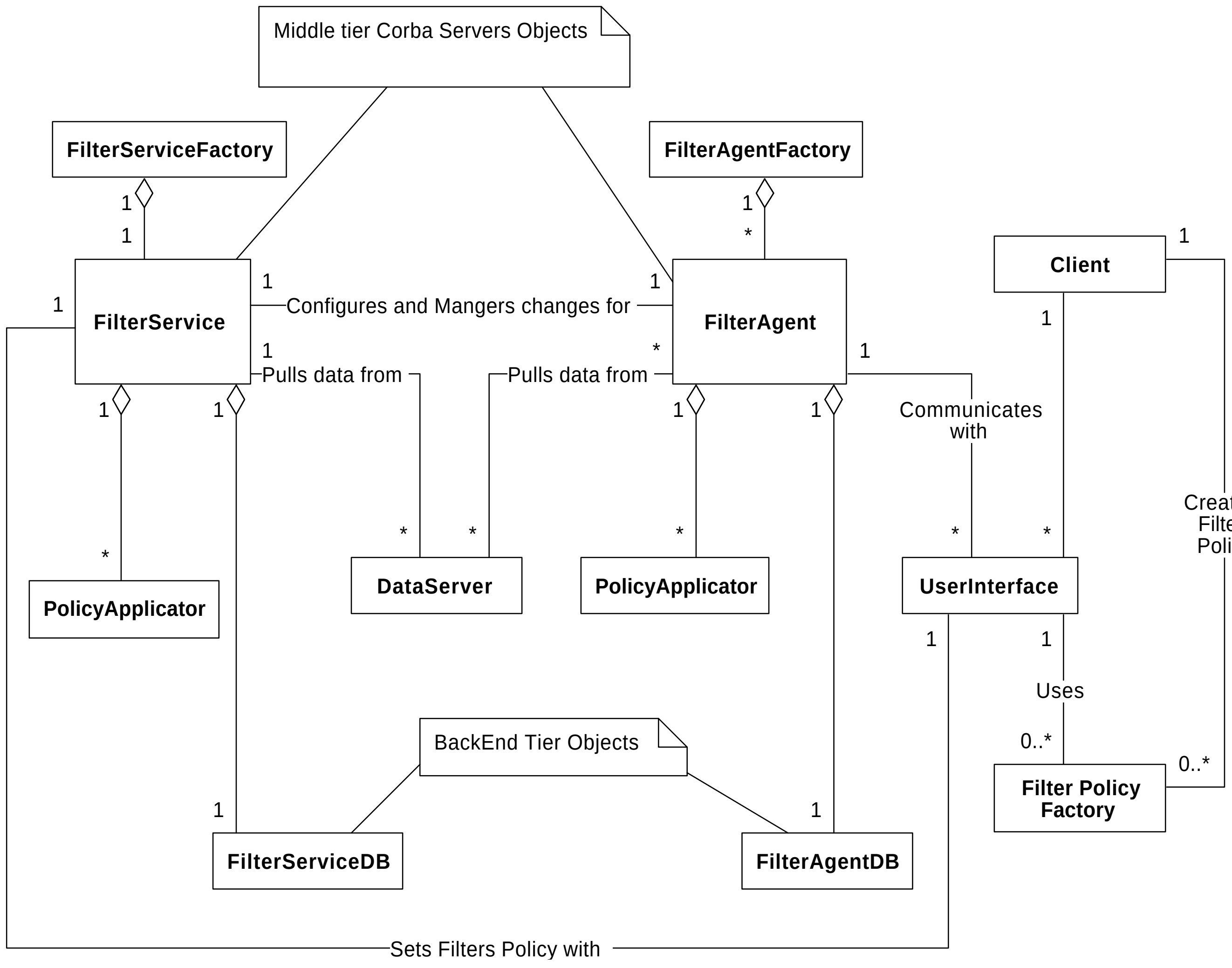


Figure 2 Distributed Data Filter Architecture Pattern

Participants

Client

A Client uses a Filter Policy Factory to create predefined filter policies and distribute them to Filter Agents. A Client may delegate some application user windows to interface with a Filter Policy Factory to create well-know polices or derive from the Filter Policy base class to handle unsupported filtering or data transformation.

User Interface

The User Interface may interacts with a Filter Policy Factory to create predefined filter policies or the application code may derive their own class from the Data Policies base class to handle all different unsupported filtering or transformation. This is the Client Web Tier of Objects that runs as a Java Applet and downloadable E-commerce Web based Application [3]. The User Interface resolves a reference to the well-known Filter Service object. The User Interface uses the Filter Policy Factory to create the Filter Policies that are used to create a Filter Policy List. Filter Policy Lists are sent to the Filter Service component and are retained in the Filter Service Database. The User Interface uses a Filter Policy Factory to create Filter Policies and may invoke Append, Delete, and Replace Policy operations on the Filter Service object or the Filter Agent.

The User Interface contains either a receive data thread or a receive data call back object that collects or displays the data stream. The is the Filter Service access end point (Data Sink) that consumes the filtered received data or re-distributes it to local or remote processes. The User Interface is a component of the Remote Filter Service Application executing on a Clients machine via an application or applet. The User Interface is the high level object that will be used to show interactions with the client. The Remote Filter Service Application has many low-level objects that are abstracted out of the model being present to clarify and simplify the interactions. The number of Filter clients is not restricted however each new client must subscribe with a globally unique Client ID to the Filter Service.

Filter Service

The Filter Service is the main repository for Filter Policies and list, client lists and is administrative and change manager for remote updates to Filter Agent’s residing on different servers Observer Pattern[1]. The Filter Service receives and processes Append Policy, Delete Policy and Replace Policy update requests from Filter Clients. The Filter Client issues Activate Policy and Deactivate Policy requests to the Filter Service those associates the specified Filter Policy List with the Client ID. The Policy list is a list of predefined policy objects that contain symbols, URL, src address, dest address and user defined data items of interest data from a generalized content provider’s data stream. The generalization of the data stream is a useful abstraction and will be referred to as the Data Server. All Remote and Local Filter Service clients interact with the Filter Service Object. Remote Filter Clients have Registered IDs with the Filter Service: the Filter Service updates the Filter Agent each Filter Policy transaction.

The Filter Service contains the policy list in a Master database that provides all filter policies for registered clients. The Filter Agent contains Policy lists needed to carry the specified Filter Policy that is invoked by the Activate Policy and Deactivate Policy operations. The number of Filter

Agent objects supported by the system is dynamic, because a Notification Handle container allows multiple Filter Agent objects to subscriber to the Filter Service Component. The locations of the Filter Agents objects are not limited to the Remote Clients Address space. Any server can create Filter Agents; all that is required is a reference to a Data Server, a TCP/IP address to send the data to and a Filter Agent Factory.

