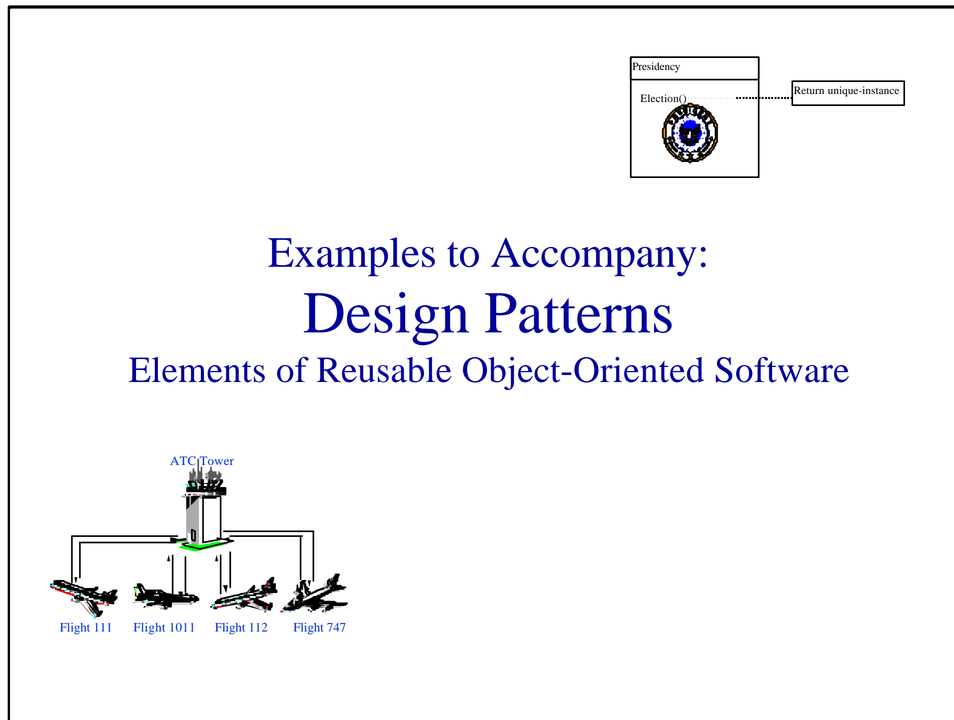


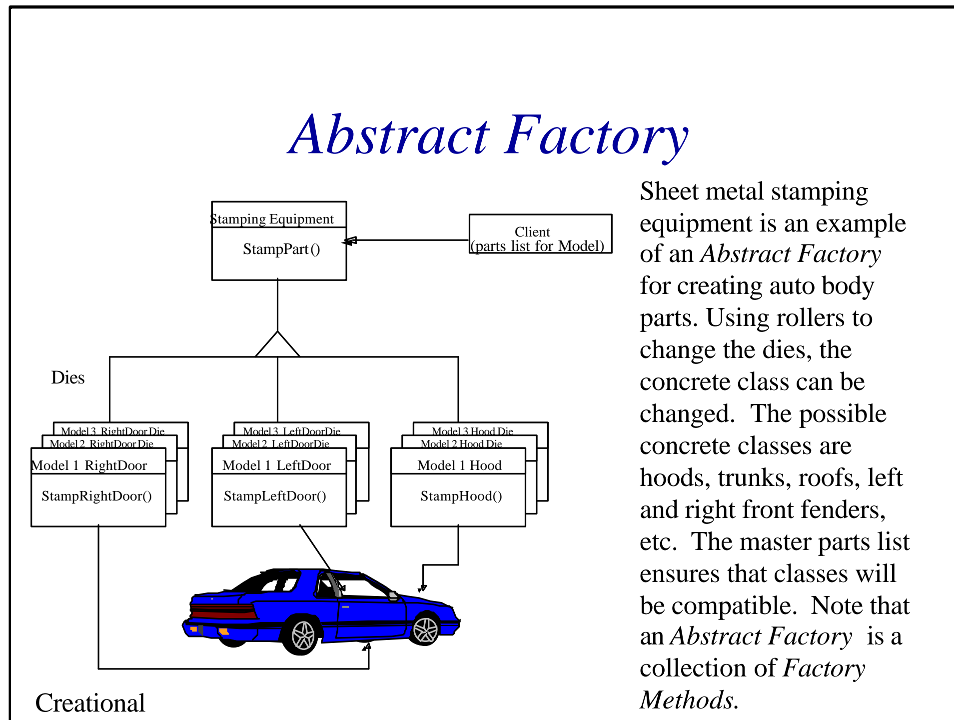
Design Patterns - Elements of Reusable Object-Oriented Software was written by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (also known as the Gang of Four, or GoF)

It was published by Addison-Wesley in 1995, and is regarded as the first text on Software Design Patterns.

Non-Software examples of these patterns were published by Michael Duell in *Object Magazine* in July, 1997. The examples here are the result of an OOPSLA '97 workshop of Non-Software Examples of Software Design patterns, conducted by Michael Duell, John Goodsen and Linda Rising.

In addition to the workshop organizers, contributors to this body of work include Brian Campbell, Jens Coldeway, Helen Klein, James Noble, Michael Richmond, and Bobby Woolf.

© AG Communication Systems - 1999



The purpose of the *Abstract Factory* is to provide an interface for creating families of related objects without specifying concrete classes.

Participant Correspondence:

The Master Parts List corresponds to the client, which groups the parts into a family of parts.

The Stamping Equipment corresponds to the Abstract Factory, as it is an interface for operations that create abstract product objects.

The dies correspond to the Concrete Factory, as they create a concrete product.

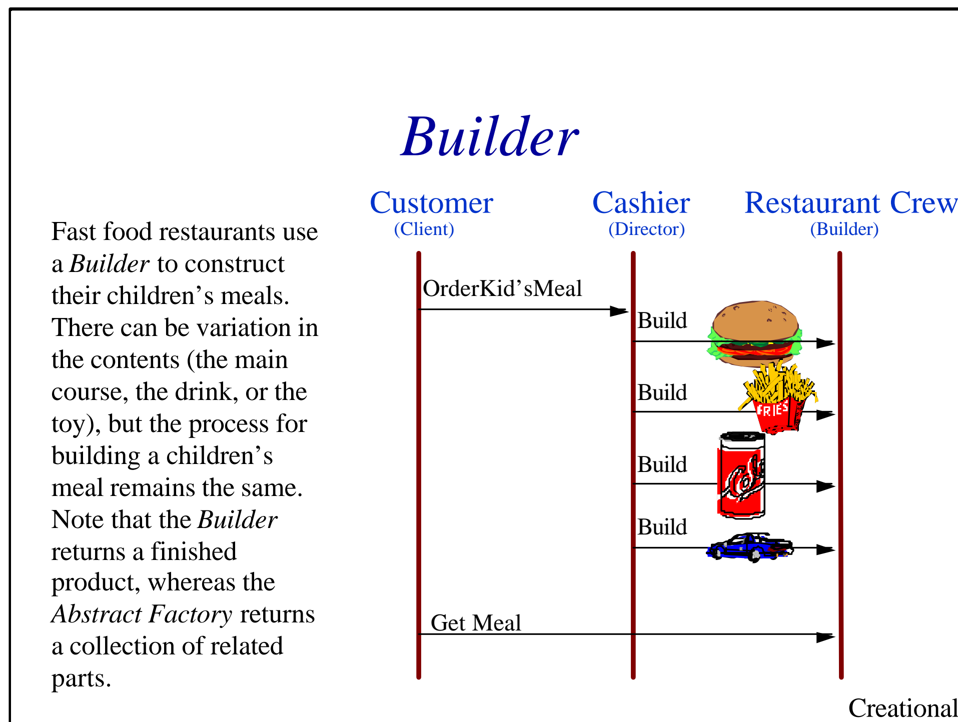
Each part category (Hood, Door, etc.) corresponds to the abstract product.

Specific parts (i.e., driver side door for 1998 Nihonsei Sedan) corresponds to the concrete products.

Consequences:

Concrete classes are isolated in the dies.

Changing dies to make new product families (Hoods to Doors) is easy.



The *Builder* pattern separates the construction of a complex object from its representation, so the same construction process can create different representations.

Participant Correspondence:

The Kid's Meal concept corresponds to the builder, which is an abstract interface for creating parts of the Product object.

The restaurant crew corresponds to the ConcreteBuilder, as they will assemble the parts of the meal (i.e. make a hamburger).

The cashier corresponds to the Director, as he or she will specify the parts needed for the Kid's Meal, resulting in a complete Kid's meal.

The Kid's Meal package corresponds to the Product, as it is a complex object created via the Builder interface.

Consequences:

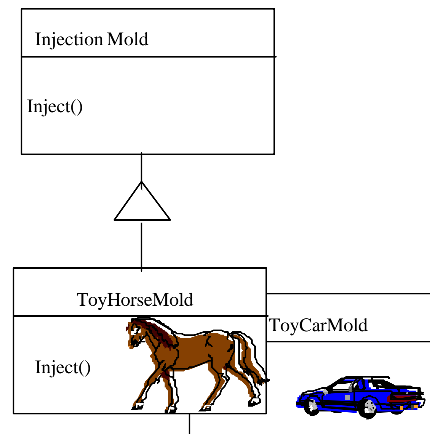
The internal representation of the Kid's meal can vary.

The construction process is isolated from the representation. The same process is used by virtually all of the fast food chains.

There is finer control over the construction process and the internal structure of the finished product. Hence two Kid's Meals from the same restaurant can consist of entirely different items.

Factory Method

In injection molding, manufacturers process plastic molding powder and inject the plastic into molds of desired shapes. Like the *Factory Method*, the subclasses (in this case the molds) determine which classes to instantiate. In the example, the ToyHorseMold class is being instantiated.



Creational

The *Factory Method* defines an interface for creating objects, but lets subclasses decide which classes to instantiate.

Participant Correspondence:

The Injection Mold corresponds to the Product, as it defines the interface of the objects created by the factory.

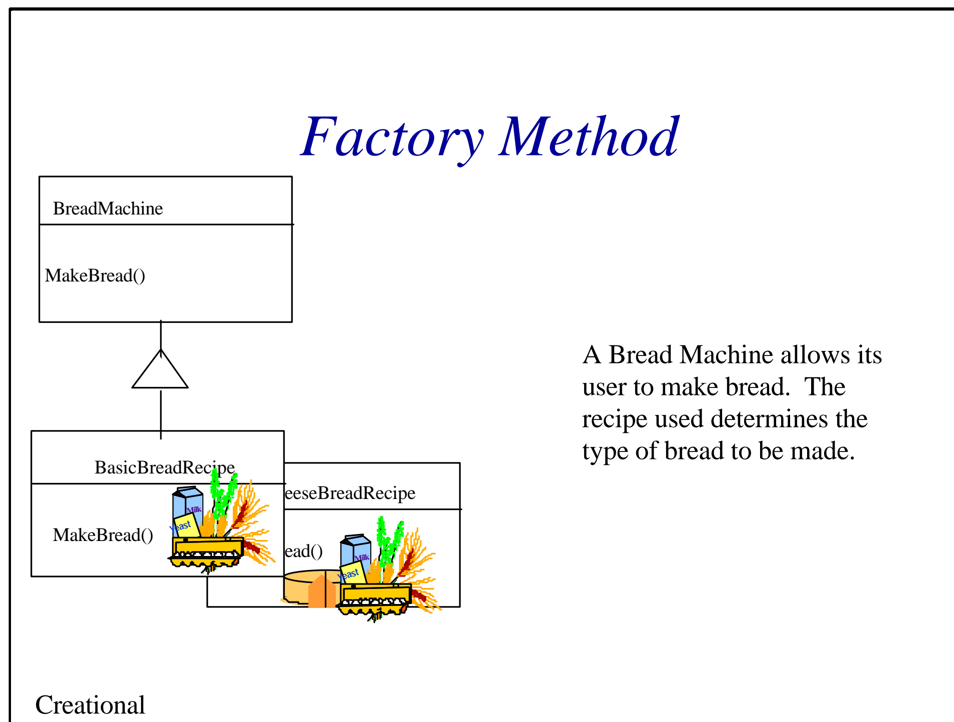
A specific mold (ToyHorseMold or ToyCarMold) corresponds to the ConcreteProduct, as these implement the Product interface.

The toy company corresponds to the Creator, since it may use the factory to create product objects.

The division of the toy company that manufactures a specific type of toy (horse or car) corresponds to the ConcreteCreator.

Consequences:

Creating objects with an Injection Mold is much more flexible than using equipment that only created toy horses. If toy unicorns become more popular than toy horses, the Injection Mold can be extended to make unicorns.



The *Factory Method* defines an interface for creating objects, but lets subclasses decide which classes to instantiate.

Participant Correspondence:

The Bread Machine corresponds to the Product, as it defines the interface of the objects created by the factory.

A specific recipe (BasicBreadRecipe or CheeseBreadRecipe) corresponds to the ConcreteProduct, as these implement the Product interface.

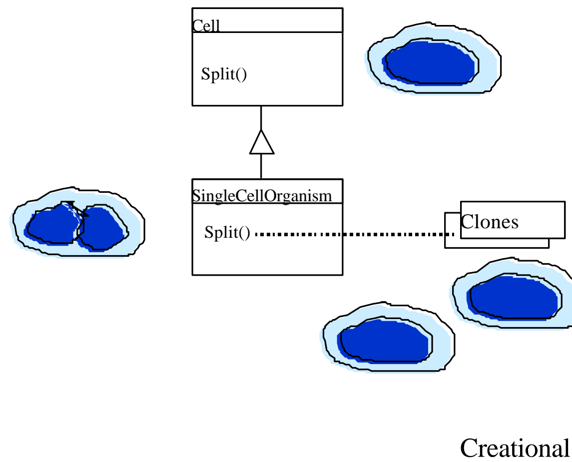
The user corresponds to the Creator, since he or she uses the factory to create product objects.

Consequences:

Creating objects with a bread machine is much more flexible than using baking equipment that only created one type of bread.

Prototype

The mitotic division of a cell results in two cells of identical genotype. This cell “cloning” is an example of the *Prototype* pattern in that the original cell takes an active role in creating a new instance of itself.



The *Prototype* pattern specifies the kind of objects to instantiate using a prototypical instance.

Participant Correspondence:

The cell corresponds to the Prototype, as it has an “interface” for cloning itself.

A specific instance of a cell corresponds to the ConcretePrototype.

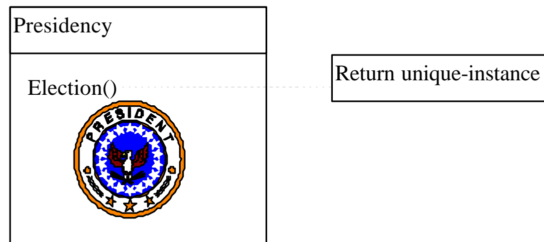
The DNA or genetic blue print corresponds to the Client, as it creates a new cell by instructing a cell to divide and clone itself.

