TABLA DE CONTENIDOS

ORGANIZACIÓN DE LA TESIS

Básicamente, esta tesis se proyecta en cuatro secciones:

- Resumen
- Introducción
- Desarrollo
- Conclusiones

Posteriormente, se presentará una breve explicación del contenido de cada una de estas partes, respetando el orden en el cual se encontrarán desarrolladas.

Resumen

En esta sección, se presenta el fundamento del comportamiento subjetivo en el mundo de los objetos, el objetivo de esta tesis y el alcance de la misma.

Introducción

La introducción expone una motivación más precisa de la incorporación del comportamiento subjetivo en el paradigma orientado a objetos y la definición de este nuevo concepto.

Desarrollo

Dicha sección, abarca básicamente dos sub-secciones:

Sección 1. El análisis completo e implantación del comportamiento subjetivo en el paradigma orientado a objetos.

Sección 2. El análisis de la introducción del comportamiento subjetivo en un dominio de aplicación particular: un *ambiente virtual cooperativo orientado a objetos.*

La **Sección 1** engloba:

- Las consecuencias de la incorporación del comportamiento subjetivo en el paradigma orientado a objetos:
  - Los conceptos del paradigma extendidos por el comportamiento subjetivo.
  - El impacto del comportamiento subjetivo sobre los conceptos básicos de la orientación a objetos.
- Nuevos conceptos introducidos por el comportamiento subjetivo.
- La organización estructural y la mecánica de evaluación del comportamiento subjetivo.
- La presentación de un modelo de clases del comportamiento subjetivo.
- La implementación del comportamiento subjetivo en un ambiente de programación orientado a objetos.

La **Sección 2** abarca:

- La definición y características básicas de un ambiente virtual cooperativo orientado a objetos (OOCVE).
- La definición de comportamiento subjetivo en un OOCVE.
  - Reutilización de los elementos definidos en el comportamiento subjetivo en el paradigma orientado a objetos.
  - Definición de nuevos elementos.

- Evaluación sobre la introducción de comportamiento subjetivo en un OOCVE.

Conclusiones

Presenta las conclusiones personales de la autora de esta tesis.

FUNDAMENTO DEL COMPORTAMIENTO SUBJETIVO

Las distintas metodologías presentes en el paradigma orientado a objetos proveen una forma particular de resolver problemas que facilita la posibilidad de representar con gran fidelidad los problemas del mundo real. Sin embargo, ciertos problemas de nuestro mundo suelen acontecer con una determinada complejidad que incapacitan la elaboración de un modelo adecuado con las actuales herramientas y técnicas de desarrollo orientadas a objetos.

Se puede observar que en la realidad, algunos "objetos" (como los animales, seres humanes, vegetales, etc.), en circunstancias específicas, se comportan de acuerdo a factores que los influyen de alguna forma, es decir, son capaces de responder de diversas maneras a un mismo requerimiento, dependiendo de las situaciones en las que se encuentran inmersos. Concretamente, esta capacidad es lo que intenta representar el comportamiento subjetivo en el paradigma orientado a objetos.

Objetivos

Los objetivos de esta tesis son:

- Definir y modelar el comportamiento subjetivo utilizando como base la tesis de trabajo de grado "*Subjetividad en un Ambiente de Objetos"*, e implementarlo en un ambiente de programación orientado a objetos. Se utiliza para este propósito, el ambiente de programación Visualworks® Non-Commercial Release 3.0. (Copyright © 1998 objectshare, Inc.).
- Analizar la utilización del comportamiento subjetivo en un dominio de aplicación particular: un *ambiente virtual cooperativo orientado a objetos*.

Alcance

Para cumplir con los objetivos se define el siguiente alcance:

1. Fundamentar la incorporación de comportamiento subjetivo en el mundo de los objetos.
2. Definir el concepto de comportamiento subjetivo en el paradigma orientado a objetos.
3. Determinar cómo influye el comportamiento subjetivo sobre los conceptos básicos de la orientación a objetos.
4. Definir y modelar los diferentes factores que pueden influir el comportamiento de un objeto y la manera en que lo hacen.
5. Definir y modelar el mecanismo necesario para resolver cómo debe responderse un mensaje, de acuerdo a los factores que están influyendo al objeto que lo recibe.
6. Extender el ambiente de programación elegido para permitir el manejo de comportamiento subjetivo, lo cual implica:
   a. Implementar los modelos anteriormente listados.
   b. Extender las herramientas para el desarrollo de aplicaciones, de manera que sea posible utilizar en una aplicación comportamiento subjetivo de manera natural.
7. Definir un ambiente virtual cooperativo orientado a objetos.
8. Analizar la introducción de comportamiento subjetivo en un OOCVE.
9. Determinar las extensiones y redefiniciones necesarias para introducir comportamiento subjetivo en un OOCVE.

MOTIVACIÓN DEL COMPORTAMIENTO SUBJETIVO

Comportamiento tradicional vs. Comportamiento subjetivo

En el mundo de los objetos, el medio de comunicación para ellos es el envío de mensajes. Un objeto puede enviar un mensaje a otro objeto para solicitar cierto servicio; si dicho mensaje es entendido, el objeto receptor dará una respuesta, la cual fue previamente especificada en un método cuando se definió el mensaje.

De esta forma, un objeto responde siempre de la misma manera cuando recibe un mensaje, ya que tiene asociada una simple implementación[1]. Sin embargo, es importante considerar que un objeto no es una entidad autónoma, es decir, está rodeado de otros objetos con los cuales, directa o indirectamente, necesita interactuar para cumplir con sus responsabilidades. En muchos casos, cierta característica de alguno de estos objetos, o características propias del mismo objeto pueden influir en su forma de comportarse.

Se enseñará un claro ejemplo del mundo real. Considere una aplicación que modela el sistema de manejo de fondos de un banco. Básicamente, pueden existir objetos que modelen cajas de ahorro, cuentas corrientes, usuarios, etc. En un momento dado, un usuario podría solicitar la extracción a una caja de ahorro de cierta cantidad de dinero. La respuesta que brindará dicha caja dependerá de determinados factores, por ejemplo, si tiene dinero suficiente para satisfacer el pedido, si el usuario que requiere de la extracción es el propietario de la caja, si la misma se encuentra habilitada o fuera de servicio, etc. Desde el instante que el usuario solicita una extracción de la caja de ahorro, existen diversos factores que pueden influir en la respuesta a brindar por dicha caja: si tiene dinero suficiente para satisfacer el pedido, la caja podría responder decrementando su monto y notificando el éxito de la extracción, o si el usuario no es el propietario de la caja, la misma podría notificar el error, etc.

De este modo, podría lograrse comportamiento un poco más "inteligente", otorgándole a un objeto la capacidad de brindar una respuesta adecuada de un conjunto de posibles respuestas asociado a un mismo mensaje, cuya elección dependa de alguna característica de un determinado objeto, o de alguna característica propia del objeto.

De esta manera, se puede afirmar que, en el paradigma tradicional orientado a objetos, un objeto reacciona objetivamente ante el envío de un mensaje; su respuesta se basa unívocamente en la especificación del método asociado a dicho mensaje. El concepto de comportamiento subjetivo se presenta como una extensión del mecanismo de comportamiento estándar, en el sentido de que habilita en un objeto la capacidad de reaccionar subjetivamente cuando recibe un mensaje, es decir, de brindar diferentes respuestas dependiendo de factores internos y/o externos que afectan al objeto en ese momento.

A continuación, se presentará la definición de comportamiento subjetivo en el paradigma orientado a objetos.

COMPORTAMIENTO SUBJETIVO

Definición

Se define *comportamiento subjetivo* como la habilidad en un objeto de poder brindar ante un mismo mensaje diferentes respuestas, dependiendo de factores internos y/o externos que lo afectan de alguna manera en ese momento.

[1] Los términos *implementación de mensaje* y *método* se usarán de manera intercambiable.

CONSECUENCIAS DE LA INCORPORACION DE COMPORTAMIENTO SUBJETIVO EN EL PARADIGMA ORIENTADO A OBJETOS

A medida que se profundice en el tema del comportamiento subjetivo, aparecerán nuevos conceptos, como consecuencia de la nueva capacidad que añade en la mecánica del comportamiento tradicional de los objetos. Dichos conceptos pueden definirse como una extensión de los actualmente definidos en el paradigma orientado a objetos; a continuación, se exponen cada uno de ellos.

Extensión de conceptos

Mensaje subjetivo

Un mensaje es una solicitud en un objeto para que lleve a cabo un servicio. Dicho mensaje es subjetivo cuando la forma en que se resuelve el servicio puede variar, dependiendo de ciertos factores que influyen en el objeto. Por lo tanto, un *mensaje subjetivo* es aquel que tiene asociadas varias formas de resolverse, y cualquiera de ellas puede ser utilizada.

Respuesta subjetiva

Cuando un objeto recibe un mensaje tradicional[2], reacciona con un determinado comportamiento y produce un cierto resultado, que comúnmente se conoce como "la respuesta del objeto". El envío de un mensaje subjetivo conduce a que el objeto brinde una de diversas respuestas posibles, debido a que dicho mensaje tiene asociadas diferentes formas de resolver el requerimiento por el objeto. De esta forma, una *respuesta subjetiva* es una de las posibles formas que tiene un objeto de resolver un mensaje subjetivo.

Método subjetivo

Cada respuesta que brinda un objeto cuando recibe un mensaje tradicional, se encuentra implementada en un método. Cada forma de resolver un mensaje subjetivo, o mejor dicho, cada respuesta subjetiva se encuentra implementada por un método denominado *método subjetivo*. En la evaluación de un mensaje subjetivo, cuando se determina cuál es el método subjetivo que lo resolverá, la forma de ejecución del mismo es análoga a la tradicional. Cabe destacar que todos los métodos subjetivos asociados a un mismo mensaje subjetivo son polimórficos entre sí, de modo que cualquiera de ellos pueda ser ejecutado.

Clase subjetiva

En una clase se definen los mensajes que entenderán sus instancias. Una clase es subjetiva cuando define al menos un mensaje subjetivo, o hereda de alguna de sus superclases al menos un mensaje subjetivo, sin redefinirlo.

Objeto subjetivo

Un objeto es subjetivo cuando es instancia de una clase subjetiva. Es importante destacar que un objeto subjetivo puede tener comportamiento subjetivo, ya que dentro del protocolo que entiende existe uno o más mensajes subjetivos.

[2] Por mensaje "tradicional" se entiende a un mensaje que no es subjetivo.

Ambiente de Programación Orientado a Objetos subjetivo

Como se mencionó anteriormente, ciertos objetos del mundo real tienen la capacidad de, en determinadas circunstancias, reaccionar subjetivamente dependiendo de los factores que los influyen en ese momento. Sin embargo, no para todos los objetos tiene sentido atribuirles capacidades de subjetividad en su comportamiento. Este hecho fomenta la idea de que no debe forzarse a un ambiente puramente subjetivo, en donde todos los objetos sean capaces de comportarse subjetivamente; es útil poder contar con un ambiente mixto, es decir, un ambiente orientado a objetos donde el programador pueda incorporar comportamiento subjetivo en las clases[3] que desee. De este modo, se obtiene una clara extensión de un ambiente de programación orientado a objetos, y no una restricción del mismo. Por lo tanto, un ambiente de programación orientado a objetos es subjetivo cuando soporta la definición de comportamiento subjetivo en las clases.

Impacto sobre conceptos básicos de la orientación a objetos

Dado que el comportamiento subjetivo afecta a un rasgo importante de los objetos (el comportamiento), cabe notar que su inclusión en el paradigma puede tener impacto en algunos conceptos básicos. Concretamente, debe resaltarse su relación con:

- Sharing de información
- Polimorfismo
- Encapsulamiento

A continuación se presenta la relación del comportamiento subjetivo con dichos conceptos.

Sharing de informacion

Debido a que el comportamiento subjetivo definido en esta tesis se aplica en un ambiente basado en clases, se tendrá sólo en cuenta uno de los mecanismos de *sharing de información*: la herencia.

Herencia en un Ambiente de Programación Orientado a Objetos sin Comportamiento Subjetivo

La herencia define una de las jerarquías de clases más importante en los sistemas orientados a objetos. Básicamente, representa una jerarquía de abstracciones, en la que una subclase hereda estructura y comportamiento de una o más superclases.

Semánticamente, la herencia denota una relación "*es un*". Por ejemplo, un oso "*es un*" tipo de mamífero, una casa "*es un*" tipo de inmueble, y una rosa "*es un*" tipo particular de flor. Así, la herencia implica una jerarquía de generalización/especialización, en la que una subclase especializa el comportamiento o estructura, más general, de sus superclases.