Filter Agent

The Filter Agent is a remote proxy that provides local representation for an object in a different address space [1]. The Filter Agent Filter Policy objects are remotely activated or deactivated by the Filter Service in response to incoming Filter Client requests. The Filter Agent database is maintained remotely by the Filter Service. The Filter Service creates deletion and appends lists for Data policies related to remote deactivation and activation respectively. Filter Service uses the Appends lists, as a pre-loading of the Filter Policy objects needed to support a pending Activation on the Filter Agent. The Deletion lists are used to clean up the Filter Agent repository after a Filter Client is deactivated. The Remote operation invocations on the Filter Agent are individual Append Policy method invocations followed by an Activate Request operation invocation. For Policy Deactivation, the Deactivate Policy is invoked, followed by the individual Delete Policy operation one per Deactivate list entry. The Filter Agent object contains a reference to the Data Server that provides the data stream to be filtered. The Data Server is an abstraction of any type of data stream; it can be a complex image file format or a simple serial communication protocol. The Filter Agent receives the data stream from the content provider and filters the data, then sends it to the Filter Client. The Filter Agent issues a callback to the Filter Client [2].

Filter Service Policy Database and Filter Agent Policy Database

The Filter Service and Filter Agent Policy database objects are encapsulated within the Filter Service object and the Filter Agent objects. The Policy databases or back-end Web objects [3] provide low-level Object Storage Filter Policy objects and per internal or external in a Distributed Data Filter System. In an Object Web-based version the Distributed Data Filter databases would be separate objects to take advantages of Component databases and vendor-supplied database management system.

Figure 3 shows the Filter Components and major top tier objects.

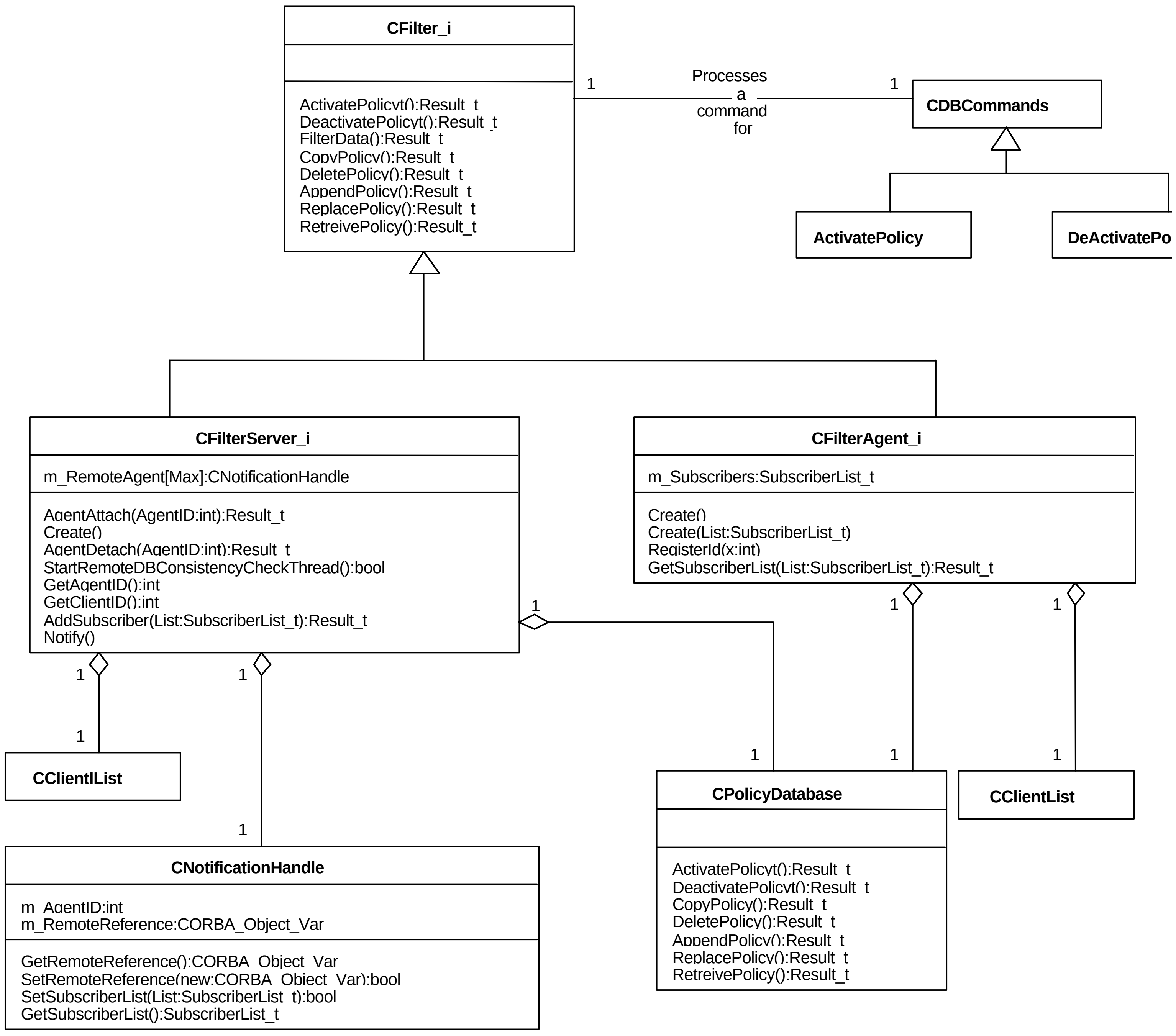
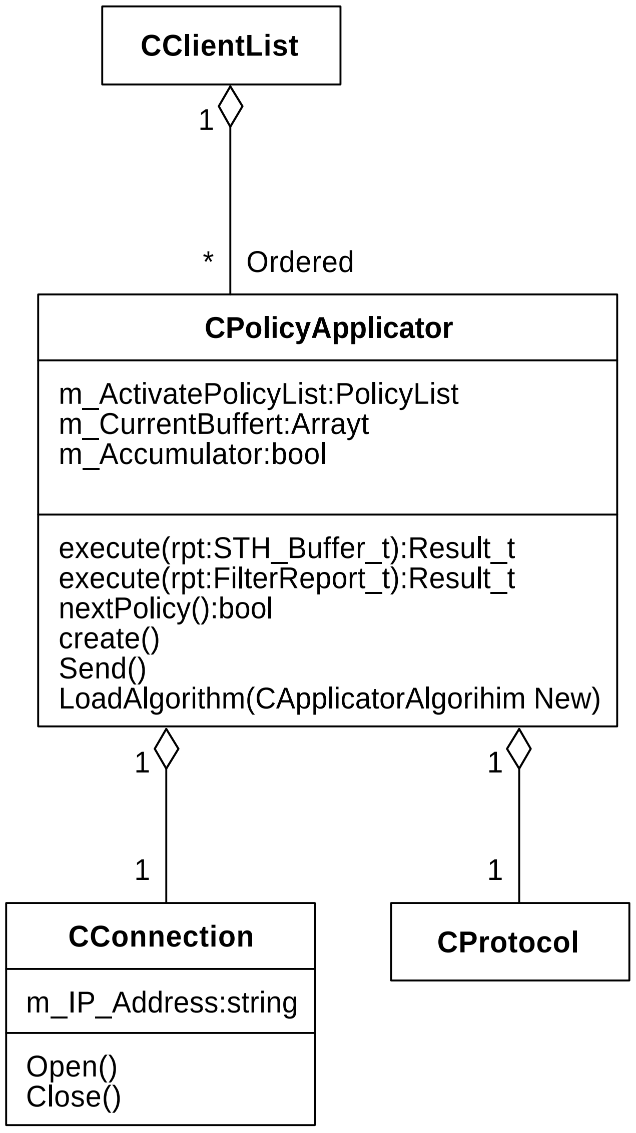