Consequences:

Many applications build objects from parts and subparts. For convenience, complex systems can be instantiated using subparts again and again. In complex organisms, cells divide, and form various organs which in turn, make up the organism.

Singleton

The office of the Presidency of the United States is an example of a *Singleton*, since there can be at most one active president at any given time. Regardless of who holds the office, the title "The President of the United States" is a global point of reference to the individual.



Creational

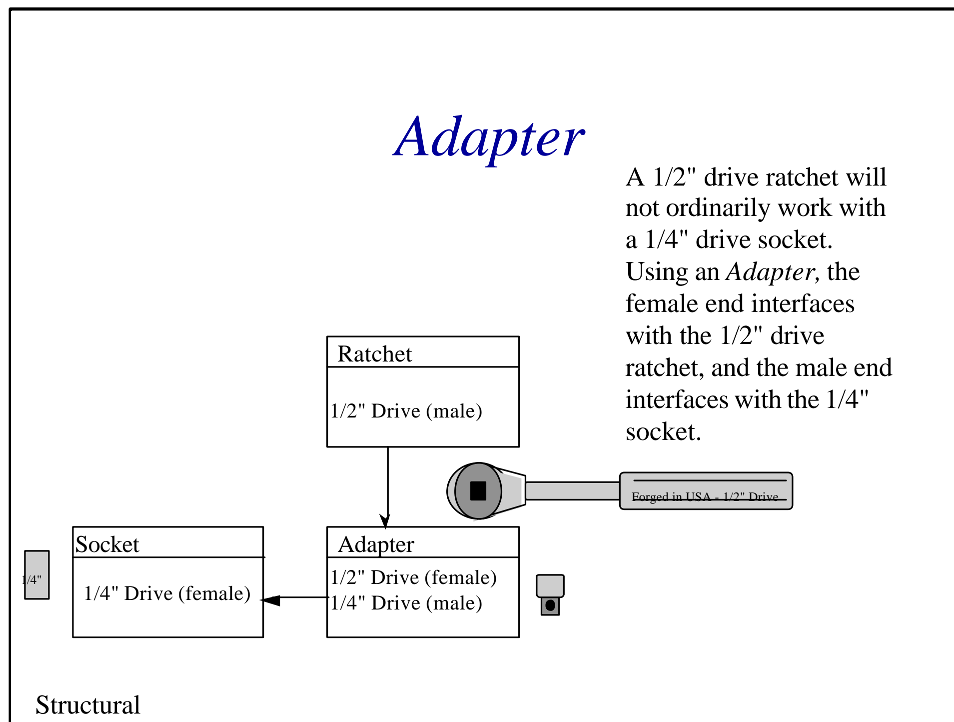
The *Singleton* pattern ensures that a class has only one instance, and provides a global point of reference to that instance.

Participant Correspondence:

The Office of the Presidency of the United States corresponds to the Singleton. The office has an instance operator (the title of "President of the United States") which provides access to the person in the office. At any time, at most one unique instance of the president exists.

Consequences:

The title of the office provides controlled access to a sole instance of the president. Since the office of the presidency encapsulates the president, there is strict control over how and when the president can be accessed.



The *Adapter* pattern allows otherwise incompatible classes to work together by converting the interface of one class to an interface expected by the clients.

Participant Correspondence:

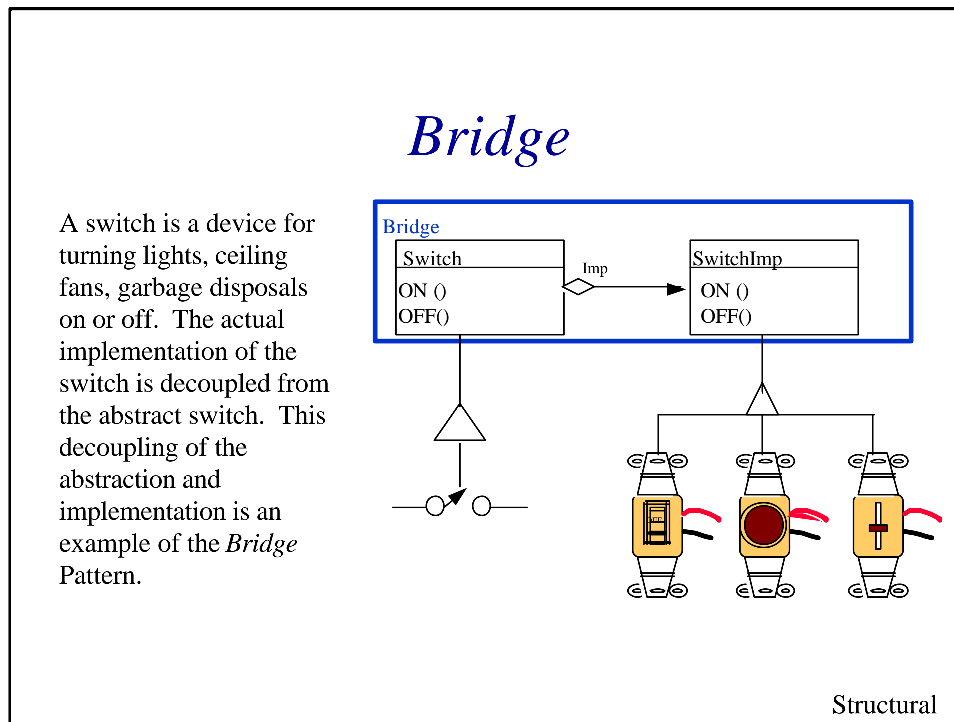
The ratchet corresponds to the Target, as it is the domain specific interface that the client uses.

The socket corresponds to the Adaptee, since it contains an interface (1/4" drive) that needs adapting.

The socket adapter corresponds to the Adapter, as it adapts the inteface of the Adaptee to that of the Target.

Consequences:

The Adaptee is adapted to the target by committing to a concrete adapter object.



The *Bridge* pattern decouples an abstraction from its implementation, so that the two can vary independently. Note that the schematics of house wiring state only where switches will be located, not what type of switch it will be.

Participant Correspondence:

In the example, the Switch corresponds to the Abstraction.

The SwitchImp corresponds to the Implementor.

The specific type of switch would correspond to the ConcreteImplementor.

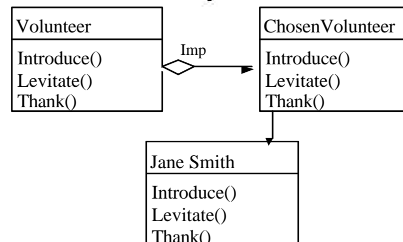
Consequences:

The interface and implementation are decoupled. With the *Bridge* the implementation of an abstraction is often done at run time. In the switch example, the selection of a physical switch can be delayed until the switch is actually wired. The switch can be changed without requiring a redesign of the house.

Implementation details are hidden. Builders need only know that a switch is needed. The house can be framed, wired, and dry walled without anyone knowing the concrete implementation of the switch.

Bridge

A magician relies on the bridge pattern for his act. The act is developed with a volunteer, but the identity of the volunteer is not known until the time of the performance



Structural

The *Bridge* pattern decouples an abstraction from its implementation, so that the two can vary independently. Note that a magician's act requires an abstract volunteer. The specific identity of the volunteer is not known until the time of the act. The specific identity can (and often does) vary from performance to performance.

Participant Correspondence:

In the example, the Volunteer corresponds to the Abstraction.

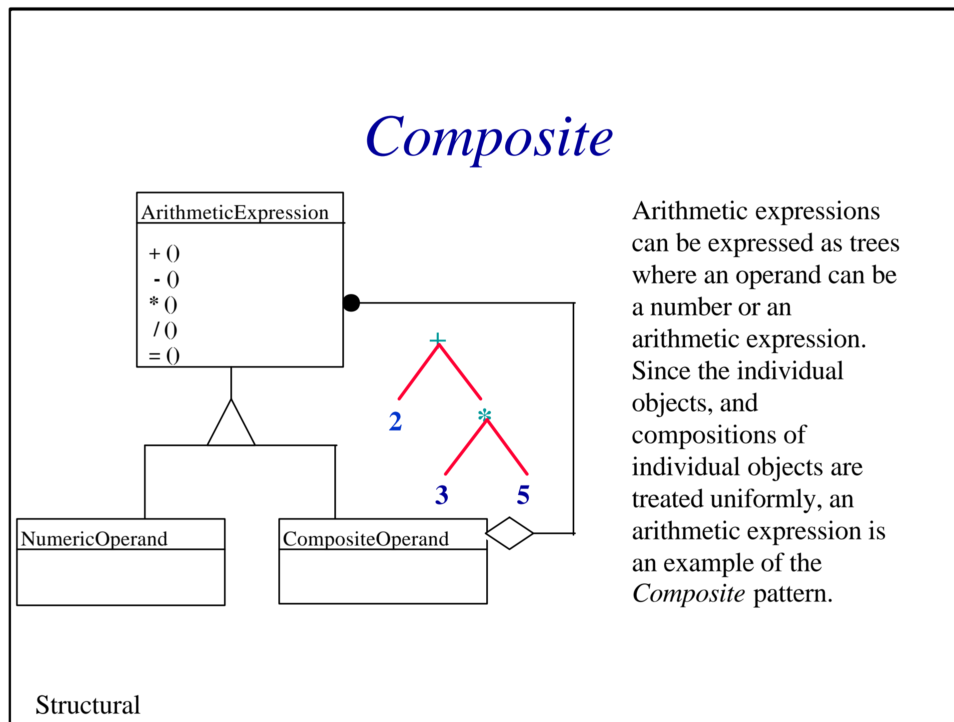
The ChosenVolunteer corresponds to the Implementor.

The specific volunteer (Jane Smith) corresponds to the ConcreteImplementor.

Consequences:

The interface and implementation are decoupled. With the *Bridge* the implementation of an abstraction is often done at run time. In the magician example, the selection of a specific volunteer can be delayed until the performance. The volunteer can be changed without requiring a redesign of the act.

Implementation details are hidden. The magician does not need to know who is in the audience before hand.



The *Composite* composes objects into tree structures, and lets clients treat individual objects and compositions uniformly.

Participant Correspondence:

Any arithmetic expression is a component.

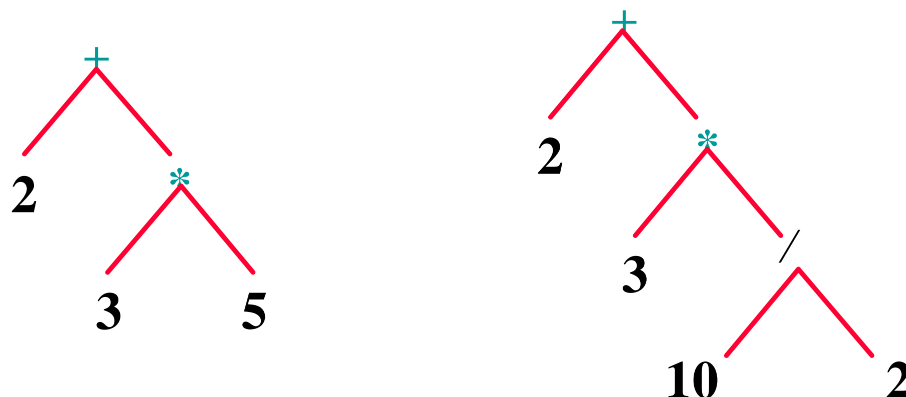
Numeric operands correspond to leafs, while expressions containing at least one operator correspond to composites.

Whoever forms the expression is the client.

Consequences:

The composite pattern defines class hierarchies consisting of leaf objects and composite objects. Composite objects are treated the same way as leaf objects (their value is added, subtracted, etc. from another value).

With composites, it is easy to add new kinds of components. For example, the following expression can be rewritten:



Composite

Composite is often exhibited in recipes. A recipe consists of a list of ingredients which may be atomic elements (such as milk and parsley) or composite elements (such as a roux).

White Roux
1 tablespoon flour
1 tablespoon butter

To make roux, melt the butter in a saucepan over medium heat. When the butter starts to froth, stir in flour, combining well.

Continue cooking the roux over heat until it turns a pale brown and has a nutty fragrance.

Pasley Sauce
1 portion of white roux
1 cup milk
2 tablespoons of parsley

To make sauce, over heat, add a little of the milk to the roux, stirring until combined. Continue adding milk slowly until you have a smooth liquid. Stir over medium heat for 2 minutes. Then stir in parsley, and cook for another 30 seconds. Remove from heat, and leave sauce standing to thicken.

Structural

The *Composite* composes objects into tree structures, and lets clients treat individual objects and compositions uniformly.

Participant Correspondence:

Any item in a recipe is a component.

The simple elements such as milk, correspond to the leaf objects.

Elements such as the white roux, which are themselves composed of leaf elements are composites.

The chef corresponds to the client.

Consequences:

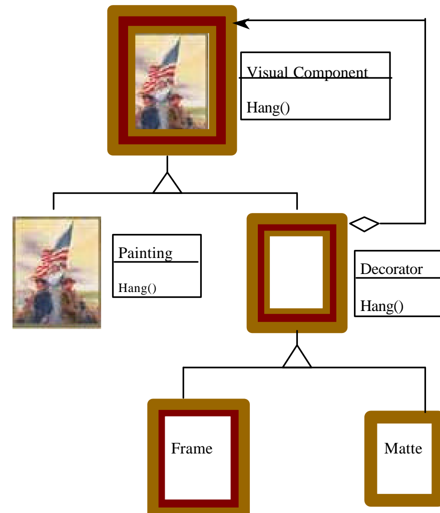
Recipes can be written more simply by combining primitive and composite objects.

When combining elements of a recipe, composite elements are added the same way that simple elements are.

It is easy to add new kinds of elements, as is evidenced by the frequency in which recipes are combined.

Decorator

Paintings can be hung on a wall with or without frames. Often, paintings will be matted and framed before hanging. The painting, frame, and matting form a visual component



Structural

The *Decorator* attaches additional responsibilities to an object dynamically.

Participant Correspondence:

The abstract "painting" corresponds to the Component.

A concrete painting corresponds to the ConcreteComponent.

The frame and matte correspond to the Decorator.