Herencia en un Ambiente de Programación Orientado a Objetos Subjetivo

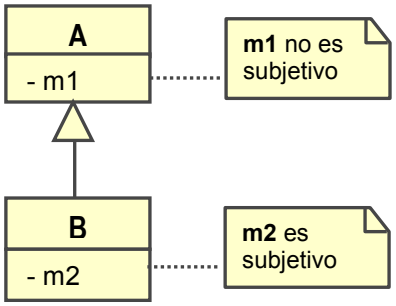
Es importante resaltar que, en un ambiente de programación orientado a objetos subjetivo o no, la herencia de estructura entre clases no se hace variable, debido a que el comportamiento de un objeto es el aspecto que se decidió que puede hacerse subjetivo, y no la información que contiene el mismo. Por lo cual, no se harán consideraciones especiales sobre la herencia en este aspecto. Simplemente se analizará la herencia de comportamiento.

[3] En esta tesis, se desarrollará el comportamiento subjetivo en un ambiente basado en clases.

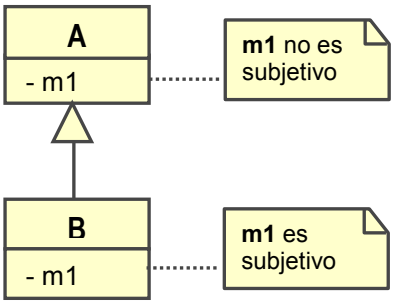
A continuación, se mostrará cómo se extiende mediante la herencia el comportamiento subjetivo y no subjetivo, considerando los siguientes casos en los que al menos está involucrada una clase subjetiva:

CASO 1: A es una clase no subjetiva y B es una clase subjetiva

CASO 1.a



CASO 1.b



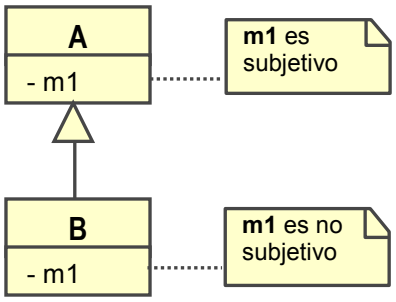
En ambos casos, la clase B es subjetiva, lo cual significa que sus instancias entienden al menos un mensaje subjetivo. Esto se debe a que:

- **CASO 1.a**: La clase B extiende el comportamiento de la clase A, definiendo un nuevo mensaje subjetivo.
- **CASO 1.b**: La clase B redefine un mensaje común (no subjetivo) definido en la clase A, de forma tal que lo hace subjetivo.

De esta manera, se expande o modifica el comportamiento definido en clases comunes (no subjetivas) con comportamiento subjetivo.

Por lo general, estos casos de subclasificación serán los más habituales, debido a que el comportamiento subjetivo se introducirá en un ambiente donde existirán clases no subjetivas.

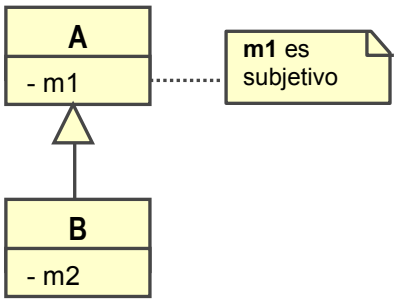
CASO 2: A es una clase subjetiva y B es una clase no subjetiva



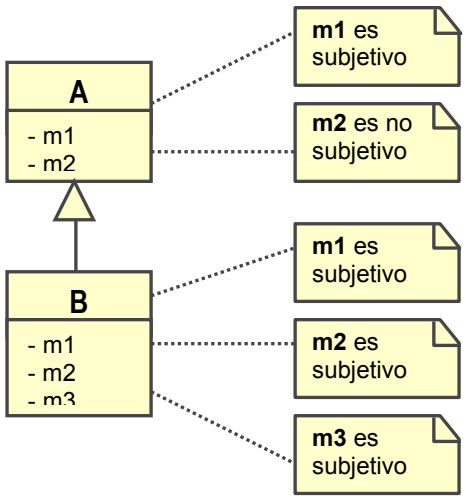
En este caso, podría parecer que la clase B es subjetiva, ya que está heredando el mensaje subjetivo **m1**. Sin embargo, debido a que dicha clase redefine este mensaje de forma no subjetiva (y asumiendo que hace lo mismo con todos los mensajes subjetivos definidos en sus superclases), pasa a ser una clase común.

CASO 3: A y B son ambas clases subjetivas

CASO 3.a



CASO 3.b



- **CASO 3.a**: La clase B hereda de la clase A el mensaje subjetivo **m1** y no lo redefine. De esta forma, es una clase subjetiva, ya que parte del comportamiento que define es subjetivo.
- **CASO 3.b**: En este caso, puede suceder que la clase B extienda el comportamiento definido en la clase A de alguna de las siguientes formas:
  - Redefiniendo mensajes subjetivos de la clase A de manera subjetiva.
  - Redefiniendo mensajes no subjetivos de la clase A de manera subjetiva.
  - Definiendo nuevos mensajes subjetivos.

Todos los casos anteriores pueden ser decisiones de diseño totalmente razonables.

Polimorfismo

Polimorfismo en un Ambiente de Programación Orientado a Objetos sin Comportamiento Subjetivo

En los ambientes orientados a objetos, dos objetos que responden a un mismo mensaje de manera polimórfica pueden ser intercambiados en forma transparente por el emisor del mensaje; esto se debe en gran medida a que las semánticas de dichos mensajes son equivalentes, aunque sus implementaciones sean distintas.

En forma trivial, un objeto es polimórfico consigo mismo si se considera a la misma implementación de un mensaje.

Polimorfismo en un Ambiente de Programación Orientado a Objetos Subjetivo

En un ambiente de programación orientado a objetos subjetivo es posible que un objeto sea polimórfico consigo mismo de manera no trivial, debido a que, si es un objeto subjetivo, es capaz de variar la forma de responder ante un mismo mensaje. Esto se debe a que un mensaje subjetivo se implementa con distintos métodos subjetivos polimórficos entre sí. El objeto subjetivo puede responder a un mismo mensaje de diferentes formas,

semánticamente equivalentes entre sí, pero determinadas por diferentes métodos polimórficos. Por lo tanto, el polimorfismo se extiende en un ambiente de este tipo, permitiendo que un objeto sea polimórfico consigo mismo de manera no trivial.

Encapsulamiento

Encapsulamiento en un Ambiente de Programación Orientado a Objetos sin Comportamiento Subjetivo

En la orientación a objetos, es muy frecuente referirse a un objeto como una caja negra, debido a que no es necesario conocer como está implementado para que interactúe con los demás objetos. Para esto, la clase de un objeto se define en dos partes:

- Un protocolo utilizado por los objetos en una aplicación para interactuar con las instancias de dicha clase,
- La implementación de los objetos de dicha clase (métodos y estructura interna).

Es por medio del encapsulamiento que podemos definir la estructura interna y los métodos de una clase de manera tal que sus instancias trabajen como unidades independientes de los demás objetos con los que interactúan. En otras palabras, con el encapsulamiento se gana modularidad, y se protege a los objetos de ser manipulados de forma inadecuada por objetos externos. Además, permite a un objeto ocultar información al mundo exterior, o bien restringir el acceso a la misma.

Un beneficio del encapsulamiento es que un objeto puede cambiar su implementación sin modificar su protocolo, con lo cual no afecta al resto de los objetos. Es útil para aumentar el nivel de abstracción; como consecuencia se obtiene un código modificable y extensible.

Encapsulamiento en un Ambiente de Programación Orientado a Objetos Subjetivo

La incorporación de comportamiento subjetivo no debe violar el concepto de encapsulamiento. Por lo tanto, un objeto subjetivo debe encapsular los diferentes métodos asociados a cada uno de sus mensajes subjetivos, como así también el mecanismo que determina cuál de estos métodos se ejecutará cuando se invoquen dichos mensajes.

Por otra parte, el comportamiento subjetivo de un objeto debe ser transparente para los objetos que interactúan con él, de modo que su interfaz externa no debe revelar su naturaleza. Esto significa que el emisor de un mensaje subjetivo nunca sabrá qué tipo de mensaje está enviando al receptor, ya que éste último define un único protocolo, sin hacer distinción entre mensajes subjetivos y no subjetivos.

De esta forma, el comportamiento subjetivo no viola los conceptos mencionados previamente, debido a que respeta el hecho de que el comportamiento de un objeto es transparente para los clientes del mismo, y desconocen la forma en que responde a un servicio solicitado.

En la siguiente sección, se comenzará con el desarrollo propiamente dicho del comportamiento subjetivo; se expondrán los conceptos que define, su organización estructural y la mecánica de evaluación del mismo.

ENTORNO DEL COMPORTAMIENTO SUBJETIVO

Cuando un objeto recibe un mensaje subjetivo se desarrollan una serie de acontecimientos, transparentes al mismo, en los cuales implícitamente se analiza aspectos internos y externos al objeto y, basándose en éstos, se decide cual es la respuesta adecuada

que el objeto debe brindar. En la mayoría de los casos, un aspecto es alguna característica de cierto objeto relacionado con el objeto subjetivo, en el sentido que cumple un determinado rol en relación con él. En conjunto, dichos objetos forman el *entorno del comportamiento subjetivo* (ECS), y son los únicos que pueden ser considerados para la determinación de la respuesta subjetiva.

Definición

El *ECS* es un conjunto de objetos donde cada uno cumple un rol específico con relación a un objeto subjetivo, y el cual podría afectar su comportamiento por poseer una determinada cualidad.

Caracteristicas de los objetos del ECS

Como se mencionó, ciertas particularidades de algunos objetos pueden afectar el comportamiento de un objeto subjetivo. Las cualidades o características de un objeto que se consideran relevantes en el desarrollo del comportamiento subjetivo son:

- Su *identidad*
- El *protocolo* que entiende
- Una *condición* que cumple

A continuación, se describen cada una de ellas.

Identidad de un objeto

La identidad es la propiedad de un objeto que hace posible su distinción frente a los demás objetos. Cada objeto tiene una única identidad, independientemente de su estado interno. De esta forma, el estado de un objeto puede cambiar durante su tiempo de vida, pero su identidad será la misma desde su creación hasta su destrucción. En particular, dos objetos de la misma clase, que tengan las mismas propiedades con los mismos valores, tendrán distintas identidades.

Protocolo que entiende un objeto

El comportamiento de un objeto puede considerarse como la suma de los servicios que presta a ciertos objetos, y los servicios que requiere de otros objetos. Desde este punto de vista, surge lo que *Meyer* llama "*modelo contractual de programación",* el cual establece que "la vista exterior[4] de cada objeto define un contrato del que pueden depender otros objetos, y que a su vez, debe ser llevado a cabo por la vista interior[5] del propio objeto (a menudo en colaboración con otros objetos)".

De esta manera, cada objeto cumple con un contrato que establece todas las suposiciones que pueden hacer otros objetos acerca de su comportamiento, y abarca todas sus responsabilidades (es decir, el comportamiento del que se considera responsable). Cada mensaje que forma dicho contrato se encuentra implementado por un método en la clase del objeto, y especificado mediante una firma. Al conjunto completo de mensajes que el objeto puede responder, junto con las formas de invocación de los mismos, se le denomina *protocolo*.

[4] La *vista exterior* de un objeto se refiere a su protocolo, lo que pueden "ver" el resto de los objetos para comunicarse con dicho objeto.

[5] La *vista interior* de un objeto se refiere a su estructura interna y sus detalles de implementación, lo que no deberían "ver" el resto de los objetos.

Según *Booch*, «el protocolo de un objeto denota las formas en las que puede actuar y reaccionar, y de esta manera, constituye la vista externa completa, estática y dinámica, de la abstracción que representa dicho objeto».

Condición que cumple un objeto

Como define *Booch*, «el estado de un objeto se forma por los resultados acumulados de su comportamiento, y refleja las posibles condiciones en las que puede existir». Dichas condiciones se establecen por determinadas características de sus colaboradores internos, como la identidad o el estado lógico de ellos.

Un objeto puede comprobar si cierta condición se establece en otro objeto, accediendo él mismo a uno o a varios de los objetos conocidos internamente, y verificando si se cumple o no un estado lógico en particular en cada uno de éstos o si poseen una determinada identidad. Pero de esta manera, no se respetaría el encapsulamiento del objeto. Una mejor forma de comprobar una condición en un objeto se logra agregando en su protocolo un mensaje que determine si se cumple o no dicha condición. Así, el propio objeto es el responsable de verificar si una condición se verifica o no en él.