CFilter_i
ActivatePolicyt():Result_t
DeactivatePolicyt():Result_t
FilterData():Result_t
CopyPolicy():Result_t
DeletePolicy():Result_t
AppendPolicy():Result_t
ReplacePolicy():Result_t
RetreivePolicy():Result_t
1
Processes a command for
1
CDBCommands
ActivatePolicy
DeActivatePo
CFilterServer_i
m_RemoteAgent[Max]:CNotificationHandle
AgentAttach(AgentID:int):Result_t
Create()
AgentDetach(AgentID:int):Result_t
StartRemoteDBConsistencyCheckThread():bool
GetAgentID():int
GetClientID():int
AddSubscriber(List:SubscriberList_t):Result_t
Notify()
CFilterAgent_i
m_Subscribers:SubscriberList_t
Create()
Create(List:SubscriberList_t)
RegisterId(x:int)
GetSubscriberList(List:SubscriberList_t):Result_t
1
1
1
1
1
1
1
1
1
1
1
CClientlList
CNotificationHandle
m_AgentID:int
m_RemoteReference:CORBA_Object_Var
GetRemoteReference():CORBA_Object_Var
SetRemoteReference(new:CORBA_Object_Var):bool
SetSubscriberList(List:SubscriberList_t):bool
GetSubscriberList():SubscriberList_t
CPolicyDatabase
ActivatePolicyt():Result_t
DeactivatePolicyt():Result_t
CopyPolicy():Result_t
DeletePolicy():Result_t
AppendPolicy():Result_t
ReplacePolicy():Result_t
RetreivePolicy():Result_t
CClientList

Figure 3 Filter Components and Top Tier Objects.

Policy Applicators

Data filtering is performed using a dedicated internal active Policy Applicator object. The Policy Applicator is a managed resource that is dedicated to an Activate Policy on behalf of a Filter Client. The Policy Applicators are placed in a container indexed by Client ID. The Policy Applicator contains the iterative evaluation control for Filter Policies and logical expression and relation evaluation rules for Policy Applicator strategy. The evaluation algorithms can be independently interchanged, because of the utilization of the independent encapsulation of the evaluation policy using the Strategy Pattern. The Filter Client can select the evaluation algorithm from the Remote User Interface to achieve the desired behavior [1].

Figure 4 ClientList and PolicyApplicators



DataServer

The DataServer is an abstraction of a content provider data stream maybe co-located with the Filter Agent or within a remote data source. The Data Server provides the packets or content symbol units to be filtered. The specific format and content of the data are abstracted out to make the solution more general (Data Source). The Filter Agent creates a connection object and calls the open operation. The DataServer object contains a connection object that starts a Thread to transmit data to the Filter Agent. The Filter Agent sends the filtered data to the Client Window. The Filter Agent and Data Server create a Protocol object that works with the connection object to initiate an authentication sequence. After the authentication sequence is completed the Filter Agent pulls Data from the Data Server. The DataServer is assumed to provide the data continuously until the Filter Agent calls the close operation on the connection.

Filter Service Factory

The Filter Service Factory creates a singleton Filter Service Object on the Filter Service Server at startup by the Filter Service Application process [1]. The Filter Service Factory is an instance of the factory template.