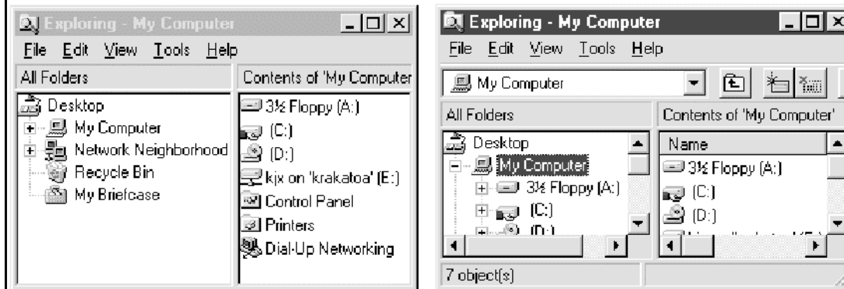
Consequences:

Adding or removing frames and mattes provide more flexibility than requiring all paintings to have the same frame.

Paintings can be customized with the addition of mattes and frames. The cost of customization is determined by the framing and matting options chosen.

The decorator and component remain separate.

Decorator



The graphics displays used on GUI desktops use decorators such as toolbars, status bars, and scroll bars.

Structural

The *Decorator* attaches additional responsibilities to an object dynamically.

Participant Correspondence:

The windowcorresponds to the Component.

A specific window corresponds to the ConcreteComponent.

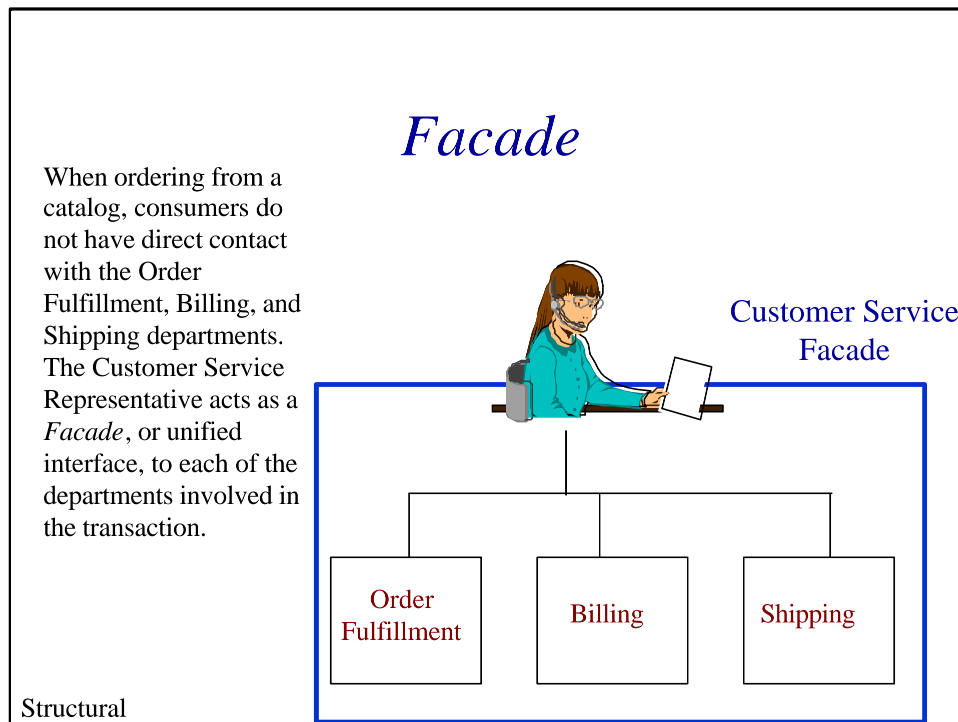
The borders, status bars, and scroll bars correspond to the Decorator.

Consequences:

Adding items such as scroll bars as needed provides more flexibility than requiring all windows to have scroll bars.

If scrolling is not needed, the cost of a scroll bar is not incurred.

The decorator and component remain separate.



The *Facade* defines a unified, higher level interface to a subsystem, that makes it easier to use.

Participant Correspondence:

The customer service representative corresponds to the Façade.

The individual departments correspond to the subsystem classes.

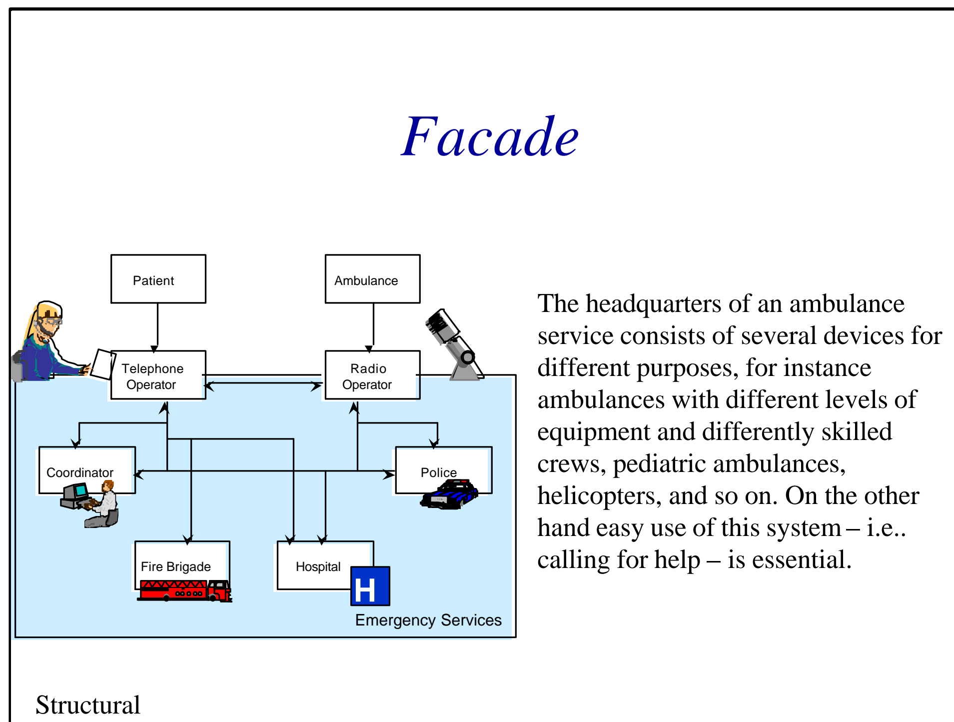
Consequences:

Clients are shielded from individual departments. When ordering an item, it is not necessary to check the stock, generate an invoice, and arrange for shipping. All steps are accomplished through the customer service representative.

The internal structure of the organization can be changed without affecting the client. For example, the shipping department can be contracted to another organization, without impacting the client's interface with the company.

Note:

Some catalog department stores require the customer to select the item, check to see if it is in stock, order it, go to a counter to pay for it, and then go to another counter to receive it. In this example, a façade is not used.



The *Facade* defines a unified, higher level interface to a subsystem, that makes it easier to use.

Participant Correspondence:

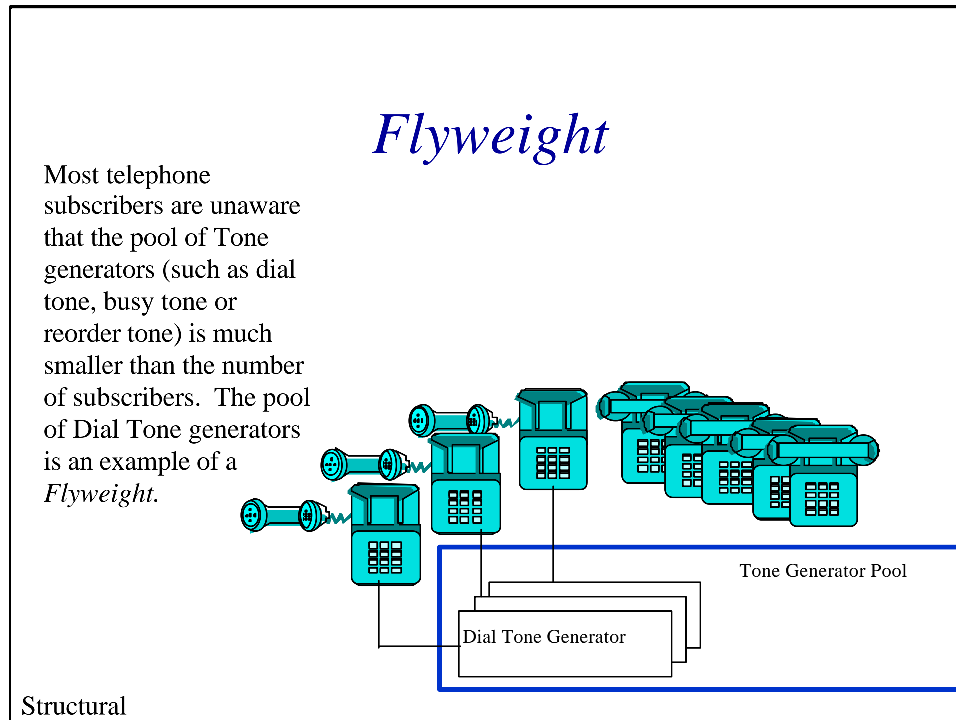
The emergency services operator (9-1-1 operator in North America) corresponds to the Façade.

The individual emergency services correspond to the subsystem classes.

Consequences:

Clients are shielded from individual departments. When emergency services are needed, it is not necessary to call the ambulance, police and fire departments separately. The emergency services operator dispatches services as needed.

There is a weak coupling between services. In the event of a burglary, the fire brigade need not be dispatched. Regardless of who is dispatched, the client interface remains the same.



The *Flyweight* uses sharing to support large numbers of objects efficiently.

Participant Correspondence:

Tone generators correspond to the Flyweight.

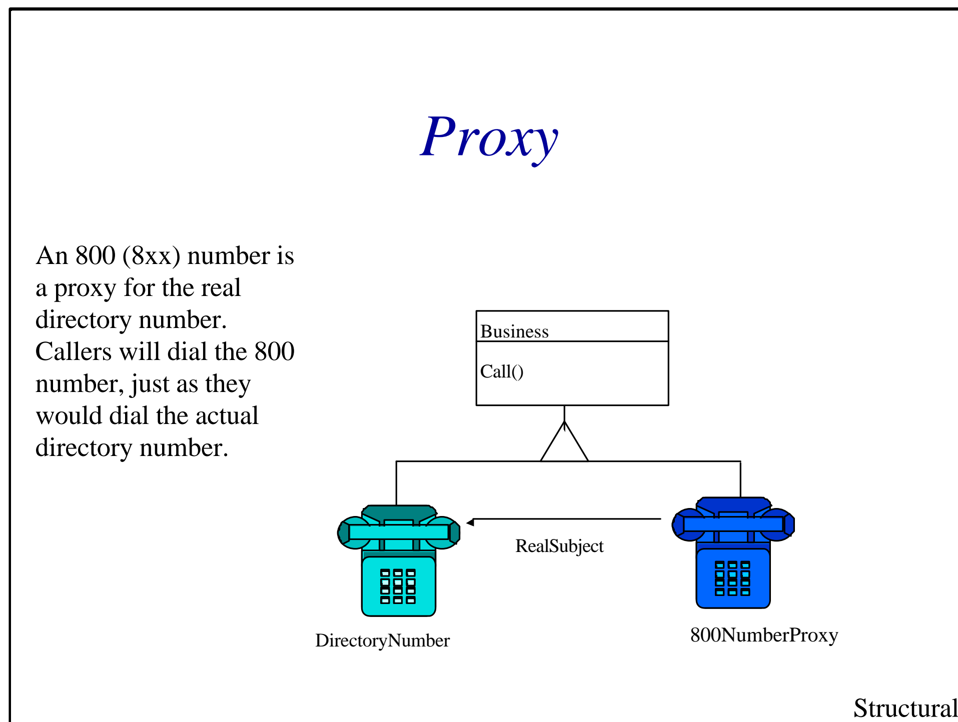
The physical Dial Tone generator corresponds to the ConcreteFlyweight.

The Tone Generator pool corresponds to the FlyweightFactory. When a tone generator is requested, it is connected to the subscriber line. When it is no longer need, it is disconnected so that it can be used by another.

The telephone switch corresponds to the client, since it maintains the reference to the flyweights.

Consequences:

Developing mechanisms for sharing items between multiple users is not without cost. The cost is offset by a reduction in the number of tone generators required, and the reduction in space required in the physical plant.



The *Proxy* provides a surrogate or place holder to provide access to an object.

Participant Correspondence:

The 800 number corresponds to the Proxy. The 800 number is not actually assigned to a line, but it is translated to a number that is.

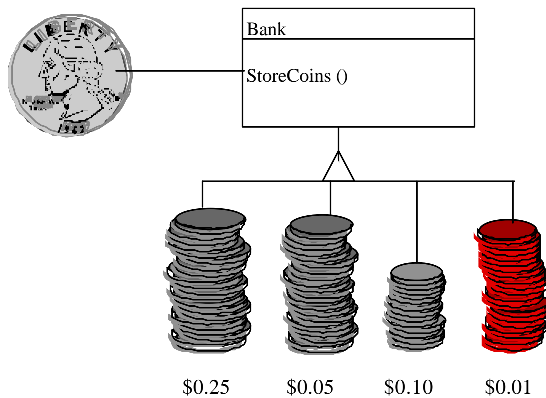
The directory number corresponds to the real subject. It is assigned to a line.

The business corresponds to the subject. It is the interface that allows the RealSubject and Proxy to work together.

Consequences:

The proxy introduces a level of indirection when accessing an object. The indirection can be used for security, or disguising the fact that an object is located in a different address space. In the case of the 800 number, it is a remote proxy, which hides the actual location of the business.

Chain of Responsibility



A mechanical sorting bank uses a single slot for all coins. As each coin is dropped, a *Chain of Responsibility* determines which tube accommodates each coin. If a tube cannot accommodate the coin, the coin is passed on until a tube can accept the coin.

Behavioral

The *Chain of Responsibility* pattern avoids coupling the sender of a request to the receiver.

Participant Correspondence:

The person dropping a coin in the bank corresponds to the client, since he is initiating the request.

The coin corresponds to the request.

Each tube corresponds to ConcreteHandlers in the chain.

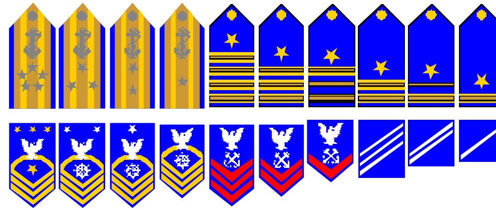
Consequences:

There is reduced coupling since the coin slot object does not need to know the proper tube *apriori*. Although there are 4 tubes, only one slot is used.

Receipt is not guaranteed. Since there is no explicit receiver, the request may be passed by all members in the chain. An attempt to put a Canadian $2 coin in the bank would result in a coin that is not put in any slot.

Chain of Responsibility

The *Chain of Responsibility* is demonstrated in the military, where some underling asks for approval and the request is passed from superior to superior until someone finally makes a decision. If a Seaman is asked for permission to enter a Base, he will likely forward the request up the chain of command.