LAS FUERZAS

Como se mencionó en un principio, la habilidad de un objeto de comportarse subjetivamente le otorga la capacidad de poder satisfacer un mismo requerimiento de diversas maneras, dependiendo de factores internos y/o externos que lo estén influyendo en ese momento.

Por su parte, para incluir subjetividad en la evaluación de un mensaje, es necesario establecer previamente, en la definición del mismo, cuáles son los factores que influirán, y a qué respuesta subjetiva conducirán.

Un factor se define como:

- Una *característica del objeto*[6] que se pretende que posea alguno de los objetos del ECS[7] cuando se evalúe el mensaje subjetivo (factor externo), o bien
- Una determinada condición que debe establecerse en los objetos que conoce internamente el objeto subjetivo cuando se evalúe dicho mensaje (factor interno).

A cada uno de estos factores se le atribuye una especie de "fuerza", ya que en ciertos casos, tiene el poder de variar el comportamiento del objeto subjetivo.

Definición de Fuerza

Una fuerza es un factor de influencia que tiene el poder de participar en la elección de una respuesta subjetiva, cuando un objeto subjetivo recibe un mensaje subjetivo.

Ahora bien, el ECS se forma con aquellos objetos que cumplen determinados roles en relación con el objeto que recibe el mensaje subjetivo. Dichos objetos son:

- El emisor del mensaje,
- Los objetos del contexto,
- (si existen) los colaboradores internos del objeto subjetivo,
- (si existen) los colaboradores externos del objeto subjetivo.

[6] Las características definidas como *Identidad, Protocolo* y *Condición.*
[7] ECS: Entorno del Comportamiento Subjetivo

Cada factor externo de influencia estará asociado a uno de los objetos del ECS, y dependiendo de cual sea su rol frente al objeto subjetivo, quedará representado por la *fuerza del emisor*, la *fuerza del contexto*, la *fuerza del colaborador interno* o la *fuerza del colaborador externo*; el único factor interno quedará representado por la *fuerza del estado*.

Definición de Predominio de Fuerza

En la evaluación de un mensaje subjetivo, una o más fuerzas pueden predominar. Se dice que una fuerza:

- Externa (emisor, colaborador interno, colaborador externo o contexto) predomina cuando la *característica del objeto* pretendida para su objeto asociado dentro del ECS (establecida en la definición del mensaje), coincide con la característica que tiene dicho objeto en el momento de la evaluación del mensaje;
- Interna (estado*)* predomina cuando la condición pretendida en el objeto subjetivo (establecida en la definición del mensaje), se concreta en el momento de la evaluación del mensaje.

Por lo tanto, cuando un objeto recibe un mensaje subjetivo, se analizan las fuerzas que fueron configuradas como influyentes en el comportamiento del mismo. Si alguna de estas fuerzas predomina, conducirá en algún sentido específico el camino para la elección de la respuesta correcta.

Seguidamente, se definen cada una de las fuerzas del comportamiento subjetivo.

LA FUERZA DEL COLABORADOR

Características del Colaborador que pueden influir en el comportamiento de un objeto

Los colaboradores que se tendrán en cuenta en el desarrollo del comportamiento subjetivo serán:

- *Colaboradores Internos*: aquellos colaboradores relacionados con el objeto subjetivo mediante el conocimiento interno.
- *Colaboradores Externos*: aquellos colaboradores relacionados con el objeto subjetivo mediante el conocimiento temporal externo.

Un colaborador (interno o externo) puede afectar el comportamiento de un objeto cuando recibe un mensaje, por alguna de sus siguientes características:

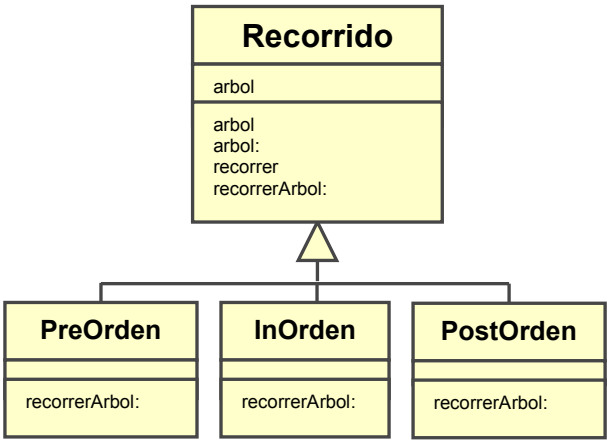
- Su identidad,
- Su protocolo,
- Cierta condición que cumple.

Ejemplos

Para apreciar mejor la influencia de un colaborador en el comportamiento de un objeto, se presentarán algunos ejemplos, usando la notación UML y código Smalltalk.

Ejemplo 1 (Fuerza del Colaborador Interno)

Considere un diseño en objetos con el siguiente modelo asociado:



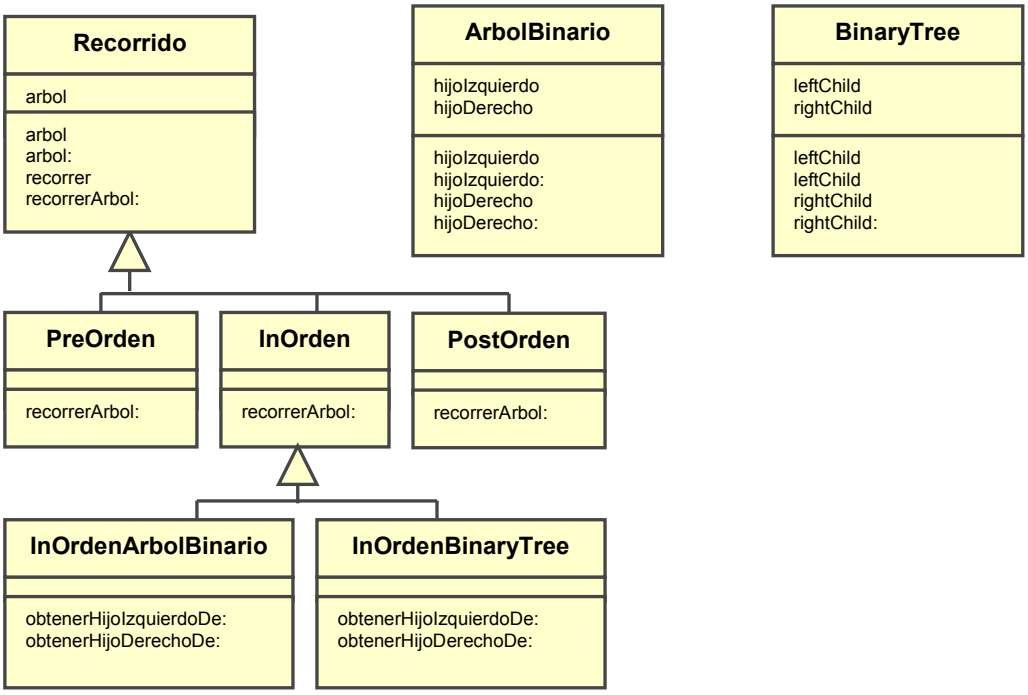
Las instancias de las subclases de *Recorrido* son responsables de recorrer estructuras de árbol binario: las instancias de *PreOrden* recorren árboles en forma "*pre-orden*", las instancias de *InOrden* los recorren en forma "*in-orden*" y las instancias de *PostOrden* en forma "*post-orden*". Para que cada uno de dichos objetos pueda realizar correctamente su recorrido, necesita pedirle al objeto árbol que esté recorriendo sus hijos. De esta manera, los recorridos deben conocer el protocolo que entienden los objetos árbol para obtener sus hijos. Considere la siguiente implementación del método *recorrer* en la clase *Recorrido* y el método *recorrerArbol:unArbol* en la clase *InOrden:*

```
Recorrido>>recorrer

      self recorrerArbol: self arbol.


InOrden>>recorrerArbol: unArbol

      "Se recorre el árbol en forma in-orden"

      self recorrerArbol: (unArbol hijoIzquierdo).
      ......
      "Se procesa la información de unArbol"
            ......
      self recorrerArbol: (unArbol hijoDerecho).
```

El inconveniente que se presenta es que los recorridos deben conocer el protocolo que entienden los árboles, atentando este hecho con la reusabilidad de dichos recorridos. Estos objetos no pueden ser reutilizados para recorrer cualquier tipo de árbol, ya que en la implementación del método que realiza el recorrido necesitan acceder de una manera establecida a los hijos de un árbol y no es cierto que todas las clases que representan árboles binarios respetan el mismo protocolo para el acceso a sus hijos.

Una posible solución, es subclasificar las clases *PreOrden, InOrden* y *PostOrden* por cada clase de árbol binario cuyas instancias se desean recorrer. De este modo, cada una de dichas subclases utiliza en la implementación de sus recorridos, el protocolo del árbol a

recorrer. Por ejemplo, si existen dos clases de árboles binarios: *ArbolBinario* y *BinaryTree*, podría subclasificarse las subclases de *Recorrido* de la siguiente manera:



Así, la implementación del método *recorrerArbol: unArbol* de la clase *InOrden* podría redefinirse de la siguiente manera:

```
InOrden>>recorrerArbol: unArbol

      self recorrerArbol: (self obtenerHijoIzquierdoDe:unArbol).
      ......
      "Se procesa la información de unArbol"
          ......
      self recorrerArbol: (self obtenerHijoDerechoDe:unArbol).
```

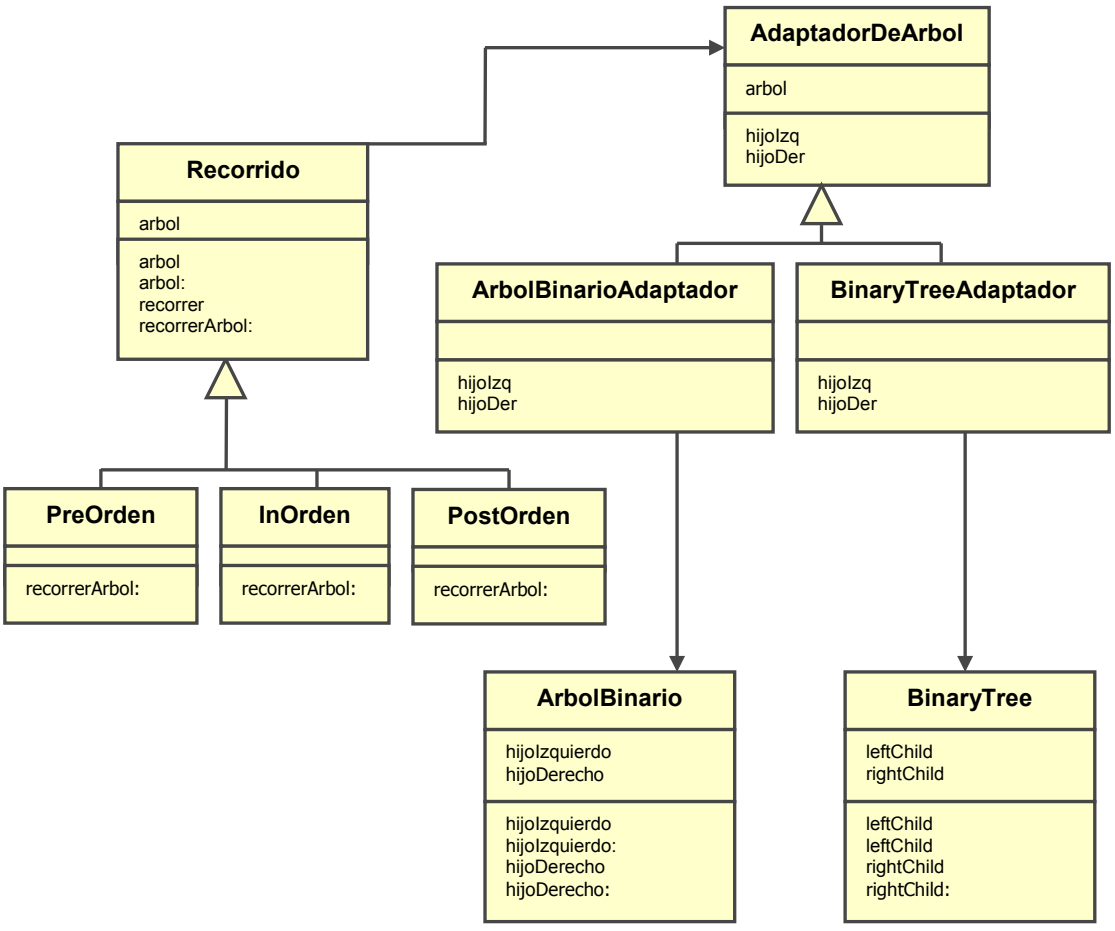
Cada subclase de *InOrden* debe redefinir los métodos *obtenerHijoIzquierdoDe:unArbol* y *obtenerHijoDerechoDe:unArbol* de manera tal que en cada uno ser recupere adecuadamente el hijo izquierdo y el hijo derecho de *unArbol*. En el ejemplo, las clases *InOrdenArbolBinario* y *InOrdenBinaryTree* definirían los métodos anteriores de la siguiente manera:

```
InOrdenArbolBinario>> obtenerHijoIzquierdoDe:unArbol

      ^unArbol hijoIzquierdo


InOrdenArbolBinario>> obtenerHijoDerechoDe:unArbol

      ^unArbol hijoDerecho
```

```
InOrdenBinaryTree>> obtenerHijoIzquierdoDe:unArbol

      ^unArbol leftChild.


InOrdenBinaryTree>> obtenerHijoDerechoDe:unArbol

      ^unArbol rightChild.
```

Esta es una forma de resolver el problema de los protocolos desiguales entre las diferentes clases de árboles. Pero tiene la desventaja de que es necesario subclasificar cada clase de recorrido por cada clase de árbol nueva a recorrer que define diferente protocolo de acceso a sus hijos.