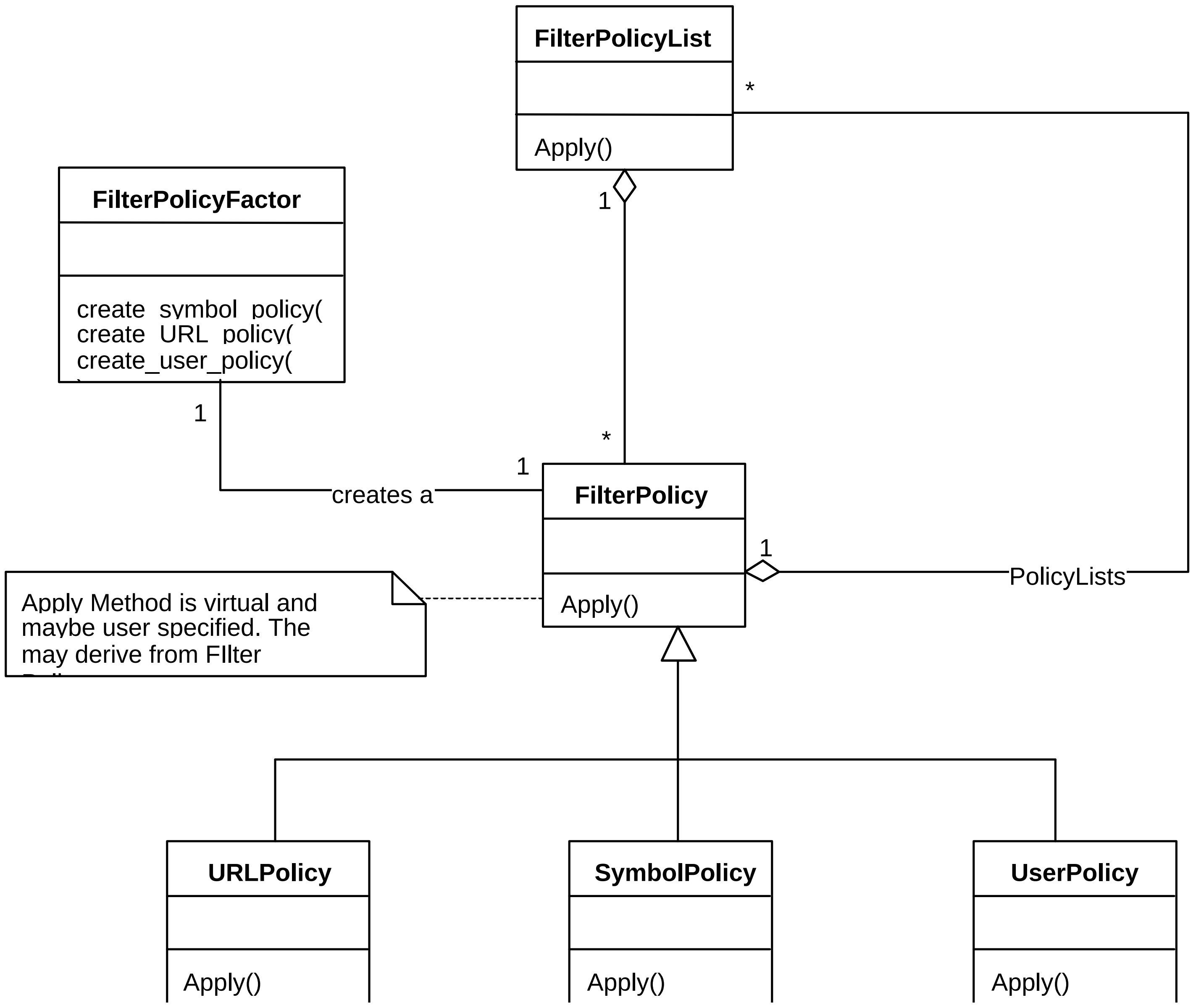
Filter Agent Factory

The Filter Agent Factory provides for the creation and initialization of multiple Filter Agent objects on the Content Providers Server or the Filter Clients Server [1]. The Filter Agent Factory is a persistent registered resource with the network Naming Service executing on the server. Upon the Remote Filter Service startup process a Filter Agent Factory is created to produce Filter Agent Object on Request [1]. creates the Filter Agent objects. The Filter Agent Factory is used to create multiple copies of the Filter Agent with either a predetermined subscriber list or a blank subscriber list. The Filter Client’s User Interface is register with the Filter Agent objects by invoking the GetClientID operation on the Filter Service object and registering it with Filter Agent by invoking the RegisterID operation. The RegisterID operation invokes the AddSubscriber operation on the Filter Service, which causes the Clients Filter Agent to be configured by the Filter Service.

Filter Policy Factory

The Filter Policy Factory is an abstract part of the Filter Agent, that allows the client to create, predefined and user defined Filter policies.The create policy operations are defined by the Filter Agent and the Filter Service Components and allow the creation of either predefined items or user-defined Filter policies. Each Filter Policy contains an apply method which can be overloaded to contain user defined behaviors.

Figure 5 Filter Policy Creation Objects and Storage Classes

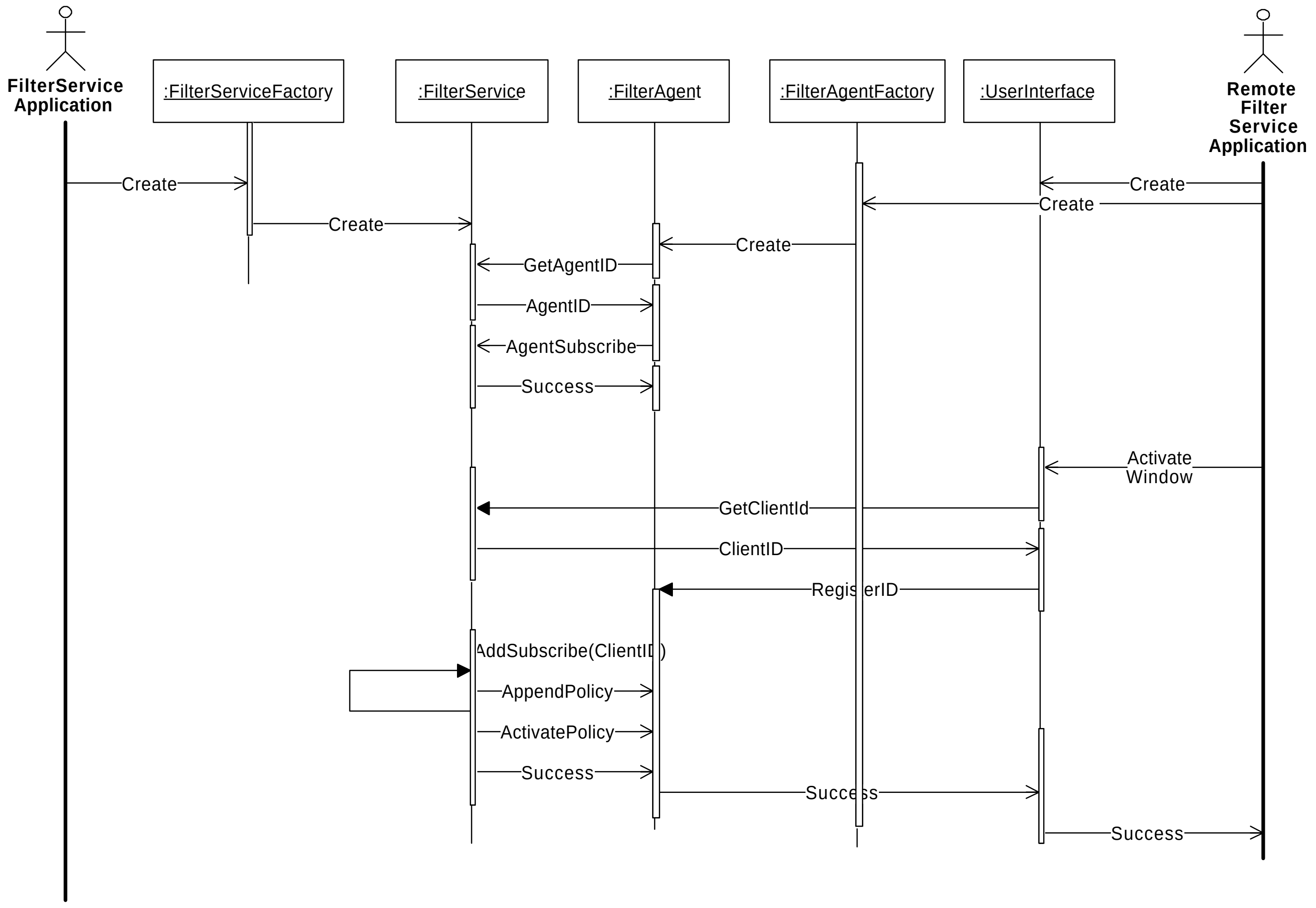


Collaborations

The Distributed Data Filter Architecture Pattern encapsulates many design patterns to provide a service that is flexible and reconfigurable. The following sequences and class interactions represent the key object s and collaboration needed to perform Distributed Data Filtering.

The Proxy Pattern is the underlying structural pattern that best describes the architecture utilized in the creation of the Filter Service Components. The two tiers of Filter Server Objects bridge the Address Spaces to provide a seamless acquisition of data from a content provider data server to a user customized local environment. The deployment of a efficient Distributed Data Filter System participates the used of a Remote Proxy as a Local Representative of the Filter Service in the Client address space hence the need for the Filter Agent. The Filter Agent only exists because of location and perhaps some memory optimization considerations. The Filter Agent maintains a reference to the Real Subject (Filter Service) by resolving the name with the naming service during the subscription process. The Filter Agent has the same interface as the Real subject (Filter Service) and is substituted for the Real Filter Service locally.