Behavioral

The *Chain of Responsibility* pattern avoids coupling the sender of a request to the receiver.

Participant Correspondence:

The person asking permission to enter a Naval Base corresponds to the client, since he is initiating the request.

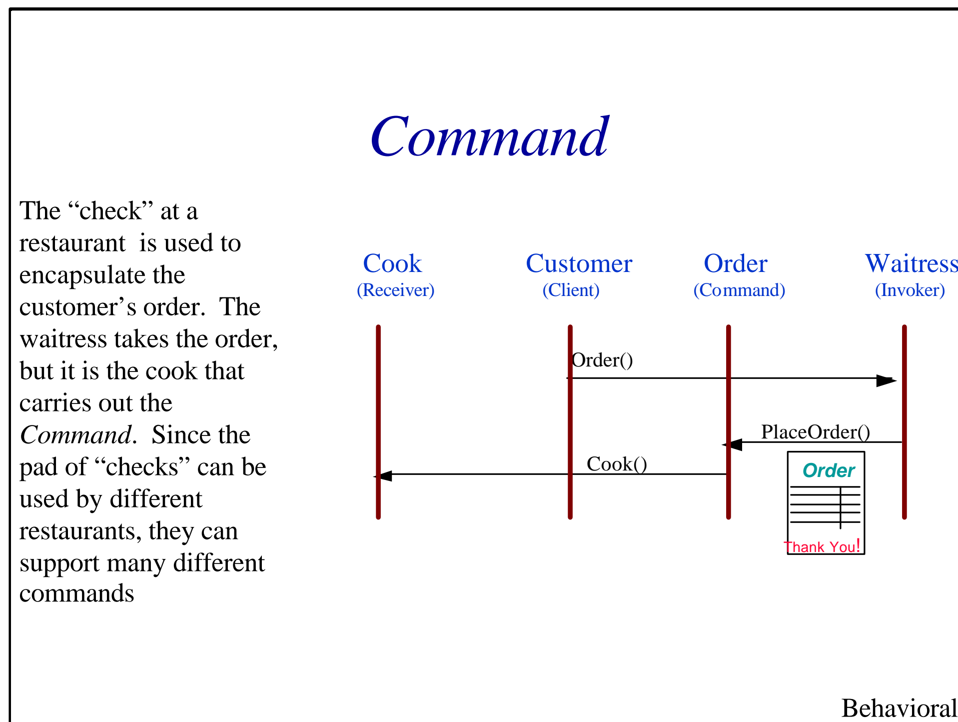
Each person in the chain of command corresponds to ConcreteHandlers in the chain.

Consequences:

There is reduced coupling since the requester does not have to know which person has the authority to grant the request. Such knowledge would require that contact be made with the person in authority.

There is added flexibility in assigning responsibilities. Based on the nature of the visit, the person with the appropriate level of authority may change. The Seaman has the authority to let a fellow sailor on the base, but not a newspaper photographer.

Receipt is not guaranteed. The client can only pose the request to one member of the chain. Clients have no control over who handles the request.



The *Command* pattern allows requests to be encapsulated as objects, thereby allowing clients to be parameterized with different requests.

Participant Correspondence:

The written order corresponds to the command. It creates a binding between the cook and the action.

The cook corresponds to the receiver. He receives the order and is responsible for executing it.

The customer corresponds to the client. She creates the command by placing an order.

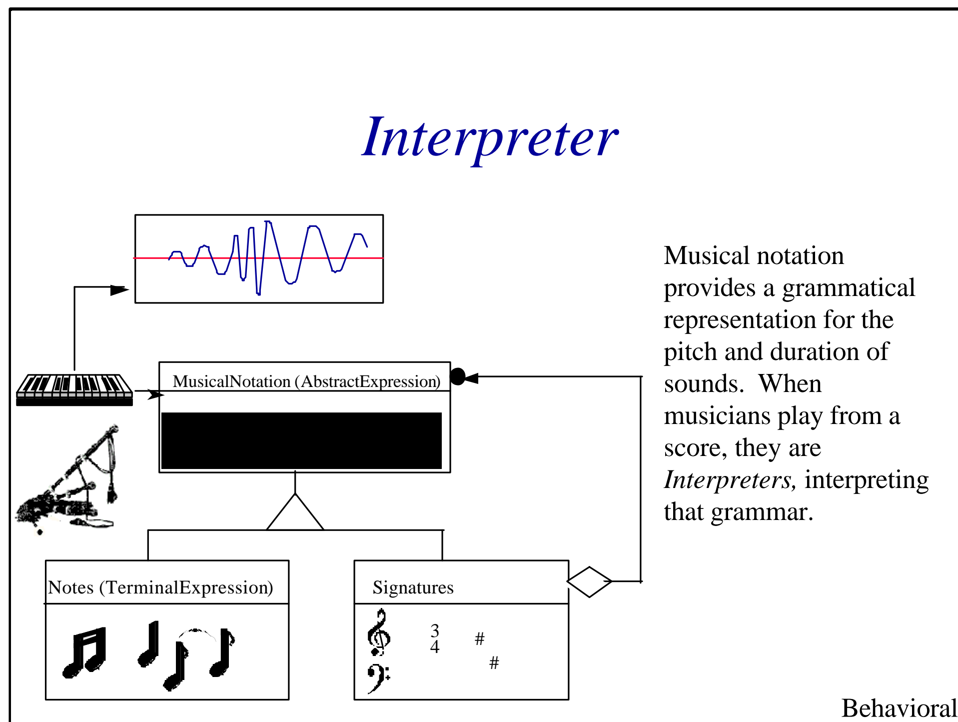
The waitress corresponds to the invoker. She activates the command by placing it in a queue.

Consequences:

The object that invokes the command and the one that performs it are decoupled. In the example, the waitress does not need to know how to cook.

Like the order, commands are first class objects that can be manipulated and extended. At any time an order can be modified. For example, the client may wish to add a piece of pie to the order.

Commands can be assembled into composite commands. When several people at a table order on the same check, the command is a composite.



The *Interpreter* pattern defines a grammatical representation for a language, and an interpreter to interpret the grammar.

Participant Correspondence:

Musical notation corresponds to the abstract expression.

Notes correspond to terminal expressions. Each note indicates pitch and duration of the tone to be played.

Time and key signatures correspond to non terminal expressions. On their own they cannot be interpretted. They do add context, however.

Knowing how to produce the proper sounds corresponds to the context, or information global to interpreters.

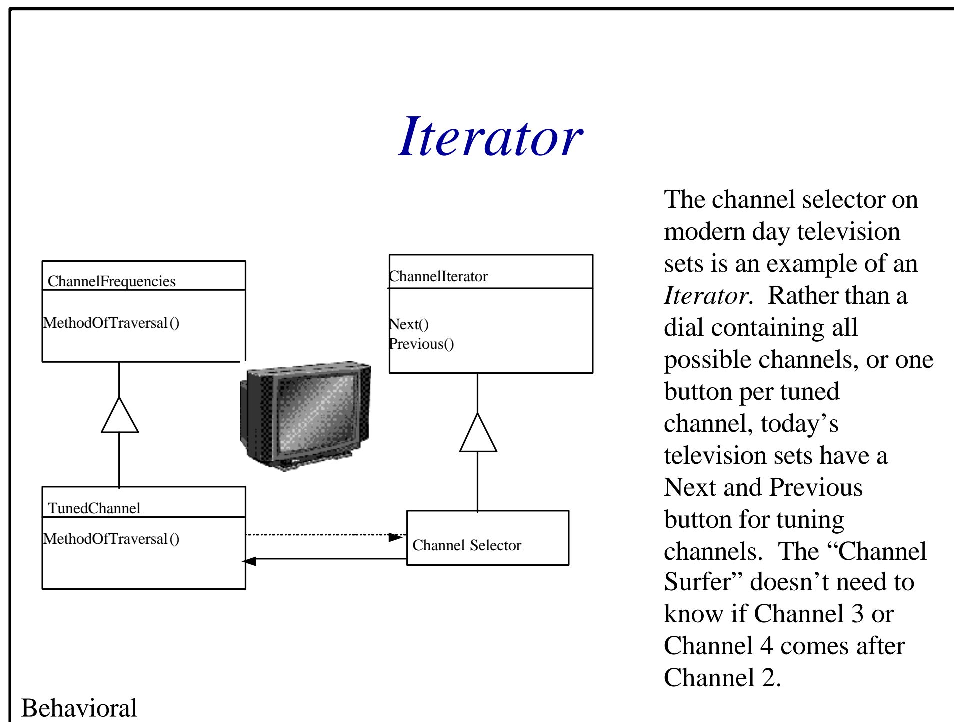
The musician interprets the music, and therefore corresponds to the interpreter.

Consequences:

It is easy to change and extend the grammar. Existing expressions can be modified to define new expressions. The sequence of three grace notes preceding the “D” in the music above is readily recognizable as a “Hard D Doubling” by Highland pipers.

Similarities in nodes make implementation easy. These similarities allow music to be transposed from one key to another.

It is easy to add new ways to interpret expressions. In music, the dynamics change the interpretation of the piece.



The *Iterator* provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object.

Participant Correspondence:

The channel selector corresponds to the iterator.

The UP/DOWN buttons on the remote control correspond to the concrete iterator.

The VHF channels 2-13, and UHF channels 14-83 correspond to the aggregate.

In any geographical area, all 82 broadcast channels are not in use. The channels in use correspond to the concrete aggregate.

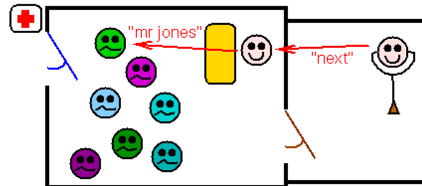
Consequences:

Iterators support variation in the traversal of an aggregate. The channel selection can traverse in ascending or descending order, from any point in the aggregate.

Iterators simplify the aggregate interface. With modern channel selectors, a channel surfer can traverse only the channels in use. With the old dials, every channel had to be traversed whether in use or not.

Iterator

The receptionist in a doctor's waiting room iterates the aggregate of patients who are seated randomly around the room. The receptionist will send the "next" patient to the doctor.



Behavioral

The *Iterator* provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object.

Participant Correspondence:

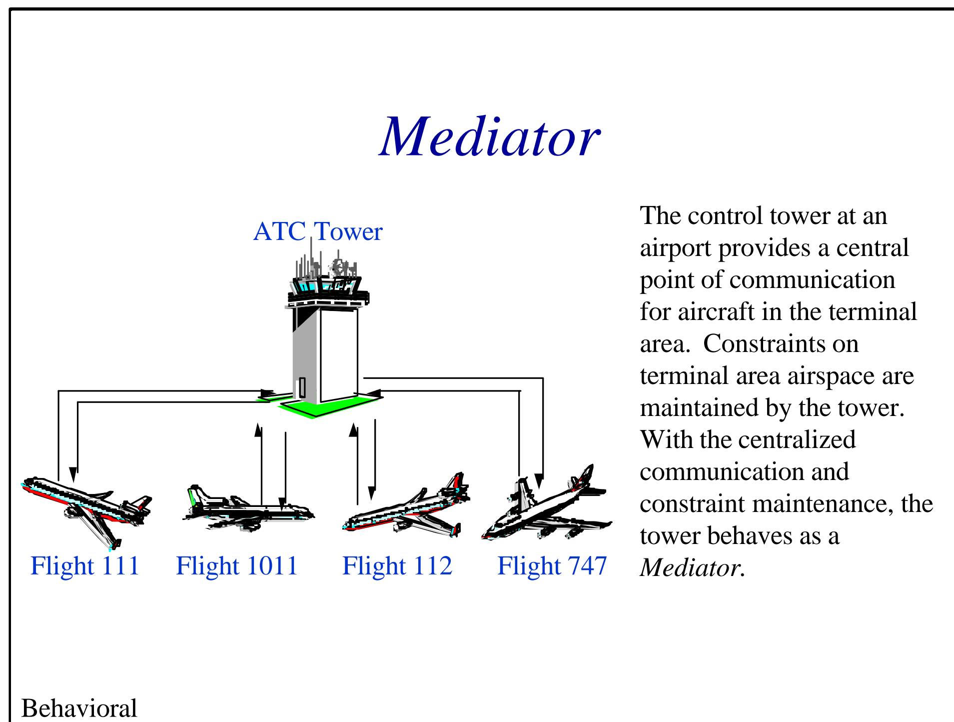
The receptionist calling names in the doctor's office corresponds to the concrete iterator.
The patients waiting in the doctor's office correspond to the concrete aggregate.

Consequences:

Iterators support variation in the traversal of an aggregate. The receptionist can select the next patient based on arrival time, appointment time or the severity of illness.

Iterators simplify the aggregate interface. Patients do not have to stand in a line in order to be seen.

More than one traversal can be pending on an aggregate. If there are multiple doctors in the office, the receptionist traverses the aggregate based on who the patient is seeing. In effect, multiple traversals are occurring.



The *Mediator* defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than one another.

Participant Correspondence:

Air Traffic Control corresponds to the Mediator.

The specific tower at an airport corresponds to the ConcreteMediator.

Each arriving and departing aircraft corresponds to the colleagues.

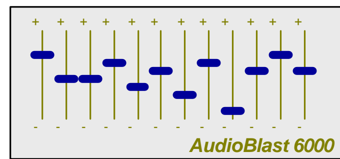
Consequences:

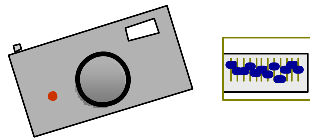
Constraints are localized within the Mediator. Changes to constraints need only be dealt with in the tower. Aircraft will still take off and land only when cleared to do so.

Aircraft interactions are decoupled. The many to many interactions are replaced with a one to many interaction. Each aircraft communicates with the tower, rather than with each other.

Control is centralized in the tower. Complexity of interaction between aircraft is traded for complexity in the mediator.

Memento





There can be an infinite number of settings for a piece of audio mixing equipment. An engineer could take a photograph of a particular setting, and use the photograph to restore the switch settings to the desired state if perturbed.

Behavioral

The *Memento* captures and externalizes an object's internal state, so the object can be restored to that state later.

Participant Correspondence:

The mixing equipment corresponds to the original object, whose state is being saved.

The Photograph is the memento.

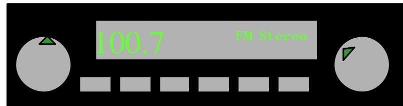
The engineer that takes the photo is the originator. He will also use the memento to restore the state of the switch settings.

The drawer where the memento is stored is the caretaker.

Consequences:

The photograph eliminates the need for everyone in the studio to know the switch settings in case they are perturbed. The photograph also stores information that the engineer should manage, outside of the engineer (i.e. not in his memory).