Otra posible solución, es abstraer la adaptación de los protocolos desiguales de acceso a los hijos en una jerarquía de clases. Esto significa crear una nueva jerarquía de clases cuyo principal propósito sea adaptar el protocolo de cada clase de árbol a uno fijo que utilizan los recorridos. La jerarquía de los recorridos quedaría conformada de la misma forma que en un principio, es decir, la superclase sería *Recorrido* y sus subclases serían *PreOrden, InOrden* y *PostOrden*. En la nueva jerarquía, existe por cada clase de árbol una clase que "adapta" el protocolo que utilizan los recorridos para acceder a los hijos al protocolo que defina dicha clase de árbol. De esta forma, el modelo de clases sería el siguiente:



Las clases de los recorridos accederán a los hijos de los árboles siempre de la misma manera:

```
InOrden>>recorrerArbol: unArbol

      self recorrerArbol: unArbol hijoIzq.
      ......
      "Se procesa la información de unArbol"
            ......
      self recorrerArbol: unArbol hijoDer.
```

De la misma manera accederán las clases *PreOrden* y *PostOrden.* Cabe destacar que el objeto que recibe como parámetro el método *recorrerArbol:unArbol* no es un árbol, se trata de una instancia de la clase *ArbolBinarioAdaptador* o una instancia de la clase *BinaryTreeAdaptador* las cuales conocen a un árbol, y se encargan de pedirle adecuadamente sus hijos. Así, dichas clase de adaptadores redefine los mensajes de acceso a los hijos de un árbol de la siguiente manera:

```
ArbolBinarioAdaptador>>hijoIzq

      ^self arbol hijoIzquierdo
```

```
ArbolBinarioAdaptador>>hijoDer

      ^self arbol hijoDerecho
```

```
BinaryTreeAdaptador>>hijoIzq

      ^self arbol leftChild
```

```
BinaryTreeAdaptador>>hijoDer

      ^self arbol rightChild
```

Esta solución parece más apropiada que la anterior, ya que no se requiere subclasificar cada clase de recorrido por cada nueva clase de árbol a recorrer; sólo se necesita definir una clase adaptadora que acceda adecuadamente a los hijos de los árboles de la nueva clase.

Sin embargo, podría evitarse definir la jerarquía de adaptadores empleando comportamiento subjetivo. Podría utilizarse la *fuerza del colaborador interno* en el mensaje *recorrerArbol:unArbol* de las subclases de *Recorrido* de manera tal que:

- Si el **colaborador interno** que conoce el recorrido (un objeto árbol) **entiende el protocolo** definido por la clase *ArbolBinario*, entonces el mensaje se define de la siguiente manera:

```
InOrden>>recorrerArbol: unArbol

      self recorrerArbol: unArbol hijoIzquierdo.
      ......
      "Se procesa la información de unArbol"
            ......
      self recorrerArbol: unArbol hijoDerecho.
```

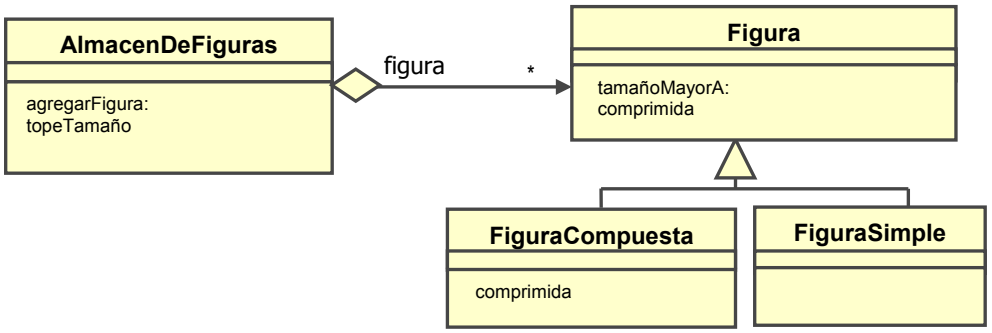
- Si el **colaborador interno** que conoce el recorrido (objeto árbol) **entiende el protocolo** definido por la clase *BinaryTree*, el mensaje se define de la siguiente forma:

```
InOrden>>recorrerArbol: unArbol

      self recorrerArbol: unArbol leafchild.
      ......
      "Se procesa la información de unArbol"
            ......
      self recorrerArbol: unArbol rightChild.
```

Con lo cual, queda resuelto el problema sin la necesidad de definir una jerarquía de clases para adaptar protocolos.

Ejemplo 2 (la Fuerza del Colaborador Externo)

Considere un diseño en objetos con el siguiente modelo asociado:



Las instancias de *AlmacenDeFiguras* son responsables de almacenar figuras. Para esto, mantiene dos colecciones: una con figuras simples y otra con figuras compuestas. En determinados casos, una figura que sea instancia de *FiguraCompuesta* puede tener un gran tamaño y, por requerimientos del sistema, es conveniente comprimirla y luego almacenarla, cuando su tamaño supera un valor *X*.

De esta manera, el almacén de figuras debe comprimir a aquellas figuras que superen el valor *X* antes de almacenarlas. Así, cada figura debe ser capaz de poder determinar si su tamaño es mayor o igual a un cierto valor.

Una solución trivial implica que, cada vez que al almacén de figuras se le da una figura compuesta para almacenar, compruebe su tamaño, y basándose en él decida si comprimir o no la figura. Una implementación para el mensaje *agregarFigura:unaFigura* podría ser:

```
AlmacenDeFiguras>>agregarFigura: unaFigura
      (unaFigura class = FiguraSimple)
      ifTrue:[self agregarFiguraSimple:unaFigura]
      ifFalse:[(unaFigura tamañoMayorA: self topeTamaño)
            ifFalse:[self agregarFiguraCompuesta:unaFigura]
            ifTrue:[
                  self agregarFiguraCompuesta:(unaFigura comprimida)]]
```

De esta manera se resuelve el problema de comprimir las figuras, pero el código puede empeorar en el caso de que se agreguen más restricciones.

Otra solución podría ser que se delegue a la propia figura la responsabilidad de agregarse en el almacén de figuras. Así, cada figura sería responsable de comprimirse en el caso de que su tamaño supere el valor de *X.* La clase *Figura* definiría el mensaje *agregarseEnAlmacenDeFigura:unalmagen*, y cada subclase lo redefiniría de la siguiente manera:

```
FiguraCompuesta>>agregarseEnAlmacenDeFigura: unalmagen

      (self tamañoMayorA: unalmagen topeTamaño)
      ifTrue:[unalmagen agregarFiguraCompuesta: (self comprimida)]
      ifFalse:[unalmagen agregarFiguraCompuesta: self].


FiguraSimple>>agregarseEnAlmacenDeFigura: unalmagen

          unalmagen agregarFiguraSimple: self.
```

Dicha solución tiene la ventaja de que el almacén de figuras no tiene que analizar todos los posibles tipos de figuras para agregarlas, con el precio de que se introduce en las clases de las figuras la responsabilidad de agregarse adecuadamente en un almacén de figuras, aumentando el acoplo entre las clases de dichos objetos.

Una solución con comportamiento subjetivo podría establecerse usando la *fuerza del colaborador externo* en el mensaje *agregarFigura: unaFigura* de la clase *AlmacenDeFiguras*, de la siguiente manera:

- Cuando el **colaborador externo** (un objeto figura) **entiende el protocolo** que define la clase *FiguraSimple*:

  ```
  self agregarFiguraSimple: unaFigura.
  ```

- Cuando el **colaborador externo** (un objeto figura) **cumple con la condición** "tamaño menor que *X*":

  ```
  self agregarFiguraCompuesta: unaFigura.
  ```

- Cuando el **colaborador externo** (un objeto figura) **cumple con la condición** "tamaño mayor que *X*":

  ```
  self agregarFiguraCompuesta: (unaFigura comprimida).
  ```

- En cualquier otro caso, no se almacena.

De esta forma, no es necesario verificar explícitamente la clase del colaborador externo (la figura) que llega como parámetro en el mensaje, ni el tamaño del mismo.

Entonces... ¿Cual es la Fuerza del Colaborador?

La *fuerza del colaborador* es uno de los factores externos de influencia. Se basa en uno de los colaboradores (interno o externo) del objeto subjetivo, y tiene en cuenta una de las *características del objeto* de dicho colaborador. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL ESTADO

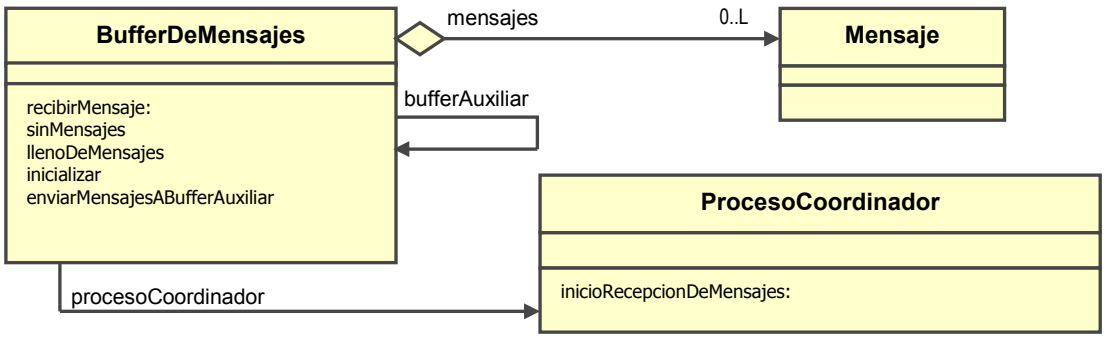
Características del Estado que pueden influir en el comportamiento de un objeto

El conjunto de colaboradores internos que conforman el estado de un objeto varía durante el transcurso de su vida. Es posible que en algún instante, se establezcan ciertas características en uno o varias de estos colaboradores que simbolice un determinado estado lógico para dicho objeto. La presencia de un estado lógico específico puede tener un efecto importante en el comportamiento del objeto, influyendo en la elección de la respuesta a brindar ante un mensaje.

Ejemplo

Se presentará un ejemplo que demuestra como un determinado estado lógico puede tener influencia en el comportamiento de un objeto, usando la notación UML y código *Smalltalk*.

Considere un diseño de objetos con el siguiente modelo asociado:



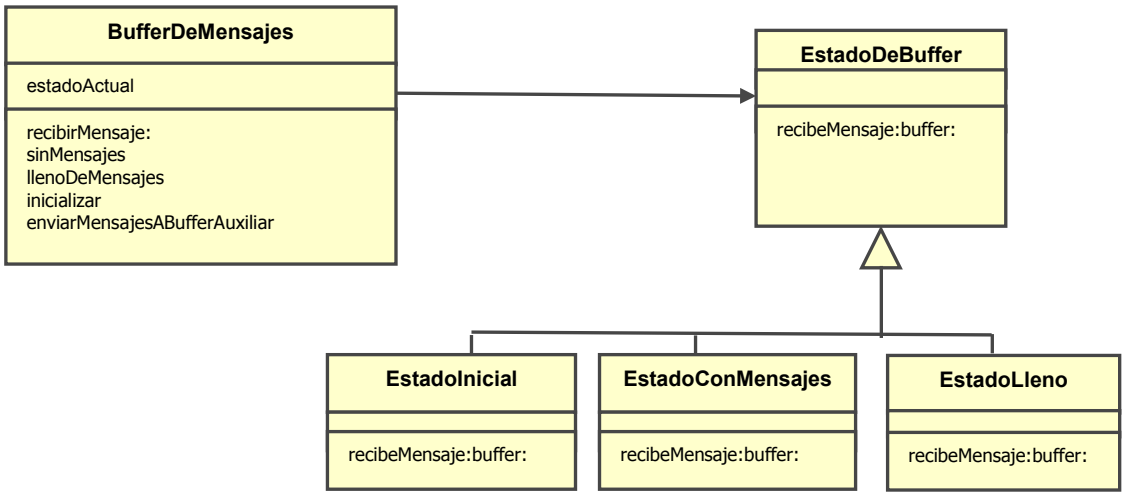
Un objeto de la clase *BufferDeMensajes* debe almacenar mensajes (instancias de la clase *Mensaje*) que recibe de otros objetos. Como requisito adicional, debe notificar a un proceso coordinador cuando recibe el primer mensaje estando vacío (es decir, sin mensajes recibidos aún), y enviar todos sus mensajes a otro buffer auxiliar cuando la cantidad que contiene de los mismos supere su límite permitido.