Figure 6. Filter Agent and Filter Service Creation



Remote and Local Policy Activation

The activation of a Client's Filter Policy may be done locally by directly connected network clients on the Filter Service Objects or Remote Filter Agents. For directly connected users of the Filter Service the activation is simply an Activate Policy operation invocation with no interaction between the Filter Service and the Filter Agent. The Activate Policy operation invocation creates an instance of a command object [1], which encapsulates all internal object creation and messaging required supporting both Remote and Local Activation's. The Activate Policy object creates a persistent Policy Applicator object and places it into the Client List. For Remote Activation's the Activate Policy object invokes the CreateAppendList and RemoteAppend operations. The Filter Service creates an AppendList and Append Policy objects to the Filter Agent database that will be needed to support the pending activation. The final operation invocation by the Filter Service is the Activate Policy operation on the Filter Agent that executes as defined above. The sequence diagrams in Figures 4 and 5, show the remote and local activation of Filter Clients, respectively.

Figure 7 Remote Activation

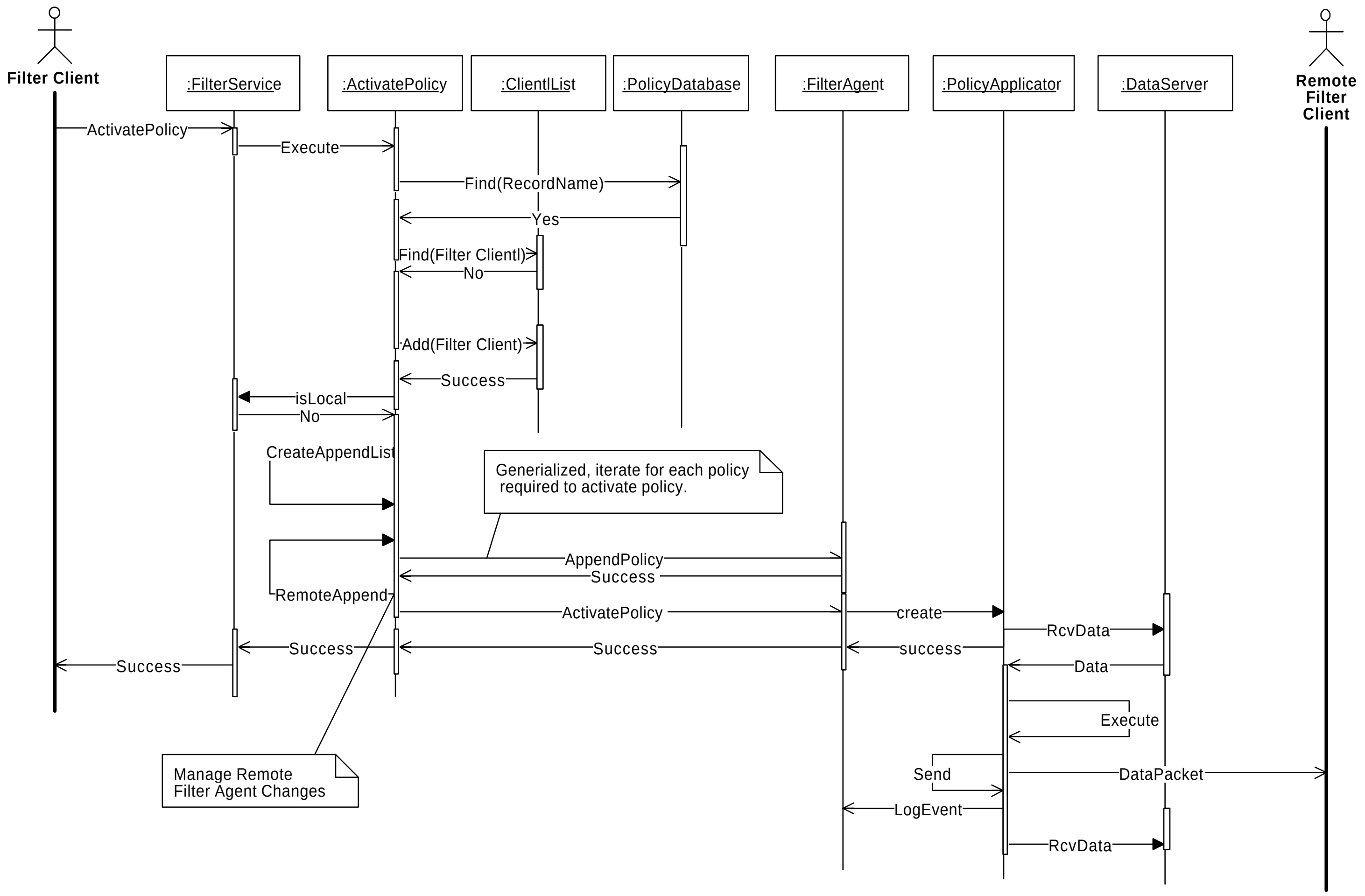
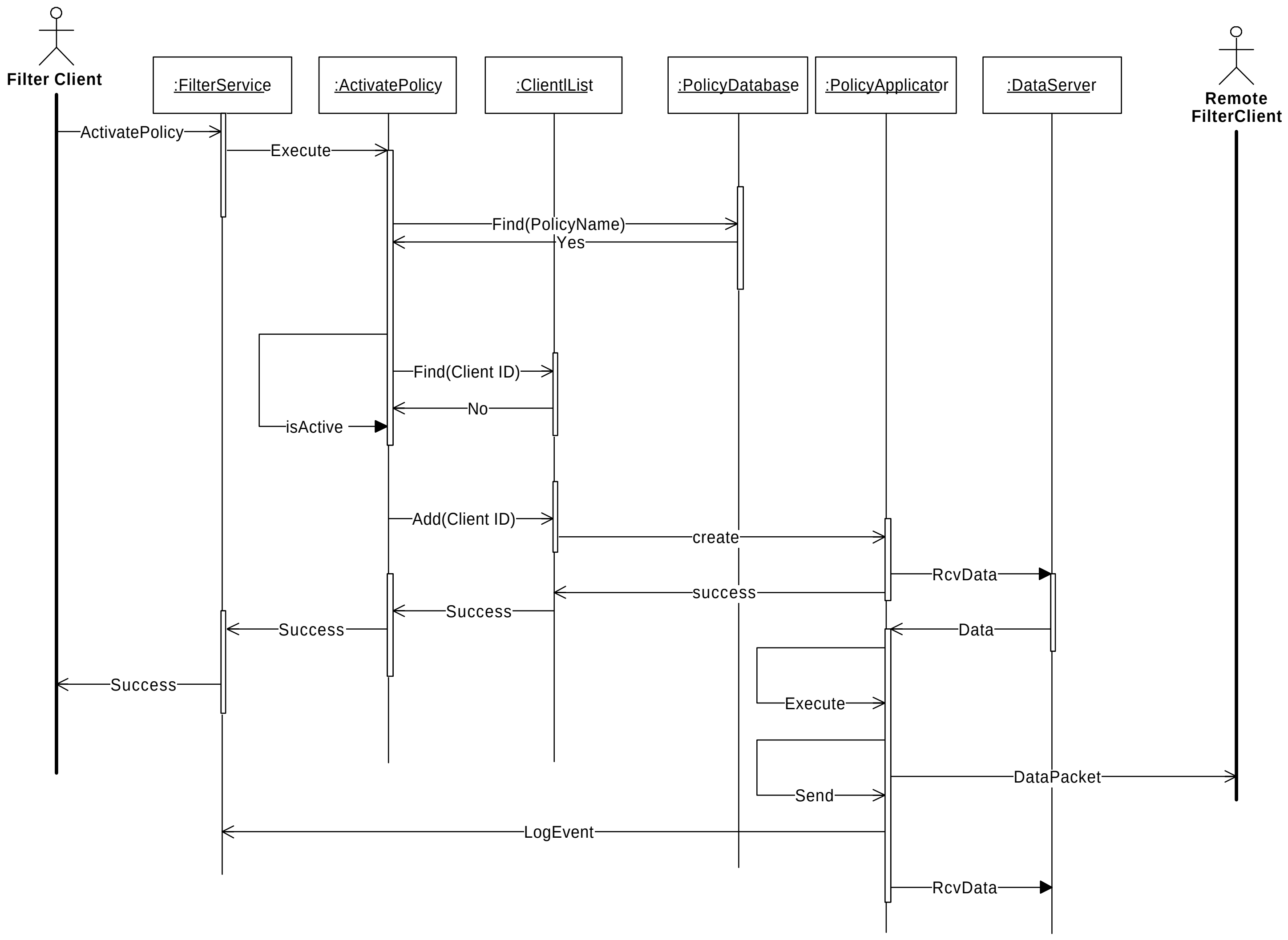