Memento



Most people are particular about the radio station that they listen to in the car. When there is more than one driver, (Father, Mother, Child), the radio station is likely to have changed with the driver. The preset buttons serve as mementos, allowing the radio to be restored to the desired tuning with one button push.

Behavioral

The *Memento* captures and externalizes an object's internal state, so the object can be restored to that state later.

Participant Correspondence:

The radio tuning corresponds to the original object, whose state is being saved.

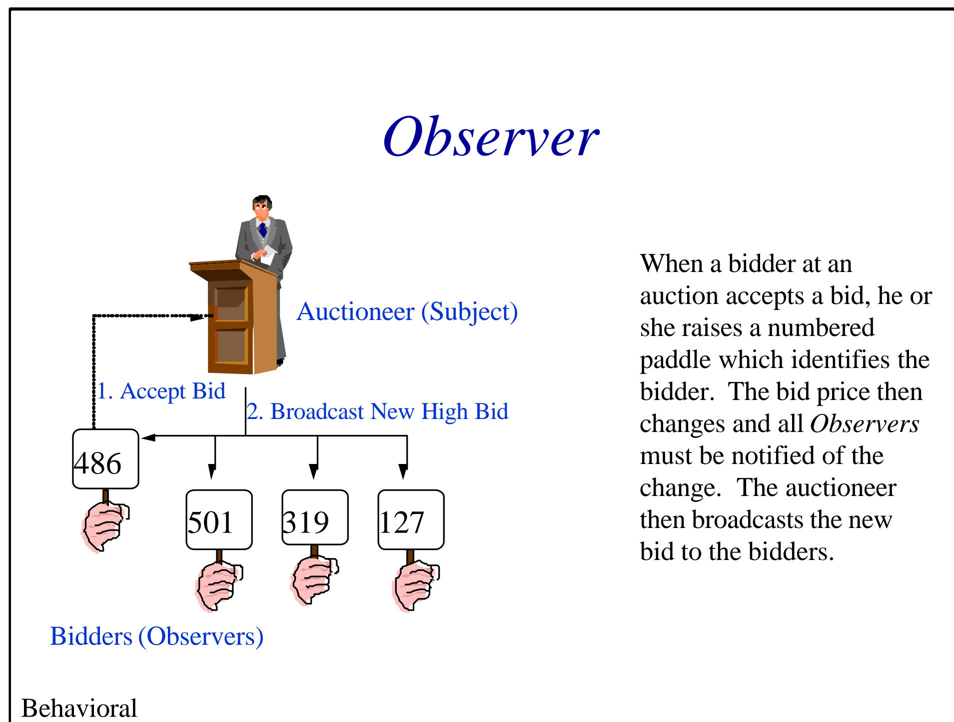
The preset button is the memento.

The driver who sets the preset button is the originator. He will also use the memento to restore the state of the radio tuning.

The radio where the button is located is the caretaker.

Consequences:

The button eliminates the need for the drivers to memorize the radio frequencies of their favorite stations. The preset buttons store the information so that the tuning can be restored.



The *Observer* defines a one to many relationship, so that when one object changes state, the others are notified and updated automatically.

Participant Correspondence:

The auctioneer corresponds to the Subject. He knows the observers, since they must register for the auction.

The current bid corresponds to the ConcreteSubject The observers are most interested in its state.

The bidders correspond to the Observers. They need to know when the current bid changes.

Each individual bidder with different tolerances for the bidding correspond to the ConcreteObservers.

Consequences:

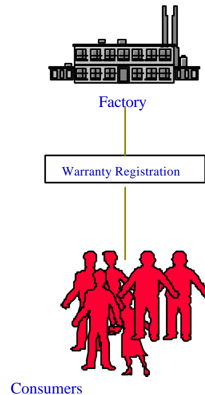
There is abstract coupling between the subject and observers. All that the auctioneer knows is that bidders will bid. He does not know when the price will become too steep for an individual bidder.

There is support for broadcast communication. When the auctioneer announces the current bid, that information is broadcast to all interested parties.

Observers can cause an avalanche of unexpected updates, since they can be blind to the ultimate cost of changing the subject. A bidder may intend to raise the bid by $50, and end up starting a bidding war.

Observer

Consumers who register for a product warranty are like observers. When the safety record of the product changes (like in a recall), all registered observers are notified.



Behavioral

The *Observer* defines a one to many relationship, so that when one object changes state, the others are notified and updated automatically.

Participant Correspondence:

The company corresponds to the Subject. It knows the observers, since they must register for the warranty.

The product safety/reliability record corresponds to the ConcreteSubject The observers are most interested in its state.

The consumers correspond to the Observers. They need to know when the current product safety/reliability record changes.

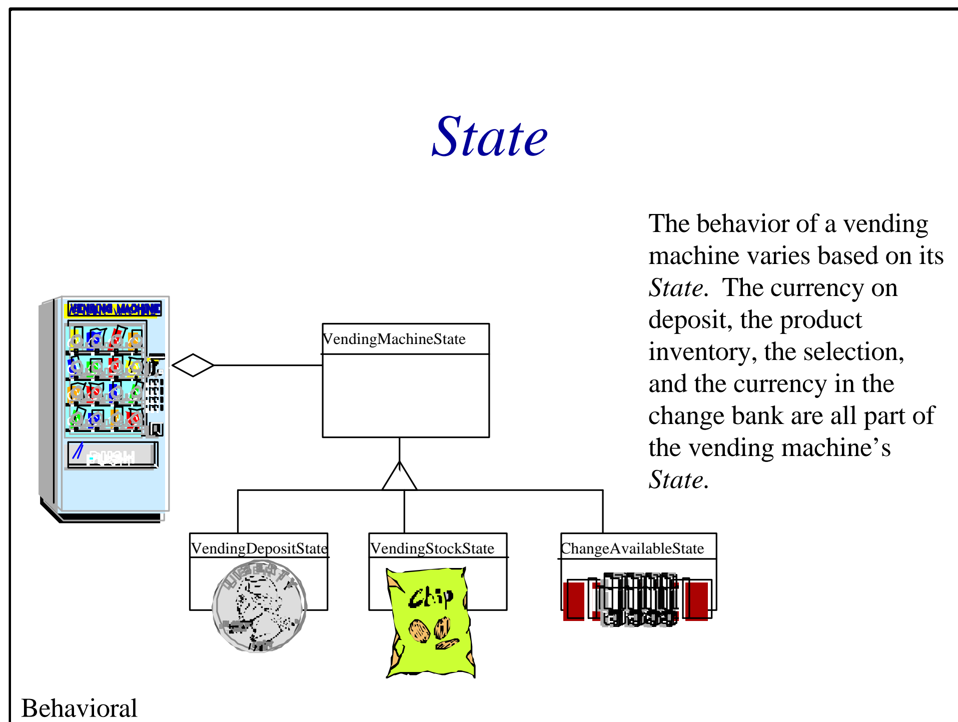
Each individual consumer corresponds to the ConcreteObservers, since they will have different experiences with the product.

Consequences:

There is abstract coupling between the subject and observers. All that the company knows is that consumers have registered for the warranty. It does not know which once will require warranty service.

There is support for broadcast communication. If a recall occurs, a form letter is sent out to all registered owners.

Observers can cause an avalanche of unexpected updates, since they can be blind to the ultimate cost of changing the subject. Consumers trying to obtain warranty service are not aware of other consumer's experience with the product. If enough claims are submitted, the product may be recalled.



The *State* pattern allows an object to change its behavior when its internal state changes.

Participant Correspondence:

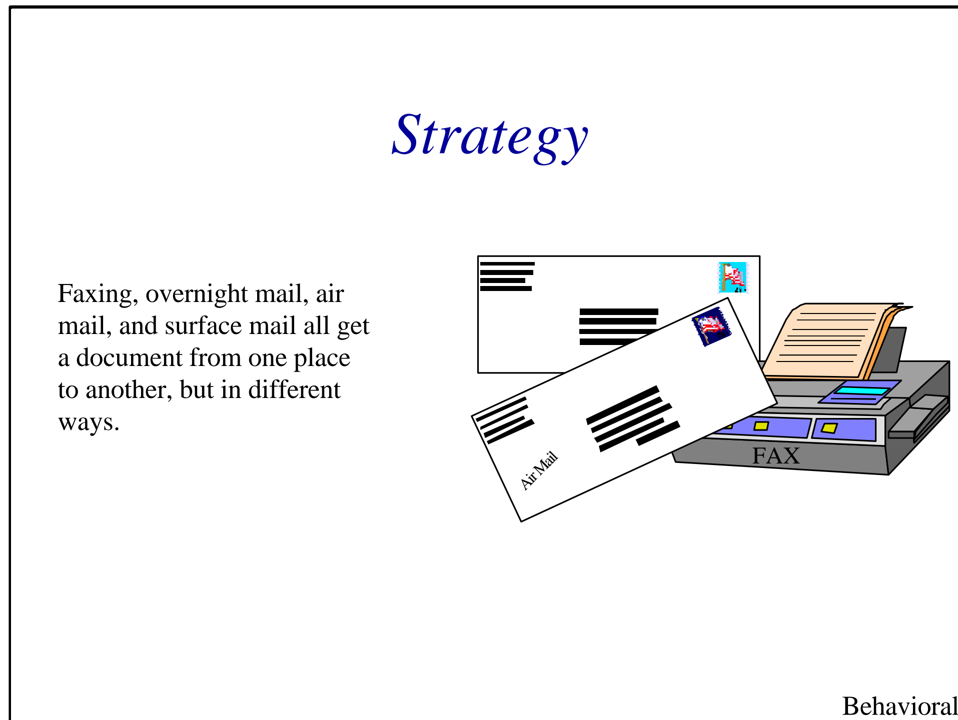
The deposit on hand, stock, and change on hand correspond to the context.

The behavior associated with each context corresponds to the state.

Consequences:

Behavior specific to a given state is localized. When the machine runs out of potato chips, the exact change light will not be illuminated, unless exact change is required.

State transitions are explicit. Product A will not be delivered unless the deposit on hand is sufficient for Product A.



A *Strategy* defines a set of algorithms that can be used interchangeably.

Participant Correspondence:

Text based communication corresponds to the Strategy.

Specific forms of text base communication, such as fax, mail, etc. correspond to the ConcreteStrategies.

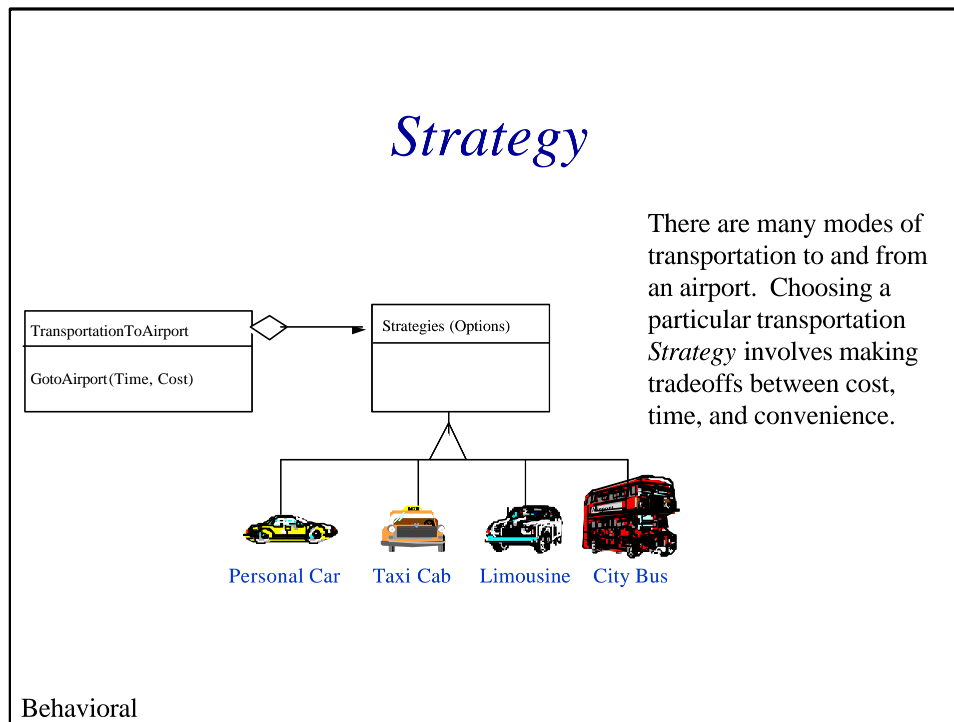
The context in which textural information is conveyed corresponds to the Context.

Consequences:

Strategies combine families of related algorithms. There are several ways to send textual information.

Strategies allow the algorithm to vary independently of the context. For example, if textual information must be delivered to the next office, placing on copy on your colleague's chair is a valid algorithm, but e-mail could also be used. The proximity does not necessarily dictate the algorithm.

Strategies offer a choice of implementations.



A *Strategy* defines a set of algorithms that can be used interchangeably.

Participant Correspondence:

Transportation to the airport corresponds to the Strategy.

Specific forms of transportation such as bus, taxi, etc. correspond to the ConcreteStrategies.

The context in which one must get to the airport (rush hour, maximum convenience, etc.) corresponds to the Context.

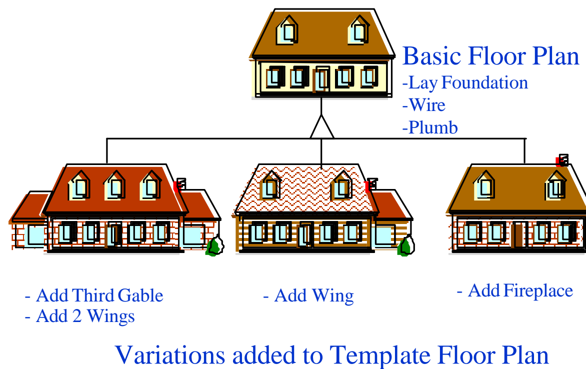
Consequences:

Strategies combine families of related algorithms. There are several ways to get to the airport.

Strategies allow the algorithm to vary independently of the context. The context alone does not dictate which method of transportation will be used.

Strategies offer a choice of implementations.

Template Method



Variations added to Template Floor Plan

Subdivision developers often use a *Template Method* to produce a variety of home models from a limited number of floor plans. The basic floor plans are a skeleton, and the differentiation is deferred until later in the building process.

Behavioral

The *Template Method* defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.

Participant Correspondence:

The basic floor plan corresponds to the AbstractClass.