En la clase *BufferDeMensajes* podría definirse el método *recibirMensaje:unMensaje* de la siguiente manera:

```
BufferDeMensajes>>recibirMensaje: unMensaje
      (self sinMensajes)
      ifTrue:[
            self notificarAProcesoCoordinador].
      (self llenoDeMensajes)
      ifTrue:[
            self enviarMensajesABufferAuxiliar].
      self agregarMensaje: unMensaje.


BufferDeMensajes>> notificarAProcesoCoordinador

      self procesoCoordinador inicioRecepcionDeMensajes: self
```

Esta solución resuelve el problema, pero como puede verse, se oscurece el código con las comprobaciones que debe realizar el buffer en el mensaje *recibirMensaje: unMensaje*.

Otra solución, podría lograrse creando una jerarquía de estados de buffer, los cuales sean responsables de mantener correctamente la lista de mensajes en el buffer, cumpliendo con los requisitos impuestos para el mismo.



De esta manera, el objeto buffer siempre conoce a un objeto instancia de alguna subclase de *EstadoDeBuffer*, el cual representa el estado actual del mismo. Cada vez que el objeto buffer recibe el mensaje *recibirMensaje:unMensaje*, se lo delega a su actual objeto estado, de la siguiente manera:

```
BufferDeMensajes>> recibirMensaje: unMensaje

      self estadoActual recibeMensaje: unMensaje buffer: self.
```

Los objetos estados son responsables de agregar correctamente el mensaje en el buffer, realizar algún tipo de operación extra si le fuera pertinente, y cambiar adecuadamente el estado del buffer si fuera necesario.

Inicialmente, el buffer conoce a un objeto instancia de *EstadoInicial.* Cuando el buffer delegue el primer mensaje a su objeto estado, el mismo se encargará de notificar al proceso coordinador este hecho y agregará el mensaje a la lista de mensajes de la siguiente manera:

```
EstadoInicial>> recibeMensaje: unMensaje buffer: unBuffer

      unBuffer notificarAProcesoCoordinador.
      unBuffer estado:(estadoConMensajes new).
      unBuffer recibirMensaje: unMensaje.
```

Es importante que se produzca en este caso un cambio de estado en el buffer, ya que ahora el buffer posee al menos un mensaje, y el hecho de recibir nuevos mensajes en este estado implica otra forma de manejarse para el buffer.

Cuando el buffer se encuentra en el estado "con mensajes", deben agregarse los nuevos mensajes que lleguen mientras que no se llene. Para esto, la clase *estadoConMensajes* define el mensaje *recibeMensaje:unMensaje buffer:unBuffer* de la siguiente manera:

```
EstadoConMensajes >> recibeMensaje: unMensaje buffer: unBuffer

      (unBuffer llenoDeMensajes)
      ifFalse:[
            ^unBuffer agregarMensaje: unMensaje].
      unBuffer estado:(EstadoLleno new).
      unBuffer recibirMensaje:unMensaje.
```

Como puede verse, en el caso de que el buffer esté lleno de mensajes, se debe producir un cambio de estado en el mismo, debido a que no se puede agregar como antes el mensaje recibido en el buffer.

El nuevo estado al que pasa el buffer es el estado "lleno". Al recibir un mensaje en este estado, es necesario volcar los mensajes anteriormente almacenados en el buffer auxiliar y almacenar el nuevo mensaje. Para esto, la clase *EstadoLleno* define el mensaje *recibeMensaje: unMensaje buffer: unBuffer* de la siguiente manera:

```
EstadoLleno>> recibeMensaje: unMensaje buffer: unBuffer

      unBuffer enviarMensajesABufferAuxiliar.
      unBuffer estado:(estadoincial new).
      unBuffer recibirMensaje: unMensaje
```

De esta forma, una vez que el buffer volcó sus mensajes en el buffer auxiliar, pasa nuevamente al estado inicial.

Esta solución es más elegante que la anterior, debido a que quedan bien diferenciados los distintos estados en los que pueden encontrarse el buffer y las acciones que deben realizarse en cada uno de ellos. Sin embargo, tiene el costo adicional de definir una jerarquía de estados para lograr dichos beneficios.

La solución con comportamiento subjetivo en este problema podría lograrse utilizando la *fuerza del estado* en el mensaje *recibirMensaje:unMensaje*, configurándolo de la siguiente forma:

- Cuando el buffer **cumple con al condición** "sin mensajes":

```
self procesoCoordinador inicioRecepcionDeMensajes:self.
self agregarMensaje:unMensaje.
```

- Cuando el buffer **cumple con al condición** "lleno de mensajes":

```
self enviarMensajesABufferAuxiliar.
self agregarMensaje: unMensaje.
```

- En cualquier otro caso:

```
self agregarMensaje: unMensaje.
```

Con comportamiento subjetivo, el buffer no necesita verificar explícitamente su estado para manejar los requisitos impuestos. En la configuración del mensaje subjetivo se establece en cada caso posible de estado las operaciones a realizar.

Entonces... ¿Cual es la Fuerza del Estado?

La *fuerza del estado* es el factor interno de influencia que se basa en uno de los posibles estados lógicos que puede encontrarse el objeto subjetivo. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si se establece o no algún determinado estado lógico; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL EMISOR

Características del Emisor que pueden influir en el comportamiento de un objeto

Un emisor de un mensaje puede afectar el comportamiento del respectivo receptor, por alguna de sus siguientes características:

- Su identidad,
- Su protocolo,
- Cierta condición que cumple.

Ejemplo

En el siguiente ejemplo se muestra como puede influir el emisor de un mensaje en la respuesta a brindar por el receptor del mismo. Se utiliza la notación UML y código *Smalltalk.*

Considere un diseño en objetos con el siguiente modelo asociado:

Un objeto *LlaveDeAcceso* conserva un código que permite el acceso a ciertos datos. Dicho código no puede ser entregado a cualquier tipo de objeto, sólo las instancias de la clase *AdministradorDeDatosX* están autorizadas a recibirlo.

Sería difícil para el objeto llave de acceso poder determinar cuando el objeto que requiere su código está autorizado a obtenerlo.

Una solución se lograría definiendo en la clase *LlaveDeAcceso* el siguiente mensaje:

```
LlaveDeAcceso>>darCodigoA: unObjeto
      (unObjeto sosAdministradorDeDatosX)
      ifTrue:[^self codigo]
      ifFalse:[^self error].
```

Las desventajas de esta solución son:

- El objeto llave de acceso necesita interactuar con el objeto que requiere su código para determinar si está autorizado a tenerlo.
- Es necesario que todos los objetos que pueden solicitar el código de acceso entiendan el mensaje *sosAdiministradorDeDatosX*, y sólo los objetos instancias de la clase *AdministradorDeDatosX* deben responder en forma "afirmativa" a dicho mensaje.

Podría evitarse definir el mensaje *sosAdiministradorDeDatosX* si se pregunta directamente por la clase del objeto que solicita el código de acceso, aunque esto no sea una buena práctica de programación. El mensaje *darCodigoA:unObjeto* quedaría definido de la siguiente forma:

```
LlaveDeAcceso>>darCodigoA: unObjeto
      (unObjeto class = AdministradorDeDatosX)
      ifTrue:[^self codigo]
      ifFalse:[^self error].
```

Las dos soluciones anteriores implican que el objeto llave de acceso debe conocer o interactuar con el objeto que requiere su código, introduciendo de esta manera un acoplo entre las clases de ambos objetos que no siempre resulta necesario.

Una solución usando comportamiento subjetivo podría obtenerse introduciendo la *fuerza del emisor*. Así, se evitaría definir el mensaje *sosAdministradorDeDatosX* o realizar cualquier tipo de comprobación del tipo del objeto que requiere el código de acceso. Además, no sería necesario que el objeto llave de acceso interactúe con el objeto que le solicita el código. Se podría definir subjetivamente el mensaje *codigo* en la clase *LlaveDeAcceso* para que:

- Cuando el objeto emisor del mensaje **entiende el protocolo** que define la clase *AdministradorDeDatosX*:

  ```
  ^self codigo.
  ```

- Cuando el objeto emisor del mensaje no **entiende el protocolo** que define la clase *AdministradorDeDatosX*:

  ```
  ^self error.
  ```

De esta forma, un objeto llave de acceso desconoce al emisor que solicita su código, evitando acoplo innecesario entre sus clases, y comprobar directa o indirectamente la clase del mismo.

Entonces... ¿Cuál es la Fuerza del Emisor?

La *fuerza del emisor* es otro de los factores externos de influencia. Se basa en el objeto emisor de un mensaje subjetivo, y tiene en cuenta una de las *características del objeto* de dicho emisor. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL CONTEXTO

Características de un objeto del Contexto que pueden influir en el comportamiento de un objeto

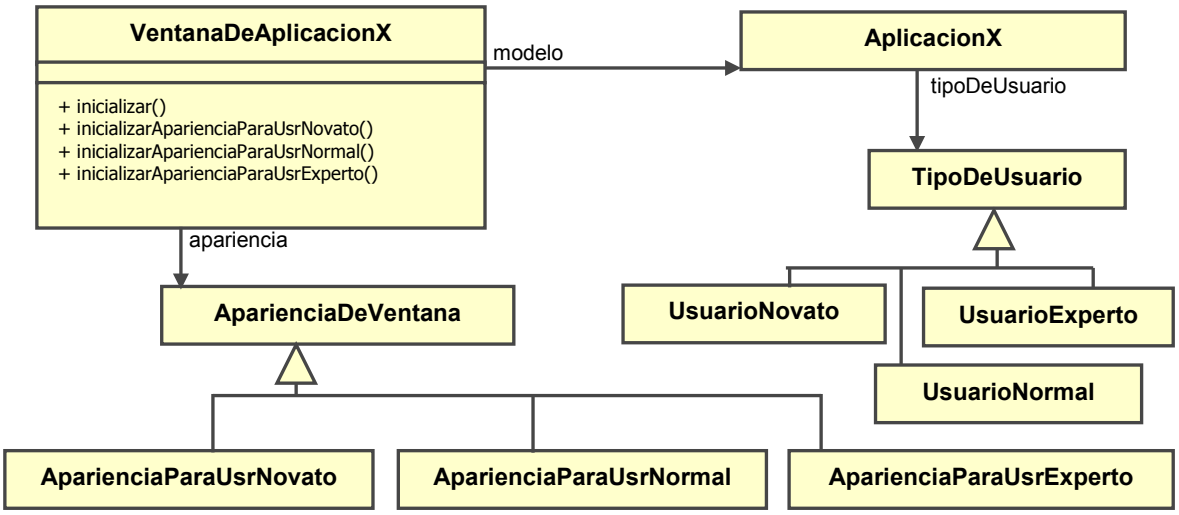
Un objeto que forma parte del contexto de otro objeto (es decir, que participa en la red de colaboración) puede influir en la respuesta a un mensaje enviado a dicho objeto, por alguna de sus siguientes características:

- Su identidad
- Su protocolo
- Cierta condición que cumple

Ejemplo

El siguiente es un ejemplo en donde un objeto del contexto de otro objeto influye en su comportamiento. Se usará la notación UML y código *Smalltalk*.

Considere un diseño en objetos con el siguiente modelo asociado:



Un objeto instancia de *VentanaDeAplicacionX* es una interfaz de usuario cuyo modelo es un objeto de la clase *AplicacionX.* Dicha ventana presenta diferente apariencia dependiendo del tipo de usuario que utiliza la aplicación. La misma, para mostrarse, deberá colaborar con:

- Un objeto de la clase *AparienciaParaUsrNovato*, si el usuario que la utiliza es uno novato,
- Un objeto de la clase *AparienciaParaUsrNormal*, si el usuario que la utiliza es uno normal, o
- Un objeto de la clase *AparienciaParaUsrExperto*, si el usuario que la utiliza es uno experto.