Figure 8 Local Activation



Remote and Local Policy Deactivation

The Deactivation of a Client Filter Policy requires an explicit operation invocation that destroys the Policy Applicator Object, which in turn destroys the Protocol Object and the Connection Object for the related DataServer data stream. For directly connected users of the Filter Service, the deactivation is simply a Deactivate operation invocation with no interaction between the Filter Service and the Filter Agent. The Deactivate Policy invocation creates an instance of the Deactivate Policy object [1], which encapsulates all internal object creation and messaging required supporting both remote and local Deactivations. The Deactivate Policy object deletes the persistent Policy Applicator object from the list of managed local clients. For Remote Deactivations, the Deactivate Policy object invokes a Deactivate Policy operation on the associated Filter Agent object. The Deactivate Policy Object invokes the CreateDeletionList and RemoteDelete operations. The Deactivate Filter object within the Filter Service performs RemoteDelete operation to the Filter Agent database for each entry contained within the DeletionList. The following sequence diagrams depict the remote and local deactivation of a Filter Client respectively.

Figure 9 Remote Deactivation

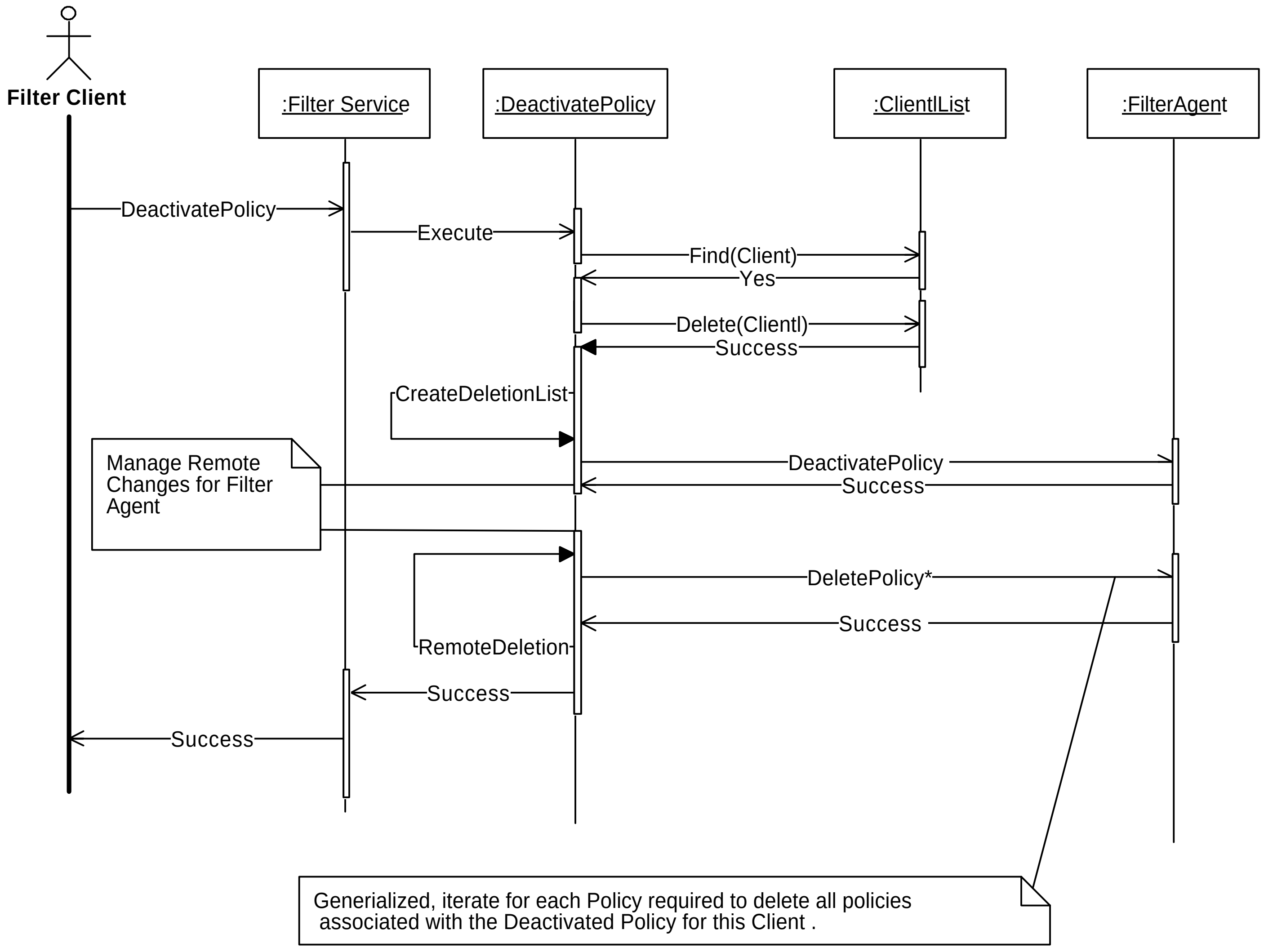
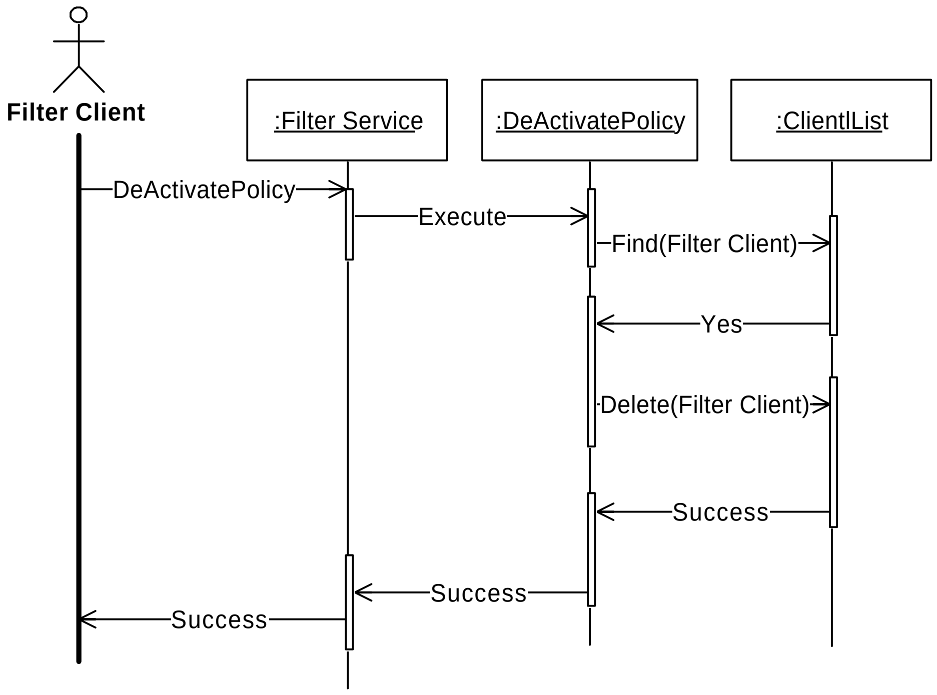