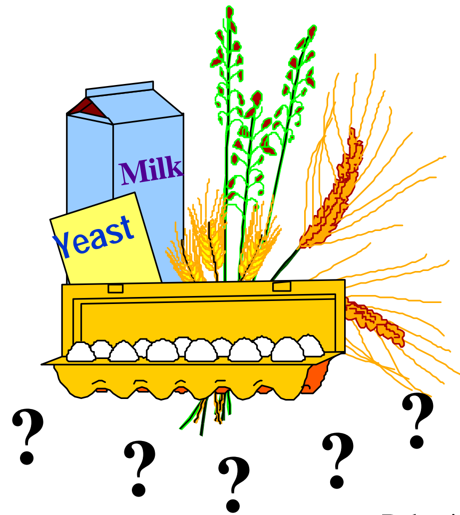
The different elevations correspond to the ConcreteClasses.

Consequences:

Templates factor out what is common, so that it can be reused. Multiple elevations can be built from a basic floor plan. The specific elevation does not need to be chosen until later in the process.

Template Method

Once a basic bread recipe
is developed, additional steps,
such as adding cinnamon,
raisins, nuts, peppers, cheese,
etc. can be used to create
different types of bread.



Behavioral

The *Template Method* defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.

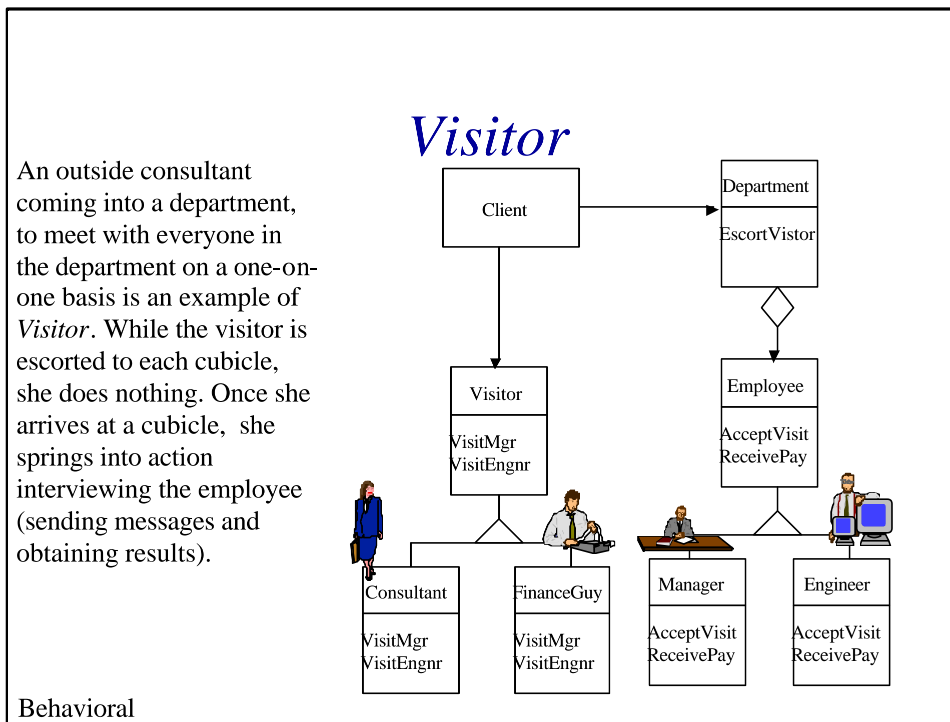
Participant Correspondence:

The basic bread recipe corresponds to the AbstractClass.

The additions to make the bread distinctive correspond the the ConcreteClass.

Consequences:

Templates factor out what is common. Multiple bread recipes can have a basic recipe in common.



The *Visitor* pattern represents an operation to be performed on the elements of an object structure, without changing the classes on which is operates.

Participant Correspondence:

The consultant corresponds to the Visitor.

The consultant also corresponds to the ConcreteVisitor, but purpose of the visit determines the ConcreteVisitor.

The element corresponds to the office visited.

The employee occupying the office corresponds to the ConcreteElement.

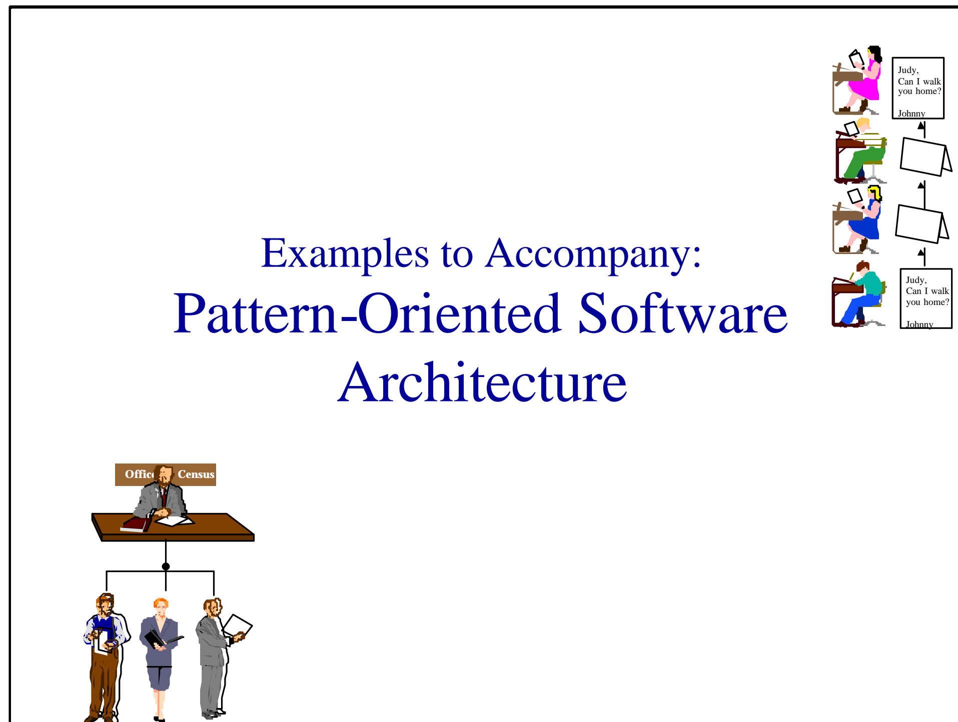
The schedule or agenda of the visit corresponds to the ObjectStructure.

Consequences:

The visitor makes adding new operations easy. By adding a new visitor, new operations can be added. For example, a survey visitor may be dispatched to survey employees. Once management finds out that they are stressed, a masseuse visitor may be sent around to relieve stress.

Visitors gather related operations, and separate unrelated ones. The engineering department is not typically involved in employee attitude surveys. It makes more sense to have a visitor, rather than an engineer conduct these.

Visitors often break encapsulation. In the real world, consultants are required to sign non-disclosure agreements, because encapsulation gets broken.



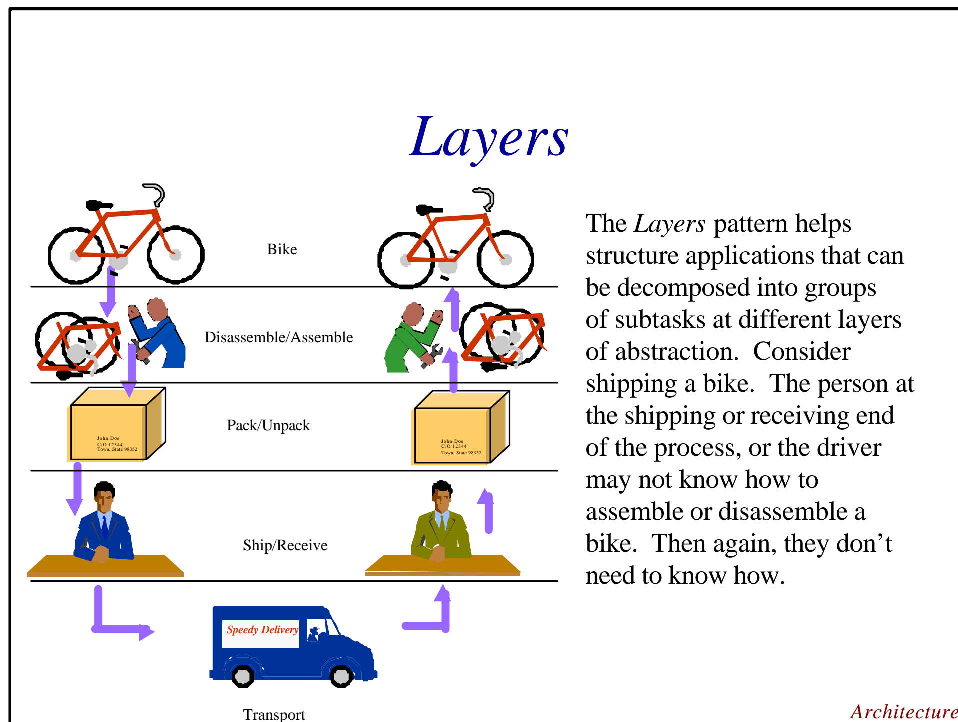
Pattern-Oriented Software Architecture (also known as PoSA) was written by Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal.

It was published by John Wiley and Sons in 1996.

The examples here are the result of an OOPSLA ‘98 workshop on Non-Software Examples of PoSA patterns, conducted by Michael Duell, Linda Rising, Peter Sommerlad and Michael Stal.

In addition to the workshop organizers, contributors to this body of work include Russ Frame, Kandi Frasier, Rik Smoody and Jun’ichi Suzuki.

© AG Communication Systems - 1999



The purpose of *Layers* is to structure applications that can be decomposed into different subtasks, with each subtask at an appropriate level of abstraction.

Participant Correspondence:

When considering disassembling a bike for transport, and reassembling it at its destination, each step corresponds to a layer.

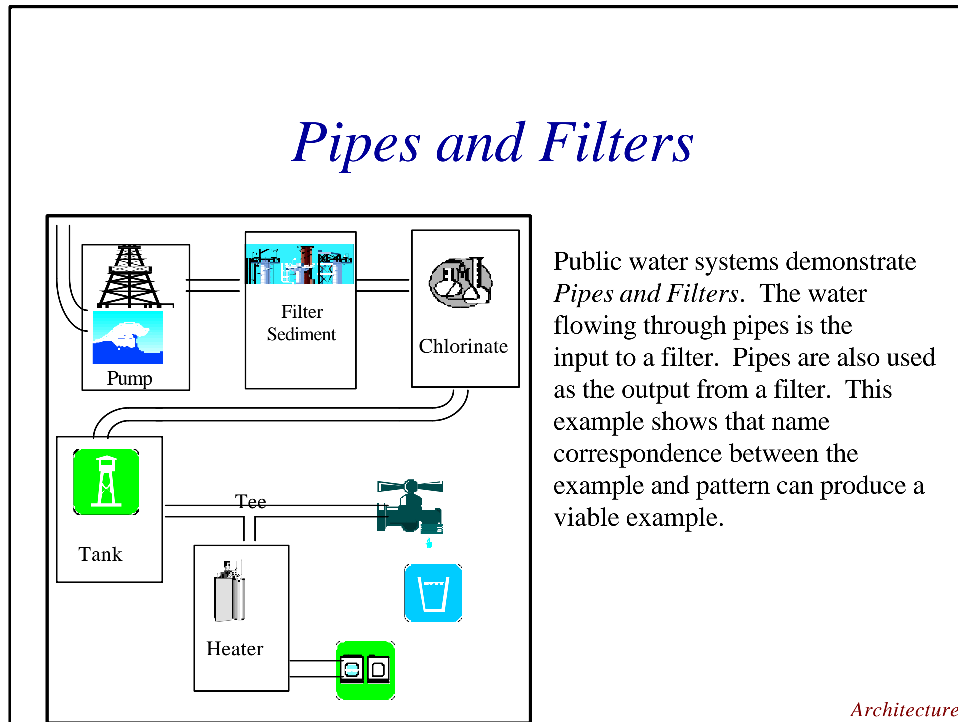
The disassembly/assembly, packing/unpacking, shipping/receiving are all tasks at different levels of abstraction.

Consequences:

The layers are exchangeable. The disassembler, packer, or shipping clerk can be changed without altering the basic structure of the example.

Layers can be used in a variety of context. The packer and pack items other than bicycles, just as the shipping clerk can ship items other than bicycles.

Dependencies are kept local. If a bike requires non-metric tools, the packing/unpacking and shipping/receiving layers are not impacted.



Pipes and Filters provides a structure for systems that process a stream of data.

Participant Correspondence:

In the public water system example, the water pipes correspond to data pipes.

The various stages that add something to or take something from the water correspond to the filters.

Consequences:

Changing filters adds flexibility. For example, fluoride could be added to the water supply without upsetting the overall system.

Recombining filters adds flexibility. Many homes have water treatment systems that soften water used for cleaning and filter chemicals from the drinking water. All of these home systems take advantage of water supplied via the upstream pipes and filters.

Blackboard

Watch just about any police show on television, and notice that the crimes cannot be solved immediately. There is often a blackboard with forensic evidence, ballistic reports, crime scene data, etc. posted. When all of the experts add their pieces of the puzzle, the police are able to solve the crime!

Architecture

The *Blackboard* pattern is used to assemble knowledge from a variety of sources when no deterministic solution strategy is known.

Participant Correspondence:

Obviously the blackboard corresponds to the blackboard class (where data is centrally managed).

Each person having subject matter expertise corresponds to a knowledge source.

The lead detective is the controller, who brings together the subject matter experts, and manages access to the blackboard.

Consequences:

It is easy to experiment with different approaches when data from different sources is brought together.

It is easy to change and maintain the blackboard, because the independence between the participants. If a crime involved a knife, there would be no need to involve a ballistics subject matter expert.

A blackboard is robust. Only conclusions supported by data and other hypothesis survive. All others are weeded out.

Broker

A travel agent is a broker for travel related services. A customer dealing with a travel agent can book passage on ships, planes and trains, reserve a hotel room and rental car, and book tours. The customer deals only with the travel agent, although several companies are involved. Note that there are often complex codes on the itinerary. The broker and travel services companies understand these codes. The customer often does not.

Architecture

The *Broker* pattern is used to structure distributed systems with decoupled components that interact by remote service invocations. The Broker component is responsible for coordinating communication between clients and servers.

Participant Correspondence:

The Travel Agent's client corresponds to the Client.

The Travel Agent corresponds the Broker.

The Travel Services, such as airlines, rental cars, hotels, etc. correspond to the Servers.