De esta forma, una ventana debería ser capaz de determinar el tipo de usuario para su modelo antes de mostrar su apariencia. Esto podría lograrse incluyendo en la inicialización de la ventana el siguiente código:

```
VentanaDeAplicacionX>>inicializar
      …
      …
      …
      TipoDeUsr:= self modelo tipoDeUsuario.
      TipoDeUsr sosUsuarioNovato
            ifTrue:[self inicializarAparienciaParaUsrNovato]
            ifFalse:[TipoDeUsr sosUsuarioNormal
                     ifTrue:[self inicializarAparienciaParaUsrNormal]
                     ifFalse:[
                        self inicializarAparienciaParaUsrExperto]].
      …
```

Las desventajas que trae esta solución son:

- La ventana debe acceder a un colaborador interno de su modelo (el objeto tipo de usuario).
- La misma debe "interrogar" al tipo de usuario para verificar de cuál se trata, implicando varias comprobaciones que deben necesariamente realizarse en el código.

Otra solución podría ser que el mismo modelo de la aplicación inicialice el tipo de interfaz de usuario a mostrar, basándose en su tipo de usuario. La clase *AplicacionX* definiría el mensaje *inicializarInterfazDeUsuario:unaInterfaz* de la siguiente forma:

```
AplicacionX>> inicializarInterfazDeUsuario:unaInterfaz

      (self tipoDeUsuario sosUsuarioNovato)
      ifTrue:[unaInterfaz inicializarAparienciaParaUsrNovato]
      ifFalse:[self tipoDeUsuario sosUsuarioNormal
               ifTrue:[unaInterfaz inicializarAparienciaParaUsrNormal]
               ifFalse:[
                 unaInterfaz inicializarAparienciaParaUsrExperto]].
```

Así, se logra que el objeto ventana no tenga que acceder a un colaborador interno de su modelo, pero no elimina la necesidad de "interrogar" al objeto tipo de usuario para saber de cuál se trata. Asimismo, es necesario que el modelo interactúe con el objeto ventana, lo cual quita reusabilidad a las clases de ambos objetos.

Una solución usando comportamiento subjetivo podría obtenerse introduciendo la *fuerza del contexto.* Se podría definir un mensaje subjetivo *inicializarApariencia* en la clase *VentanaDeAplicacionX* que brinde las siguientes respuestas:

- Cuando el tipo de usuario de su modelo (un objeto del contexto) **entiende el protocolo** que define la clase *UsuarioNovato*:

  ```
  self inicializarAparienciaParaUsrNovato
  ```

- Cuando el tipo de usuario de su modelo **entiende el protocolo** que define la clase *UsuarioNormal*:

  ```
  self inicializarAparienciaParaUsrNormal
  ```

❖ Cuando el tipo de usuario de su modelo **entiende el protocolo** que define la clase *UsuarioExperto*:

```
self inicializarAparienciaParaUsrExperto
```

El mensaje *inicializa* en la clase *VentanaDeAplicacionX* quedaría reducido a:

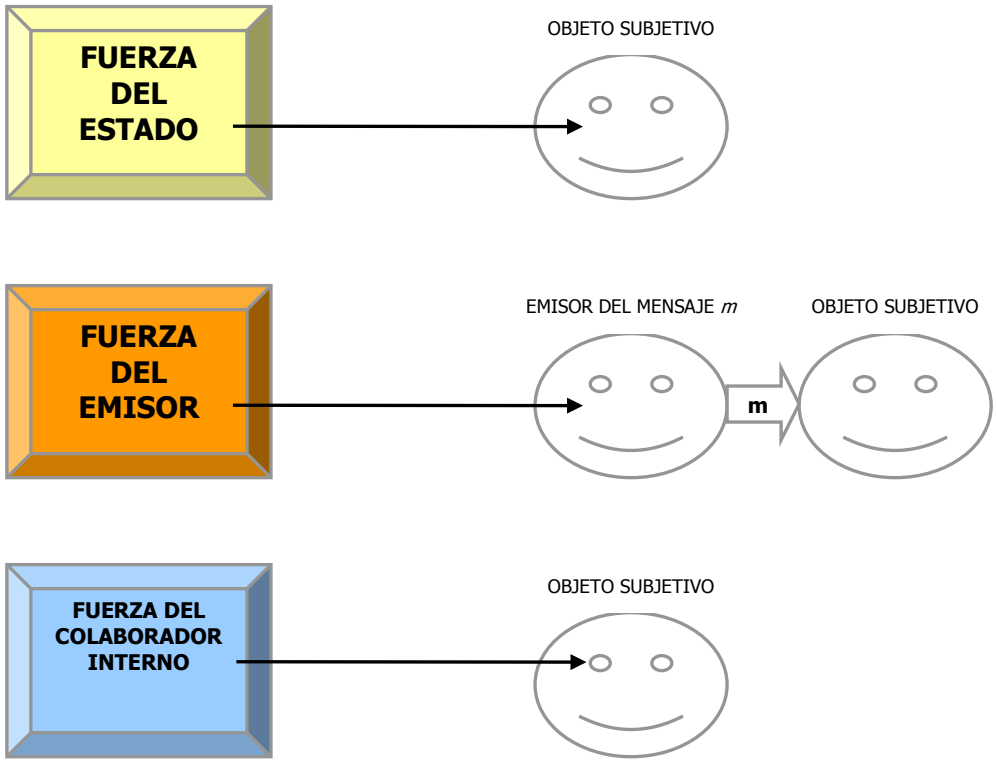
```
VentanaDeAplicacionX>>inicializar
        …
        self inicializarApariencia.
        …
```

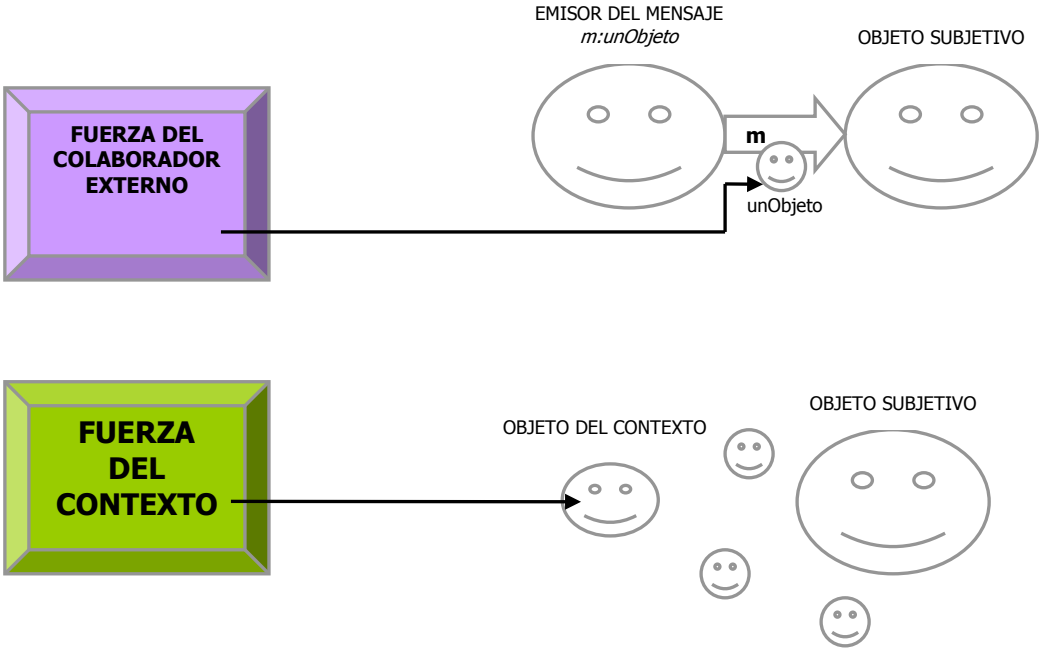
Por lo tanto, ni la ventana ni el modelo requieren verificar explícitamente cuál es el tipo de usuario de su modelo, para mostrar la apariencia correcta de la interfaz.

Entonces... ¿Cual es la Fuerza del Contexto?

La *fuerza del contexto* es un factor externo más de influencia. Se basa en uno de los objetos que forman el contexto de un objeto subjetivo, y tiene en cuenta una de sus *características del objeto*. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

El siguiente diagrama muestra en forma resumida todos los tipos de fuerzas que forman parte del comportamiento subjetivo y los objetos del ECS que cada una de estas tiene asociado.

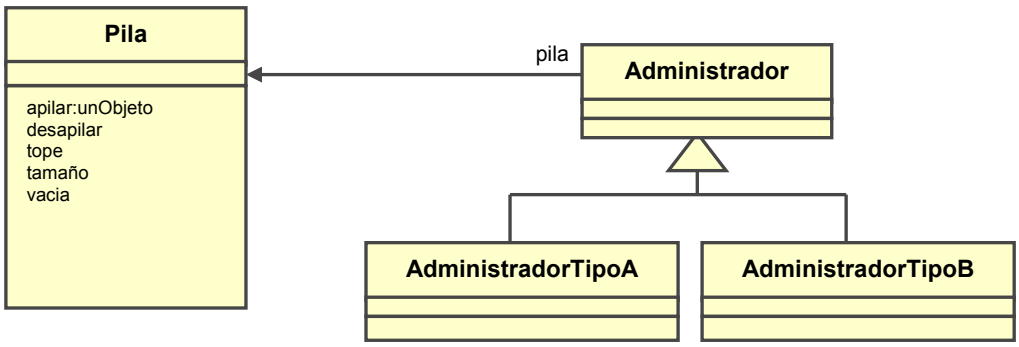




A continuación, se presenta el "esqueleto" y la "mecánica" de un mensaje subjetivo. En primer lugar, se brindará un ejemplo en el cual se expone implícitamente como se organiza estructuralmente un mensaje subjetivo; luego, se muestra cual es su organización estructural y la lógica de evaluación.

CASO DE ESTUDIO: UNA PILA SUBJETIVA

Considere el siguiente diagrama de clases como parte del modelo asociado al diseño en objetos de dicha aplicación:



Cada objeto administrador utiliza una pila para almacenar ciertos elementos. En particular, una instancia de la clase *AdministradorTipoA* puede demandar cualquier servicio de su pila (apilar y desapilar elementos, consultar el estado de la pila, etc.). Sin embargo, una instancia de la clase *AdministradorTipoB* puede interactuar con su pila para solicitarle cualquier servicio a excepción del desapilado de elementos.

Una instancia de la clase *Pila* posee el comportamiento esperado en una estructura de datos *LIFO (Last In – First Out).* Básicamente, un objeto pila es capaz de apilar y desapilar objetos, retornar el último objeto en apilarse, determinar la cantidad de objetos que apiló y comprobar si tiene algún objeto apilado.

En el caso del mensaje *desapilar*, hay que tener en cuenta que no cualquier cliente de la pila puede solicitarle a la misma la extracción de su último elemento apilado, por ejemplo, una instancia de la clase *AdministradorTipoB* tiene denegado este servicio. De ese modo, se deduce que una pila tendrá que ser capaz de comportarse de diferentes formas, dependiendo de quién le solicite el desapilado, es decir, de que tipo es el objeto emisor del mensaje *desapilar.* Podría definirse de manera subjetiva este mensaje utilizando la *fuerza del emisor*, de forma tal que:

- Si el emisor **entiende el protocolo** definido por la clase *AdministradorTipoB* → la pila no desapila su último elemento.
- Si el emisor no **entiende el protocolo** definido por la clase *AdministradorTipoB* → la pila desapila su último elemento.

No obstante, aunque es necesario que la pila varíe su comportamiento a causa de qué tipo es el cliente que requiere desapilar, la *fuerza del emisor* no es la única que puede tener influencia en el comportamiento asociado al desapilado. Una pila puede desapilar elementos siempre y cuando tenga al menos un elemento. Esto indica que el estado de la pila (vacía o no vacía) es relevante para llevar a cabo tal servicio. En consecuencia, la *fuerza del estado* juega un rol importante para la pila cuando se le solicita la extracción del último elemento, ya que:

- Si la pila **no está vacía** → desafila.
- Si la pila **está vacía** → notifica un error.

El problema que emerge es un conflicto entre fuerzas, y surge cuando dos o más fuerzas pueden tener influencia en la forma de resolver un mismo servicio. En el ejemplo, se observa que entran en conflicto la *fuerza del emisor* y la *fuerza del estado* en el requerimiento de desapilado.