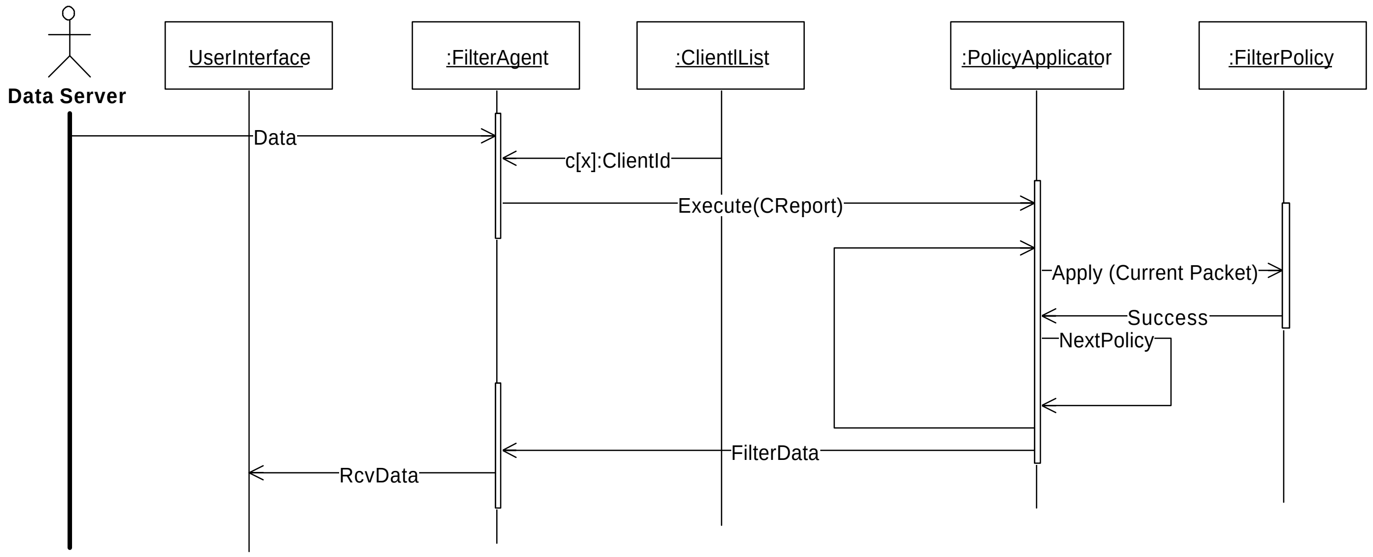
Figure 10 Local Deactivation



Data Filter

The Filter Data operation also requires a thread-safe environment that has per allocated Policy Applicators created specifically dedicated to servicing a particular Client ID. The Policy Applicator objects are active objects that provide a receive thread for the data being filtered and invoke the Filter Data operation on behalf of a particular client. If the data passes the Filter criteria then the Policy Applicator sends a message to the Protocol Object data that passed the filter, otherwise the packet data is disregarded. The network protocol and connection objects are dedicated objects that service a particular client they are contained within the Policy Applicator Object. The idea is to have the architecture to supports a Thread Pool of active Policy Applicator Objects, one per client. Each Policy Applicator dispatches filtered data through internal dedicated Connection and Protocol object pair serve as the pipe. The Filter Applicator maintains the connection and protocol object for as long as the Policy Applicator persists. This maybe implemented using the connector with in the Policy Applicator and an Acceptor in the Data Server. The Filter Applicator Connection objects presumably would be paired with a TCP/IP Protocol Object connected to the Client Window. The connection and protocol objects maintain the Policy Applicators User Interface callback object. These protocol and connection objects allow for an open standard and could be replaced or made select able to emulate other protocols and connection characteristics.

Figure 11 Filter Application Data



Consequences

Decoupling filtering from other services provides the freedom to implement these functions in different ways and to apply new policies or change existing policies.

In the case of CORBA , a Distributed Filter Service is not part of its standard architecture. This provides freedom to implement this pattern in different ways.

In some problem domains, e.g., computer vision, image recognition, etc., knowledge may be uncertain or approximate, which makes finding filtering policies difficult.

Implementation

We discuss now some issues in the definition of each one of the components of this pattern and some general aspects, i.e., concurrent implementation.

Filter Interface Definition

The Filter Interface is defined in terms of an inheritance hierarchy that encapsulates the CORBA interface components. The Filter Interface is defined by using the Interface Definition Language (IDL). The general Filter interface operations and attributes shared by both the Filter Service and Filter Agent objects is defined by the Base Class Filter interface definition. The Base Class Filter interface contains two Groups of operations. The first group of operations is Database management operations for example Copy, Append, Delete, Replace and Retrieve Policies. The second group of operations is operational directives and status queries for example Activate, Deactivate, Summary Information, Filter Status, RegisterClientID, GetSubsrcibtionList and the Filter Data operation.

Filter Agent Interface Definition

The Filter Agent defines the registered operation that is not included in the General Interface, because the behavior is unique to the Filter Agent. For Filter Agent objects has a GetSubsciberList operation which is invoked by the Filter Service to retrieve the registered Client list currently being serviced by Filter Agent. In general, all of the Policy Database management operations are inherited from the Filter Interface. The Filter Agent Database is a reduced subset of the Filter Service database that is abbreviated to include only the Filter Policy objects needed to filter currently activate client's data streams. The Filter Agent is required to AgentAttach to the Filter Service at startup Observer Pattern[1]. The Subscription process requires the Filter Agent to invoke the Filter Service GetAgentID operation to get a unique Agent identifier.