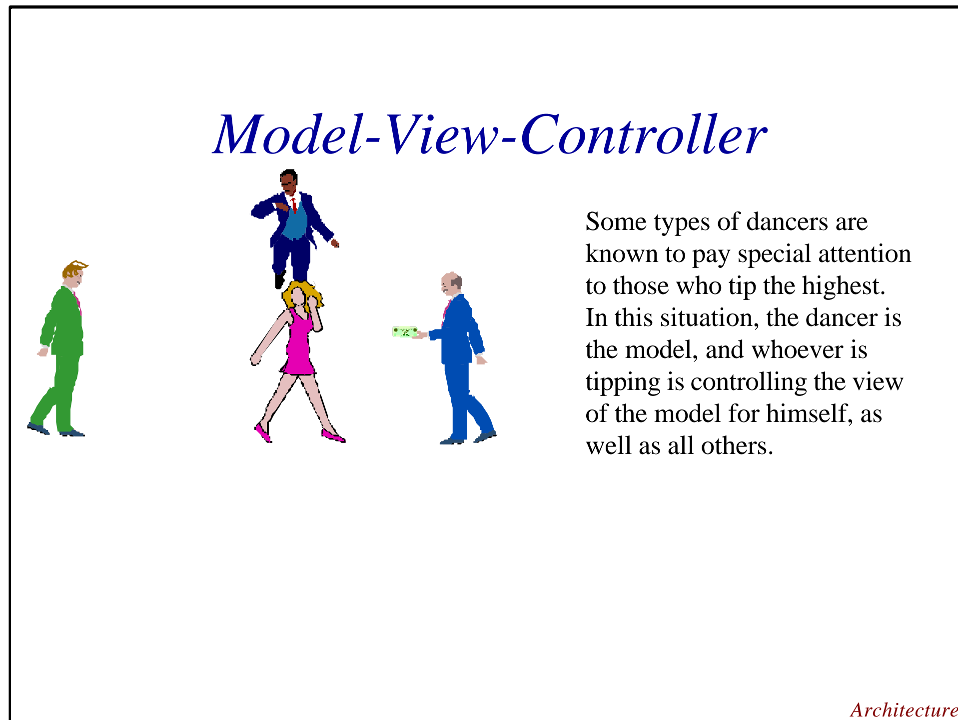
The reservation systems for the Travel Services correspond to the Server-side Proxies.

Consequences:

The Travel Agent, like the Broker offers location transparency. The client does not need to know where particular travel services are located.

If a server changes, the interface is the same to the client. The interface is the same to the client, regardless of the airline booked.

Details of internal systems are hidden. A travel agent switching from SABRE to APOLLO reservation system is transparent to the client.



The *Model-View-Controller* pattern divides an interactive application into a model (or core data), view (display) and controller.

Participant Correspondence:

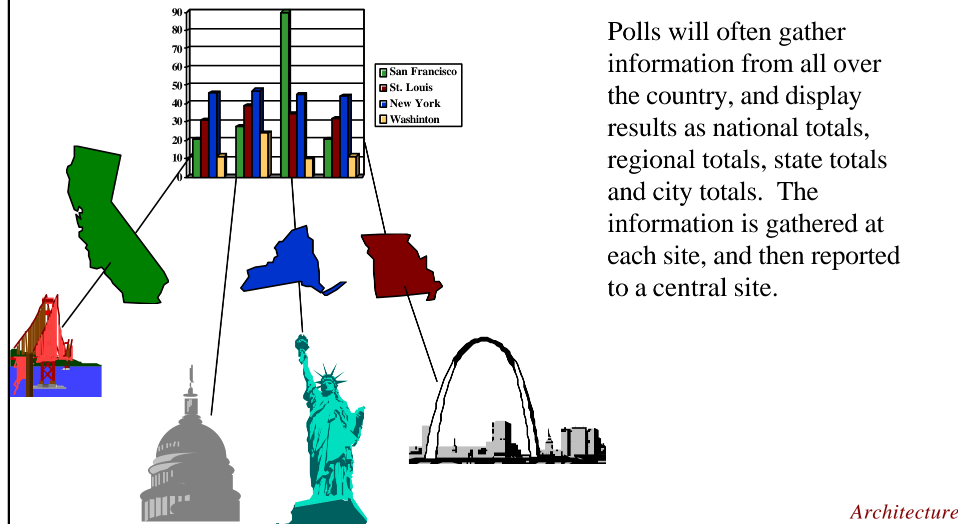
The dancer corresponds to the model.

The view of the dancer from different vantage points corresponds to the view of the model.

The big tipper, corresponds to the controller, since the view of the model changes while he is getting special attention.

Consequences:

With a dancer, there are multiple views of the same model. These views are synchronized, since the view of a model is changing simultaneously as she walks away from one observer, and towards another.



The *Presentation-Abstraction-Controller* pattern defines a structure for interactive systems in the form of a hierarchy of cooperating agents. Each agent is responsible for a specific aspect of the application's functionality.

Participant Correspondence:

The national data repository corresponds to the Top-level Agent. It controls the PAC hierarchy.

The regional and state data repositories correspond to the Intermediate-level Agents.

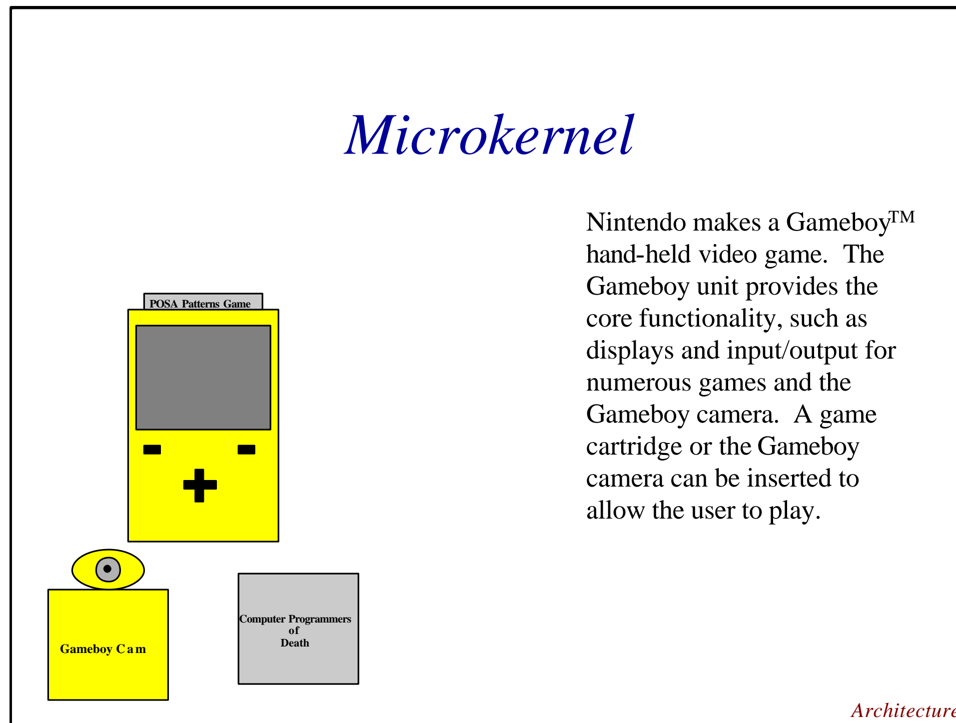
The city data repositories correspond to the Bottom-level Agents.

Consequences:

Detailed information can be obtained from each level. A synopsis of voter behavior for a city can be obtained at the city level.

The model is easily extendible. A county agent can be added between the city and state level, without impacting the regional and national levels.

Multi-tasking is supported. Voter behavior for multiple cities can be examined simultaneously.



The *Microkernel* pattern separates the minimal functional core from extended functionality.

Participant Correspondence:

The player corresponds to the client. The Gameboy™ unit corresponds to the microkernel, since it contains the core functionality.

The game cartridges correspond to the internal servers, which provide additional functionality.

The pin connections on the unit and cartridges correspond to the adapter.

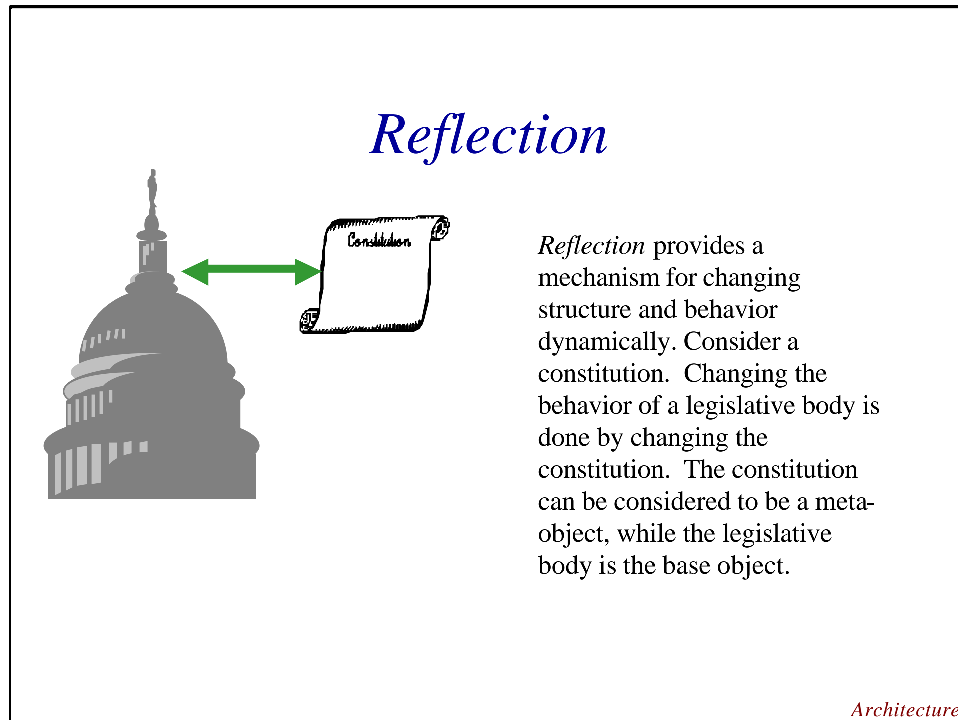
The buttons on the unit provide an interface to clients, and therefore correspond to the external server.

Consequences:

The client applications can be easily shared between hardware without porting to a new environment. Kids can trade game cartridges without violating software copyrights.

The Gameboy is very flexible. Originally it was only for games. Eventually, it could be used as a camera.

The Gameboy has separated policy and mechanism, so that new external servers could be added that implement their own views. For example, on the Gameboy camera cartridge, the lens is a new external server.



The *Reflection* pattern provides a mechanism for changing structure and behavior of a system dynamically. A system is structured as a base level, and meta level. The meta level contains information about system properties. The base level exhibits the application structure and behavior. Changes to meta level information affect subsequent base level structure and behavior.

Participant Correspondence:

The Constitution corresponds to the meta level. It contains information on how the congress is to conduct itself.

The congress is the base level. It conducts itself according to the Constitution. Changes to the Constitution affect how the congress behaves.

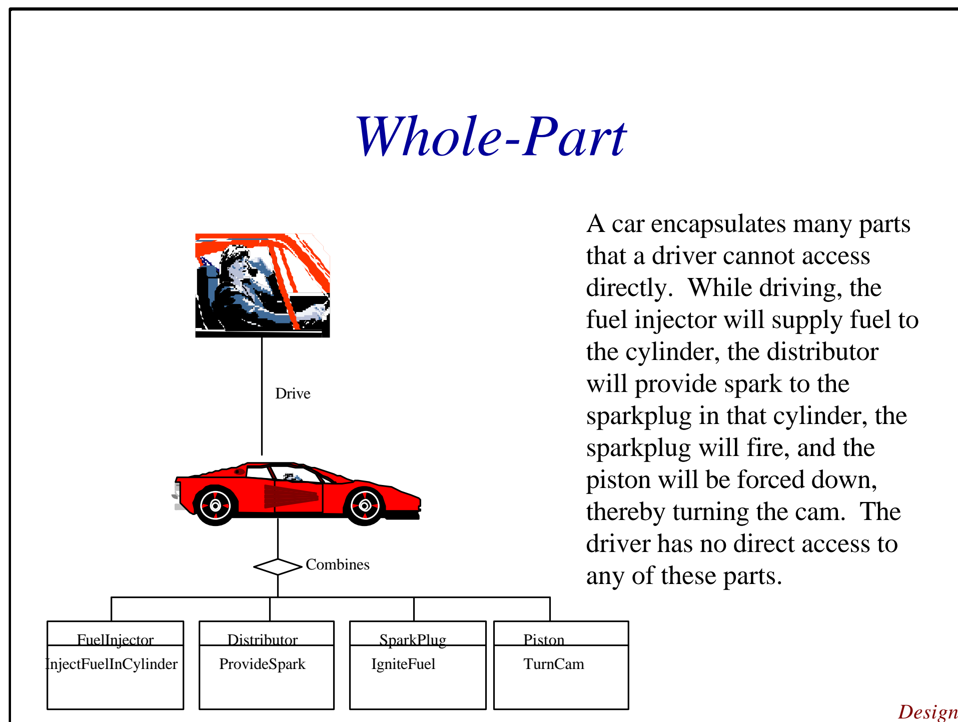
Consequences:

There is no explicit modification of the base. A change is accomplished by calling a function in the metaobject protocol. For example, the makeup of congress changes every two years due to elections as outlined in the Constitution.

Change to the system is easy. The Constitution provides safe and uniform mechanism for changing the congress.

Many types of change are supported. Looking at the types of change that have can affect the structure of congress, elections, representation based on census, and adding a new state can cause changes in structure.

Meta level changes can damage the system, just as changes to the Constitution can cause damage.



The *Whole-Part* pattern helps aggregate components that form a semantic unit.

Participant Correspondence:

The driver corresponds to the client. The driver will ultimately use all of the components through the interface of the whole.

The car corresponds to the Whole. It is made up of smaller systems.

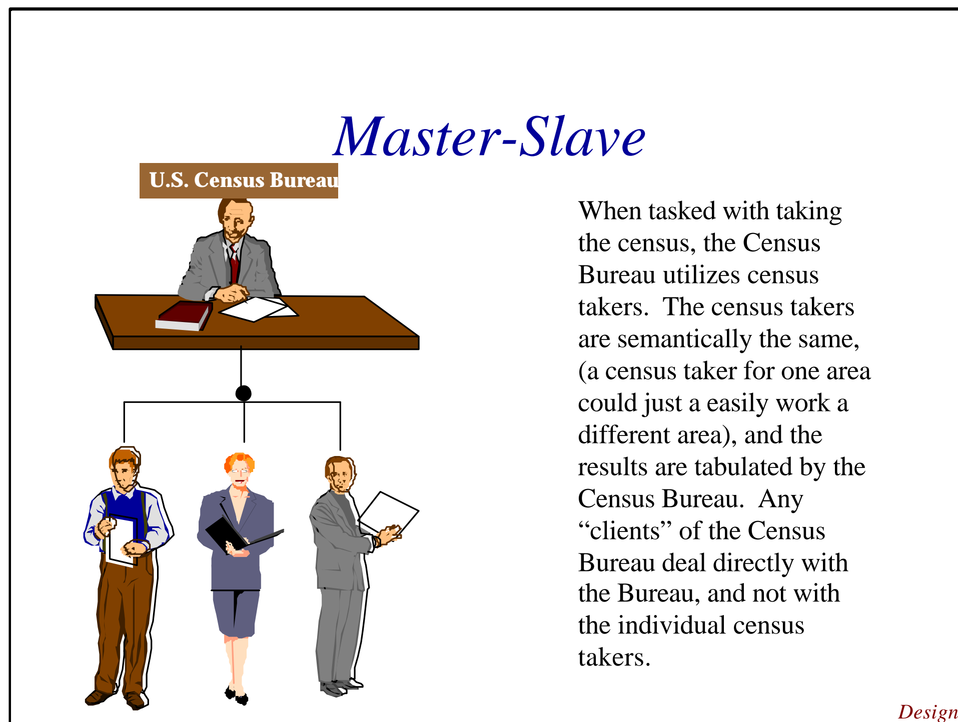
The individual systems, such as the fuel system, ignition system, cooling system, etc. correspond to the Parts.