Analizando un poco más en detalle, cuando la pila reciba el mensaje *desapilar,* tendrá "en la mano" dos formas posibles de comportarse, que dependerán del emisor y de su estado. Concretamente se tendrá lo siguiente:

ESTADO / EMISOR	Un administrador no de tipo B	Un administrador de tipo B
No vacía	❖ Desapila un elemento (por la *fuerza del emisor*), o bien ❖ Desapila un elemento (por la *fuerza del estado*)	❖ No desapila (por la *fuerza del emisor*), o bien ❖ Desapila un elemento (por la *fuerza del estado*)
Vacía	❖ Desapila un elemento (por la *fuerza del emisor*), o bien ❖ No desapila (por la *fuerza del estado*)	❖ No desapila (por la *fuerza del emisor*), o bien ❖ Notifica un error (por la *fuerza del estado*)

Para remediar este tipo de problema podría ignorarse la influencia de una o más de las fuerzas en conflicto, de manera tal que sólo una pueda predominar, pero de ese modo se limitaría en gran medida el comportamiento subjetivo de un objeto.

Una solución más eficaz se desprende de un razonamiento lógico aplicado a las fuerzas en conflicto. Siguiendo con el ejemplo de la pila, se puede afirmar que tiene sentido considerar la influencia del emisor sólo en los casos en que la pila no está vacía, debido a que, si la pila no tiene elementos, es superfluo el tipo del emisor (sea un administrador de tipo A, de tipo B, o de cualquier otro tipo, no podrá sacar un elemento de una pila vacía).

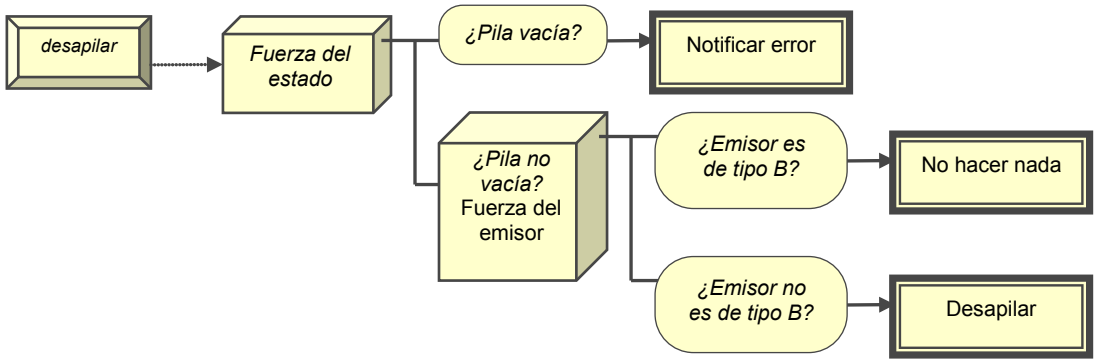
Consecuentemente, cuando se evalúe el mensaje *desapilar,* tiene sentido que en primera instancia se determine cómo predomina la *fuerza del estado*. En el caso de que la pila esté vacía, se notificará un error; en el caso de que la pila tenga uno o más elementos se determinará como influye la *fuerza del emisor* que, como se especificó anteriormente, la pila desapilará un elemento siempre y cuando el objeto emisor no sea un administrador de tipo B. Esta solución podría especificarse de la siguiente manera:

- Si la pila **está vacía** → notifica un error.
- Si la pila **no está vacía** →
  - Si el emisor **entiende el protocolo** definido por la clase *AdministradorTipoB* → no desapila el último elemento.
  - Si el emisor **no entiende el protocolo** definido por la clase *AdministradorTipoB* → desapila el último elemento.

El punto clave es establecer un orden o prioridad de evaluación entre las posibles fuerzas que pueden afectar el comportamiento asociado a un cierto requerimiento de un objeto. De este modo, cuando se presenta un conflicto entre fuerzas, es necesario decidir para cada una el orden en el cual debe evaluarse, y generalmente se desprende de manera natural de un razonamiento coherente empleado para analizar el comportamiento subjetivo en cuestión. El resultado es un *árbol de fuerzas,* el cual se describe a continuación.

ÁRBOL DE FUERZAS

Continuando con el ejemplo de la pila subjetiva, en la solución planteada para definir el mensaje subjetivo *desapilar* se presenta implícitamente una estructura de árbol de este tipo:



Este árbol se asemeja a lo que se denomina árbol de fuerzas, el cual establece el orden de evaluación de las fuerzas involucradas en un mensaje subjetivo; en consecuencia, cuando se invoque el mensaje *desapilar* la primer fuerza que se evaluará es la fuerza del estado, que analizará el estado de la pila y si determina que:

- La pila **está vacía** → conduce a que la pila notifique un error.
- La pila **no está vacía** → delega la búsqueda de la respuesta subjetiva en la fuerza del emisor, para que dicha fuerza conduzca al comportamiento correcto que debe tener la pila.

El árbol del ejemplo está configurado de forma tal que, cuando la fuerza del estado verifica que la pila se encuentra en estado vacía, induce directamente una respuesta específica de la pila (notificar un error), y cuando la pila no está vacía provoca la evaluación de otra fuerza (la del emisor) para "seguir buscando" la respuesta que debe dar la pila. Por consiguiente, dependiendo de cual sea el estado de la pila, se determina la resolución de una respuesta subjetiva o bien, se determina una nueva fuerza a evaluarse. Esto da origen a dos clases de

elementos determinantes que forman un árbol de fuerzas: *determinantes de implementación* y *determinantes de fuerzas*. Seguidamente, se definen cada una de estas clases de determinantes.

Organización estructural de un Árbol de Fuerzas

Como se señaló anteriormente, un árbol de fuerzas se compone de dos tipos de determinantes: determinantes de implementación y determinantes de fuerza. Ambos tipos poseen características y funcionalidades comunes, que se explicarán cuando se desarrolle el concepto de determinante; subsiguientemente, se expondrán las particularidades propias de cada tipo.

Determinante

Definición

Todos los nodos que conforman un árbol de fuerzas son objetos determinantes. Usualmente, cada uno de éstos posee un determinante de fuerza como nodo padre, y conserva un valor, conocido como *valor del determinante*; dependiendo de cómo fue configurado dicho valor, puede tratarse de:

- Un conjunto de elementos donde cada uno especifica una *característica del objeto,*
- Un elemento especial.

Si el valor del determinante es un conjunto de *características del objeto*, cada uno de sus componentes puede referirse a:

- La identidad de un objeto,
- El protocolo que entiende un objeto, o
- Cierta condición que cumple un objeto.

La manera de especificar cada una de estas características es la siguiente:

CARACTERÍSTICA DEL OBJETO	MEDIO DE ESPECIFICACIÓN
Identidad de un objeto	La referencia al objeto que permite a otros objetos llegar a él.
Protocolo que entiende un objeto	La clase del objeto, debido a que es una forma simplificada de referirse a todos los mensajes que puede entender. De otro modo, debería especificarse el protocolo de un objeto listando todos los mensajes que entiende.
Cierta condición que cumple un objeto	Un mensaje, el cual será enviado al objeto para verificar si cumple tal condición.

El segundo tipo de valor del determinante, es un elemento especial que es de fundamental importancia en la estructura del árbol y en su forma de evaluarse; su significado se explicará en la sección *Lógica de evaluación*.

Funcionalidad

Dependiendo de que tipo sea, un determinante puede:

- Si es un determinante de implementación, conducir a una respuesta subjetiva, o
- Si es un determinante de fuerza, conducir a la evaluación de otro determinante.

Si en efecto puede realizar tal conducción, se debe a que el valor del determinante que conserva "satisface" a un cierto objeto. Es decir, si el valor se trata de:

- Un conjunto de *características del objeto*, a saber:
  - Un conjunto de identidades → alguna de las referencias especificadas en dicho conjunto coincide con la referencia asociada al objeto a satisfacer,
  - Un conjunto de protocolos → alguna de las clases que especifica un protocolo en dicho conjunto es la misma clase o una superclase de la clase del objeto a satisfacer,
  - Un conjunto de condiciones → se cumple una condición de dicho conjunto en el objeto a satisfacer, lo cual se comprueba enviándole el mensaje que la especifica a dicho objeto y verificando el resultado positivo.
- Un elemento especial → siempre satisface al objeto. Además, como se verá más adelante, cuando un determinante está configurado con este tipo de elemento es insignificante la presencia del objeto a satisfacer.

Cuando sucede alguno de estos casos, se dice que el determinante predomina, y por tal motivo realizará una acción (dependiendo de que tipo de determinante sea) que conducirá en algún sentido la búsqueda de la respuesta subjetiva que debe dar el objeto. Es importante recalcar que en el caso particular de un determinante configurado con el elemento especial, el mismo siempre predominará.

El origen del objeto que debe satisfacer un determinante se dará a conocer cuando se expliquen los determinantes de fuerza.

Determinante de Fuerza

Definición

Un determinante de fuerza es una clase de nodo del árbol de fuerzas. Cumple con la definición y funcionalidad de un determinante, y además colabora con los siguientes objetos:

- Un objeto *tipo de fuerza,*
- Un conjunto de otros determinantes conocidos como *determinantes hijos*.

Un objeto *tipo de fuerza* es el responsable de capturar un objeto. Dependiendo de qué tipo de fuerza represente se tiene lo siguiente:

TIPO DE FUERZA	ACCION DE CAPTURA
Fuerza del emisor	Captura al objeto emisor de un mensaje subjetivo.
Fuerza del colaborador interno	Captura a un colaborador interno de un objeto subjetivo.
Fuerza del colaborador externo	Captura a un colaborador externo de un objeto subjetivo, el cual forma parte del conjunto de parámetros de un mensaje subjetivo enviado a dicho objeto.
Fuerza del contexto	Captura a un objeto del contexto de un objeto subjetivo.
Fuerza del estado	Captura al objeto receptor de un mensaje subjetivo.

En particular, la fuerza del colaborador interno, la fuerza del colaborador externo y la fuerza del contexto deben tener especificado a cual objeto deben capturar, debido a que un objeto subjetivo puede tener más de un colaborador interno, más de un colaborador externo (del mensaje subjetivo) y más de un objeto en su red de colaboración que determina su contexto.

Los determinantes hijos pueden ser determinantes de implementación u otros determinantes de fuerzas. Retomando el ejemplo de la pila subjetiva, se puede deducir que el árbol de fuerzas asociado al mensaje *desapilar* tiene como nodo raíz un determinante de fuerza, cuyo tipo de fuerza es estado, y posee dos determinantes hijos:

- Un determinante de implementación.
- Un determinante de fuerza, con la fuerza del emisor como tipo de fuerza asociada.

Es importante enfatizar tres aspectos fundamentales del determinante raíz de un árbol de fuerza:

1. No posee un determinante de fuerza padre.
2. Su valor de determinante siempre es el elemento especial.
3. Es un determinante de fuerza.

La razón de estos requisitos se entenderá cuando se explique el funcionamiento de un determinante de fuerza y un determinante de implementación.

Funcionalidad

Básicamente, un determinante de fuerza tiene que cumplir con dos responsabilidades:

Responsabilidad 1. Verificar si predomina.
Responsabilidad 2. Delegar la búsqueda de la respuesta subjetiva en alguno de sus determinantes hijos.

El segundo requerimiento del cual es responsable dicho determinante implica los siguientes pasos:

Paso 1. Solicitar a su objeto tipo de fuerza que capture el objeto adecuado, es decir, aquel objeto que el tipo de fuerza es responsable de capturar.
Paso 2. Buscar dentro del conjunto de determinantes hijos aquel que predomine con dicho objeto, es decir, el determinante hijo cuyo valor satisfaga al objeto capturado en el Paso 1.
Paso 3. Delegar en el determinante hijo predominante la búsqueda de la respuesta subjetiva.

Cabe destacar que si el hijo que predomina es un determinante de fuerza, este último realizará la misma secuencia de pasos.

La primer responsabilidad se satisface de la manera que se especificó en la funcionalidad de un determinante, y está muy relacionada con el Paso 2 del proceso anterior, debido a que, para verificar si predomina, un determinante debe comprobar que su valor satisfaga el objeto capturado por el tipo de fuerza de su determinante de fuerza padre.

Para el caso del determinante de fuerza raíz, es fundamentar que su valor sea el elemento especial, ya que, por el hecho de no tener determinante de fuerza padre, la verificación de su predominio no puede depender de la comprobación de satisfacción a un objeto (al no existir un determinante padre, no existe el objeto tipo de fuerza que capture a este objeto). Por lo tanto, un determinante de fuerza raíz siempre predomina, por definición del elemento especial, y esto tiene sentido que sea de esta forma para poder iniciar la evaluación del árbol de fuerza, como se explicará más adelante.