Filter Service Interface Definition

The Filter Service inherits from the Filter Interface and encapsulates the behavior specific to the Filter Service by defining additional operations. The AgentAttach and AgentDetach operation allow Filter Agent objects to AgentAttach and AgentDetach from the Filter Service Observer Pattern[1]. The AgentAttach operation causes the Filter Service to create and maintain and Notification Handle to the Remote Filter Agent Object. The Notification Handle is active as long as the AgentDetach operation is not invoked with the agent to be deactivated. The GetUniqueClientID and AddSubscriber operations are unique to the Filter Service objects and are needed to support the Filter Service Application and Client Windows. The GetLocalSubscriptionList and GetAgentSubscriptionList operations are provide for status reporting.

Concurrent Architecture -- Client Thread Servicing

A concurrent invocation of methods by multiple clients on the Filter Service and Filter Agent Objects requires the usage of thread-safe Design pattern. The Thread-Safe Command and Iterator Patterns [2] for Policy Database Management operations were utilized. The Design uses the Command Pattern [1] to create a completely independent thread-safe item that is a self contained implementation of the operation in a thread safe manner to process each request. The CDBCommand class encapsulates each incoming operation request as defined by the Command Pattern.

All database commands are encapsulated in a command object that are created specifically to handle the operation invocation and destroyed at the conclusion of the operation. The Command objects encapsulate all data needed to process the operation and servers to isolate the each Filter Client operation. Separate instances of Iterators are created within the Command Object to handle the Traversal of shared lists required to handler requests. The Command Objects are destroyed upon completion of the request on the Filter components Database [2]. For deletion of database objects a simple locking mechanism on the databases require the client thread to acquire a synchronization variable when sharing resources for readers and writers. The writer thread locks out new readers and waits until the all reader's threads have completed before allowing the data deletion to occur, insertion is thread safe. The concurrent operation invocations are isolated with the thread-safe objects so concurrent threads do not corrupt internal variables. One underlying assumption has been made is that the ORB library is thread safe for the concurrent calling threads invoking the operation. The following class diagram depicts the database and iterator objects, which implement the Iterator pattern in this system.

Filter Agent Factory and Filter Service Factory

The creation pattern used for factories is the Abstract Factory [1]. The Abstract Factory extends the CORBA Naming Service and allows for the Remote Filter Service application to manager instances of the Filter Agent objects. The Factory is an active object with a startup and shutdown operation that creates the Factories internal Thread Manager. The Startup operation is the operation that starts the factory event loops and waits for thread to begin execution.

The Factory constructor accepts a Factory Name and a Maximum number of resources as augments; the factory will create only the number of resource specified by the creator for this specific instance of the factory. The maximum number of resources value is used to cap the size of allocated Object arrays and the Managed Resource array for the Factory. The Factory maintains an array of Managed Resource. The Factory construction routine includes binding the Factory name to the naming service.

The RequestResource operation allocates a resource (gets a reference) informs the ORB of its presence, and returns a CORBA::_duplicate reference to a resource. The RequestResource and specifies and directs the creation or additional reference to an existing requested managed Resource.

The ReturnResource operations return a resource to the Factory and decrement the reference count to the Object. In addition, the operation evaluates the reference count for being equal to zero, if it is the last reference then the object is deactivated and the entry removed it from the Managed Resource array. The deactivation is performed within a operation Remove Managed Resource.

The operation Thread Service svc() is the thread that is started by the invocation of the Startup operation. Each factory has an operation called svc() which is invoked by a call to the Startup operation. The Startup operation causes the spawning of a worker thread that that particular instance of the factory.

It is presumed that the DataServer is another CORBA object that is a Registered Resource with the naming service.

Sample Code

Creating the Filter Policy using the Filter Agent

Client Window: local

```
FilterAgent::DataPolicyList  Policies(3) ;

Policies.Name = “Censor”;
Policies.Receive = False;
Policies.length(3);

FilterAgent::SymbolPolicy_var  Symbol ;
Symbol = myFilterAgent->create_symbol_policy (FilterAgent::SYMBOL_ID, CORBA::String
sym);

FilterAgent::URLPolicy_var  URLPolicy ;
URLPolicy = myFilterAgent->create_URL_policy (FilterAgent::URL_ID, CORBA::String sym);

// Allow s for user defined strategy
FilterAgent::UserDefined_var  Newpolicy ;
Newpolicy  = myFilterAgent->create_user_policy (FilterAgent::USER, DataPolicy Algorithm)

Policies[0] = Symbol;
Policies[1] = URLPolicy;
Policies[2] = NewPolicy;

myFilterAgent->ActivatePolicy (ClientID,Policies)
```

Known Uses

The Distributed Data Filter patterns abstractions may be used in the Filter Applications that are present in an Object Web based Filter Service. It is likely that current distributed systems are already using this type of architecture.

The concept of Distributed Filter Service Components was used in a large Distributed Object-Oriented System programmed at Raytheon, St. Petersburg, FL, using the Adaptive Communication Environment (ACE) from Washington University.

Related Patterns

The Distributed Data Filter Pattern combines many design patterns in a reusable abstract Framework, including:

Abstract Factory	-- Creation of Filter Policies and distributed objects.
Command Pattern	-- Command objects for improved extensibility and thread safe operations.
Iterator Pattern	-- Thread safe database traversals.
Strategy Pattern	-- Re-programmable Filter Policy objects and Policy Applicator objects.
Composite Pattern	-- Filter Policy List may contain other Filter Policy lists.
Observer Pattern	-- Filter Service contains the change manager: uses subscribe and publish.
Proxy Pattern	-- Component Architecture is based on remote proxies.

Two complementary patterns are:

- The Authenticator [7], that has as its objective providing client request authentication before access.
- The Bodyguard [8], that determines access rights based on authorizations.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1994.

[2] Sean Baker, *CORBA Distributed Objects Using Orbix*. Reading, MA: Addison-Wesley, 1994.

[3] Robert Orfali, Dan Harkey, and Jeri Edwards, Corba, Java, and the Object Web. *BYTE Magazine*, Oct 1997.

[4] Steve Vinoski and Doug Schmidt: Object Interconnections: Using the Portable Object Adapter for Transient and Persistent CORBA Objects*. C++ Report*, April 1998

[5] Bruce P. Douglas: *Real-Time UML Developing Efficient Objects for Embedded Systems*. Reading, MA: Addison-Wesley, 1998.

[6] Steve Vinoski and Doug Schmidt: Object Interconnections: Comparing Alternative Server Programming Techniques*. C++ Report*, Oct 1995.

[7] F. L. Brown, J. DiVietri, G. Diaz de Villegas, and E.B.Fernandez, "The Authenticator pattern", submitted to PLOP'99.

[8] F. Das Neves and A. Garrido, "Bodyguard", Chapter 13 in *Pattern Languages of Program Design 3*, Addison-Wesley 1998.