Consequences:

Parts are changeable without impacting the client. If a starter fails, it can be replaced, and the interface to the care remains the same for the driver.

Each concern is implemented by a separate part, making it easier to implement complex strategies. Many of the parts on a car are fairly simple, yet the composite is quite complex.

Reusability is supported since parts of a whole can be used in other aggregates. The auto salvage industry is build on this concept. A part can be taken from one car for use on another.



A master component distributes work to identical slave components and computes a final result from the results when the slaves return

Participant Correspondence:

The congress corresponds to the client. Congress authorizes the census, and uses its results.

The Census Bureau corresponds to the Master. It is responsible for taking the census, and does so by dividing the work into smaller tasks.

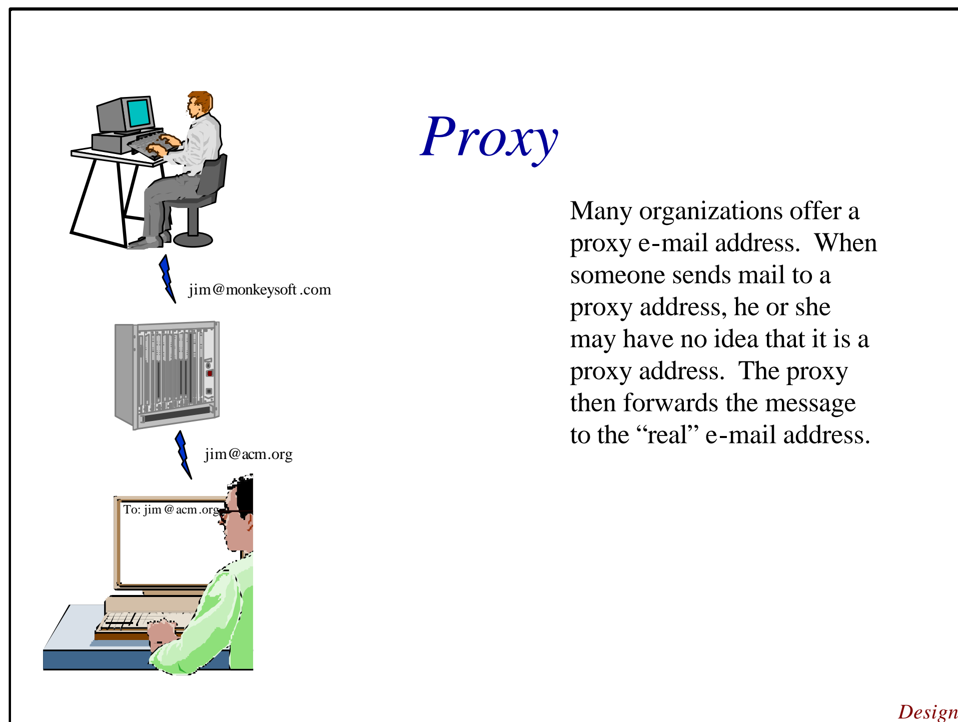
The Census takers correspond to the Slaves. They report the work to the Census Bureau. They are interchangeable.

Consequences:

Makes slaves exchangeable and extensible. A census taker can cover any designated area. If one finishes early, he can be reassigned to another area.

Separates slave and client code from the work partitioning code. Neither congress or the individual census takers are concerned with how work is partitioned.

Allows for parallel computing. With multiple census takers, they can all be working simultaneously. There is no need to finish one area before starting another.



The *Proxy* provides a surrogate or place holder to provide access to an object.

Participant Correspondence:

The organization e-mail address corresponds to the Proxy.
The "real" e-mail address corresponds to the Original.

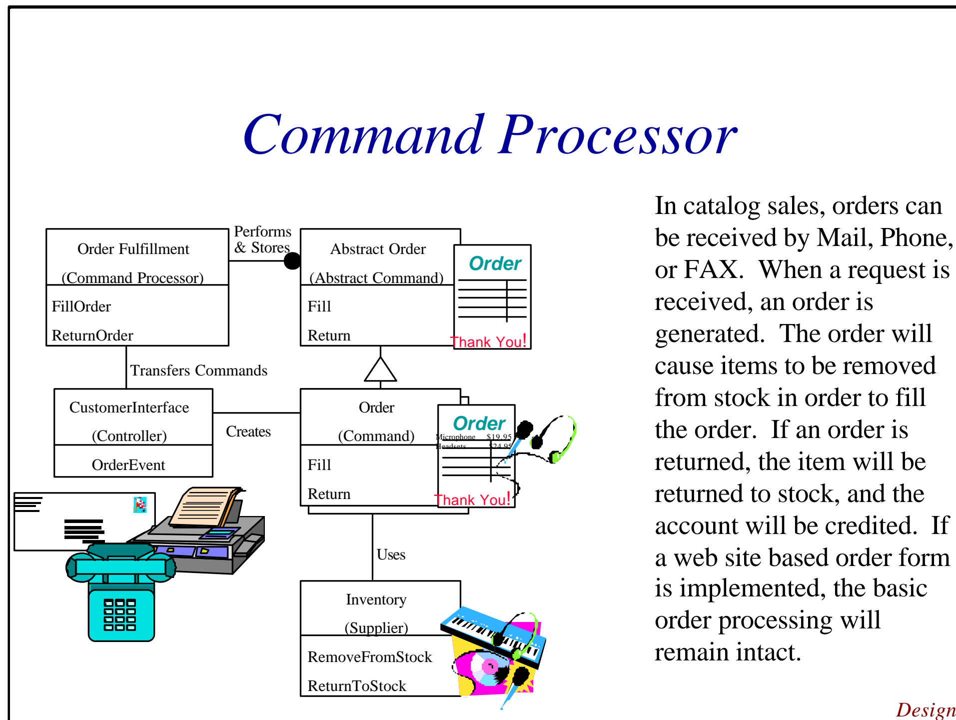
The individual having the proxy e-mail and original e-mail address corresponds to the AbstractOriginal.

The person sending e-mail to the AbstractOriginal corresponds to the Client.

Consequences:

The proxy introduces a level of indirection when accessing an object. The indirection can be used for security, or disguising the fact that an object is located in a different address space.

In the case of the e-mail example, the AbstractOriginal can change real e-mail addresses (like when changing companies) and still receive e-mail through the proxy.



The *Command Processor* pattern separates the request for a service from its execution.

Participant Correspondence:

The Customer Interface corresponds to the Controller.

The Order Fulfillment department corresponds to the Command Processor. This department is responsible for processing the order, making sure that an item is taken from stock, and sent to the customer. If the item is returned, the order fulfillment department will return the item to stock.
The order corresponds to the Capitalize command. Once a phone call, order form, or fax is received, it becomes a record is created containing the customer name, address, desired items, and method of payment.

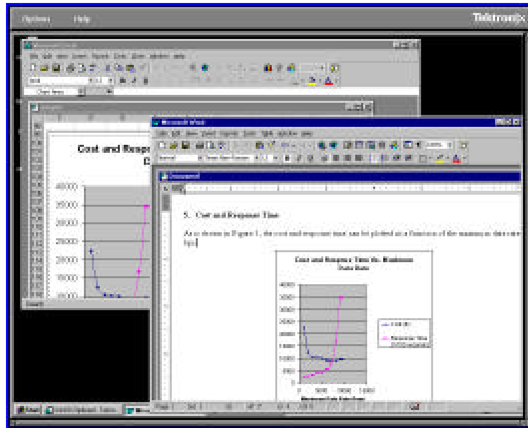
The inventory corresponds to the supplier.

Consequences:

There is flexibility in the way that requests are activated. A customer can order via mail, phone, or FAX. If a web site based order form is implemented, the basic order processing will remain intact.

Commands can be implemented concurrently. Several customers will request items daily. The orders are placed in bins, and shipped when complete.

View Handler



Although window managers are common to software, the visual nature of windows allows one to examine behavior without looking at source code. The window manager opens, manipulates, and disposes of views. It also coordinates dependencies between views, and coordinates their updates.

Design

The *View Handler* pattern helps manage all views provided by a system.

Participant Correspondence:

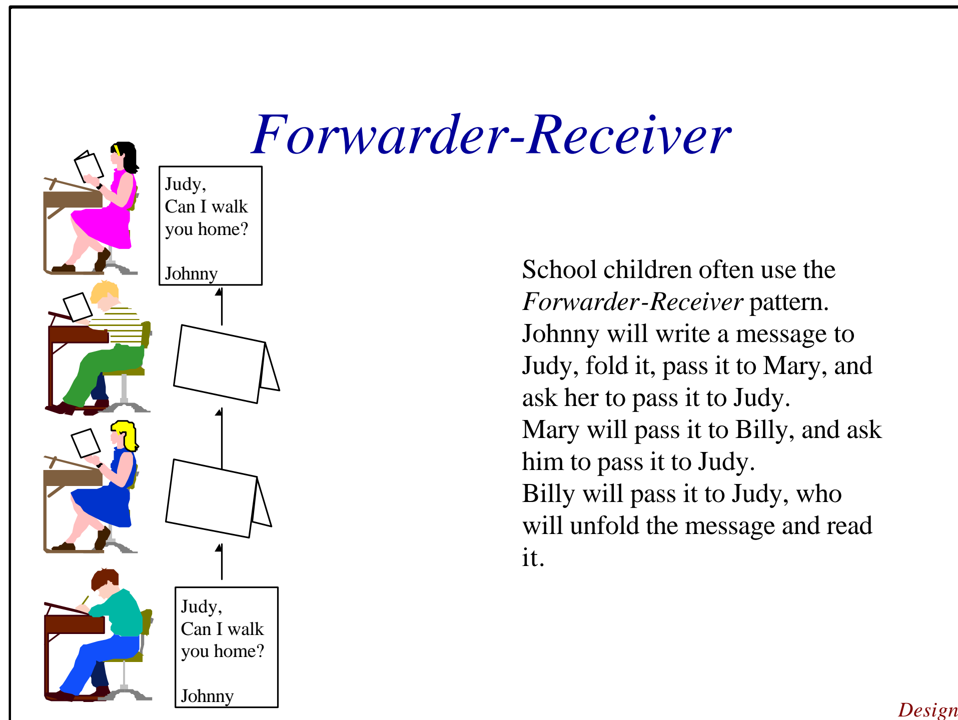
The windowing system corresponds to the view handler.

The displayed windows correspond to the views.

The underlying files or documents correspond to the supplier(s)

Consequences:

The *View Handler* provides uniform handling of views. New views can be added without affecting existing views.



The *Forwarder-Receiver* pattern introduces a peer-to-peer communication model that decouples the forwarders and receivers from the underlying communication mechanisms.

Participant Correspondence:

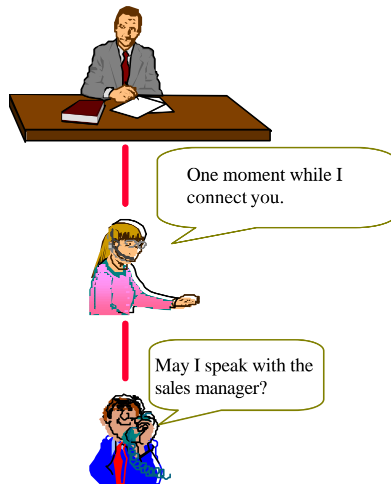
Each student corresponds to a peer. Students will act as Forwarders when they pass a note, and Receivers when they receive the note.

Consequences:

Note passing is efficient inter-process communication. A forwarder knows the location of potential receivers, but does not need to know the location of the ultimate destination.

A change in communication mechanism does not impact forwarders and receivers. For instance, if the note were written in French, it would still be passed in the same manner.

Client-Dispatcher-Server



When a client calls a company to talk with an individual, a receptionist often answers the call, and then routes it to the appropriate person. The client cannot tell if the person is in the same building, or at a remote site. Furthermore, the client does not know if the phone rings on the sales manager's desk, or if the receptionist puts him on hold, and yells, "Hey Larry, pick up the phone!"

Design

The *Client-Dispatcher-Sever* introduces an intermediate layer between clients and servers to provide location transparency and hide connection details.

Participant Correspondence:

The customer corresponds to the Client.

The receptionist corresponds to the Dispatcher.

The sales manager (or anyone else being called) corresponds to the server.

Consequences:

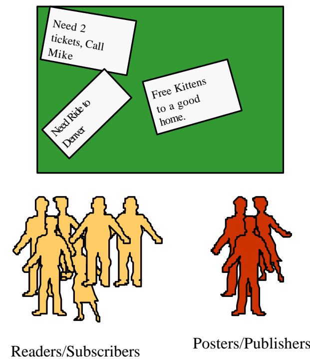
The *Client-Dispatcher-Server* introduces an intermediate layer between clients and servers.

It provides location transparency, by means of a name service.

It hides the details of the communication connection between clients and servers.

If a new sales manager is hired, the customer is not required to have this knowledge in order to reach him.

Publisher-Subscriber



To communicate to others, one can post or publish a notice on a bulletin board. The subscribers (readers) of the bulletin board can vary dynamically. State information, such as having extra tickets can be communicated to unknown individuals when it is not feasible to poll people explicitly to determine if they require the tickets.

Design

The *Publisher-Subscriber* pattern helps to keep the state of cooperating agents synchronized. A publisher notifies any number of subscribers of changes in state.

Participant Correspondence:

Someone posting a message on a bulletin board corresponds to the publisher. Someone reading the bulletin board corresponds to a subscriber.

Consequences:

Changes in state must be communicated to unknown parties. The number of subscribers can vary dynamically.

When an individual needs tickets to an event, a message is posted indicating that their state is “NeedTickets”. When tickets are obtained, the message is removed.