Determinante de Implementación

Definición

Un *determinante de implementación* es una clase de nodo del árbol de fuerzas. Cumple con la definición y funcionalidad de un determinante, y además colabora con un método subjetivo.

En el ejemplo de la pila, el árbol de fuerzas asociado al mensaje *desapilar* tendría tres determinantes de implementación:

- Uno para conducir a la respuesta subjetiva “Notificar error”, cuyo determinante padre es el nodo raíz.
- Uno para conducir a la respuesta subjetiva “No hacer nada”, con el determinante de la fuerza del emisor como padre.
- Uno para conducir a la respuesta subjetiva “Desapilar”, con el mismo padre que el anterior.

Cada una de estas respuestas subjetivas se encuentra codificada en el método subjetivo que conoce cada uno de estos determinantes.

La principal diferencia que tiene un determinante de implementación con un determinante de fuerza es que no puede contener hijos; los determinantes de implementación son las hojas de un árbol de fuerzas.

Funcionalidad

Para un determinante de implementación es muy sencilla su funcionalidad: si se verifica que predomina conduce a la ejecución de su método subjetivo asociado.

La verificación de su predominio la realiza de la misma manera que lo hace un determinante de fuerza: dado un objeto capturado por su determinante padre comprueba si su valor satisface a dicho objeto, si así sucede significa que es el predominante.

Lógica de evaluación de un Árbol de Fuerzas

Cada mensaje subjetivo está definido por un árbol de fuerzas. Cuando a un objeto se le envía un mensaje de este tipo, implícitamente se evalúa su árbol asociado para determinar cual es la respuesta subjetiva que debe brindar. Básicamente, este árbol define una estrategia de evaluación que da como resultado el método subjetivo a ejecutar por el objeto, en respuesta al mensaje recibido. El proceso que implica esta búsqueda es transparente al objeto subjetivo, debido a que en ningún momento se requiere de su colaboración en tal proceso.

En el instante que un objeto recibe un mensaje subjetivo ocurren secuencialmente dos sucesos:

- El establecimiento del ECS, y
- La evaluación del árbol de fuerzas

Se explicará a continuación cada uno de estos.

Establecimiento del ECS

Como se expuso anteriormente, el entorno del comportamiento subjetivo está formado por ciertos objetos que cumplen un determinado rol en relación con un objeto subjetivo. Cuando dicho objeto recibe un mensaje subjetivo se establece su ECS, el cual puede concebirse como el medio que permite capturar a todos aquellos objetos que pueden “forzar”

de alguna forma su comportamiento. La manera de determinar el ECS dependerá de las herramientas que brinda el ambiente de programación.

Evaluación del Árbol de Fuerzas

Una vez establecido el ECS, comienza la evaluación del árbol en su determinante de fuerza raíz. Como se explicó en el funcionamiento de esta clase de determinante, el primer paso implica la verificación de su predominio. Ineludiblemente, la raíz debe verificar el predominio de sus hijos, y este hecho es uno de los motivos por el cual surgió la necesidad de definir el elemento especial, para poder establecer que un determinante indefectiblemente predomine (como por ejemplo, la raíz del árbol de fuerzas). Otra posible solución podría consistir en definir una clase de determinante de fuerza especial, que no necesite verificar su predominio para continuar la evaluación del árbol, pero por cuestiones de diseño del problema no se optó por dicha solución. Además, se verá más adelante que la definición del elemento especial tiene una importante consecuencia en la definición y evaluación del árbol de fuerzas.

Continuando con el proceso de evaluación, el determinante de fuerza raíz debe buscar al determinante hijo que predomine, para lo cual, realiza los siguientes tres pasos:

Paso 1. Solicita a su objeto tipo de fuerza que capture del ECS el objeto correcto.
Paso 2. Busca al primer determinante hijo cuyo valor satisfaga al objeto capturado; para esto, solicita secuencialmente a cada uno de éstos que verifique su predominio basándose en este objeto.
Paso 3. Una vez encontrado el primer hijo predominante, delega en él la búsqueda de la respuesta subjetiva.

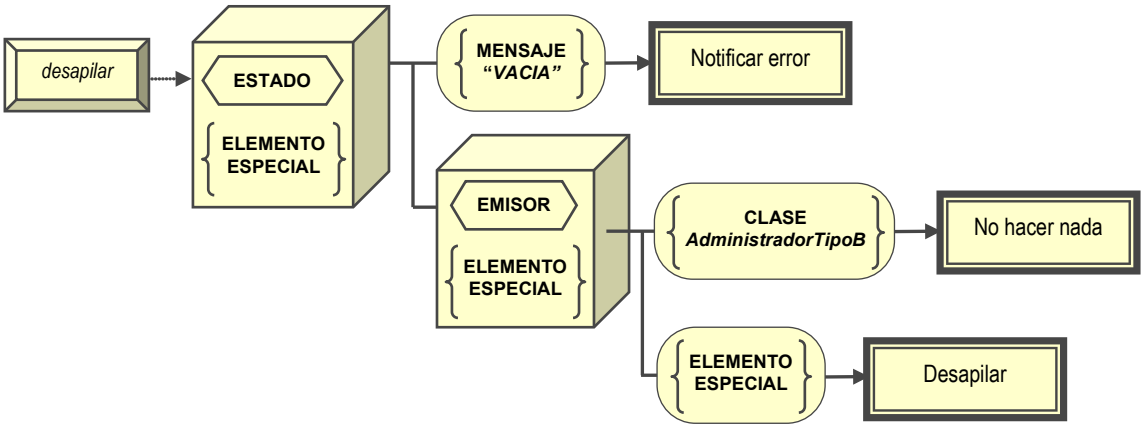
En este punto cabe destacar un motivo más por el cual tiene sentido la definición del elemento especial. En el ejemplo anterior, se concluyó que el determinante de la fuerza del emisor tiene dos determinantes de implementación como hijos. Uno de ellos conduce a la respuesta subjetiva “No hacer nada”, y tiene configurado como valor un conjunto con un único elemento, la clase *AdministradorTipoB;* de ese modo, especifica que si el emisor del mensaje subjetivo respeta el protocolo definido en dicha clase deberá conducir a esa respuesta. Para el segundo determinante, el que conduce a la respuesta “Desapilar”, es necesario especificar dentro del conjunto asociado a su valor del determinante, las clases de aquellos objetos que pueden efectivamente solicitarle a la pila que desapile. En el ejemplo se detalla que las instancias de la clase *AdministradorDeTipoA* pueden manipular la pila sin inconvenientes, pero no menciona si otros objetos del sistema tienen que utilizar con o sin restricciones esta clase de pila; por la definición del problema sólo los administradores de tipo B tienen denegado el desapilado. De este modo, los objetos de todas las clases menos *AdministradorTipoB* podrían desapilar sin inconvenientes de la pila subjetiva. Sería necesario incluir en el conjunto del valor del determinante todas las clases del ambiente, y por tal razón se requirió del elemento especial, debido a que es una forma simplificada de especificar que “en cualquier caso” el determinante predomina.

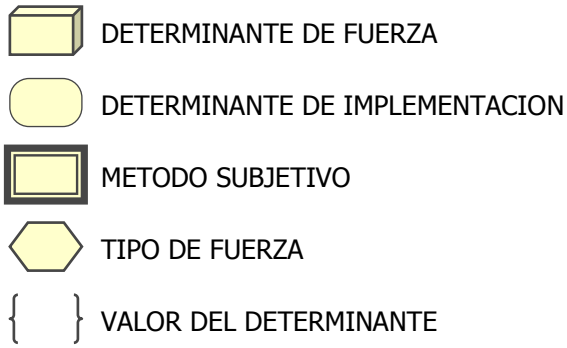
Asimismo, es de fundamental importancia el orden en que se ubica un determinante cuyo valor es el elemento especial en relación a sus hermanos (es decir, los determinantes que son hijos de su mismo padre). Dado que dicho determinante siempre predomina, debe dar prioridad primero a sus hermanos para que verifiquen su predominio, y a causa de esto, necesariamente debe ubicarse al final de la lista de hijos de su nodo padre.

Siguiendo en la evaluación, el determinante que predomina debe continuar con el proceso; si se trata de un determinante de fuerza realizará básicamente el procedimiento de tres pasos que realizó su padre, en cambio, si es un determinante de implementación el que predomina, significa que se encontró la respuesta subjetiva que debe brindar el objeto y finaliza el proceso de búsqueda; hasta ese punto llega la evaluación de árbol de fuerzas.

Ejemplo de evaluacion de un Árbol de Fuerza

Para clarificar la forma en que se evalúa un árbol de fuerzas, se mostrará la evaluación del árbol asociado al ejemplo. Se asume la siguiente configuración del mismo:



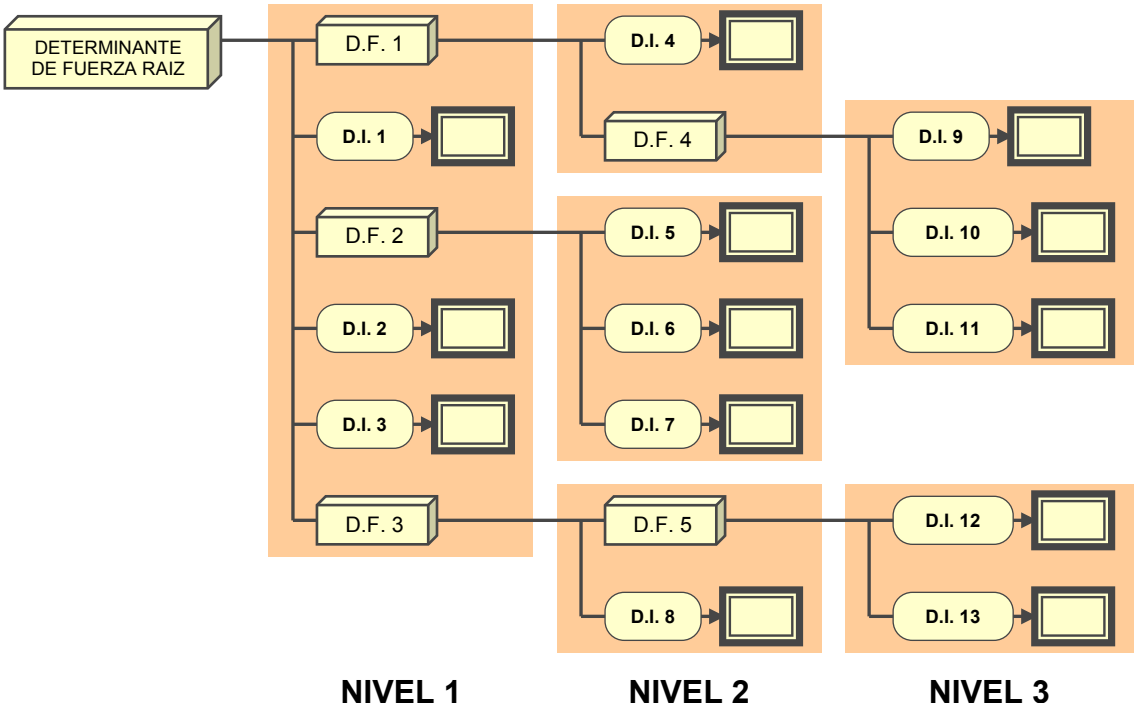


Considere el caso en que una instancia de la clase *AdministradorDeTipoA* envía el mensaje *desapilar* a una pila subjetiva que contiene al menos un elemento. En ese momento se determina su ECS. El determinante de fuerza raíz del árbol de fuerzas asociado a dicho mensaje es el que conoce al tipo de fuerza estado. Como se mencionó anteriormente, este determinante predominará por tener al elemento especial como valor y solicitará a su objeto tipo de fuerza que capture al objeto correcto, que en este caso se trata de la misma pila subjetiva. Luego, el determinante fuerza del estado busca al hijo que predomina. En primer lugar solicita a su único hijo determinante de implementación la verificación de su predominio, el cual le envía el mensaje *vacía* a la pila subjetiva para determinarlo. Dado que la pila no está vacía, este determinante recibe una respuesta negativa, y por ende, no predomina. La raíz continúa la verificación en su último hijo, el determinante fuerza del emisor. Dicho determinante de fuerza sí predomina, ya que tiene como valor de determinante el elemento especial. Otra opción podría ser que el valor de este determinante sea configurado con un mensaje *noVacia*, en lugar de utilizar el elemento especial, pero implica que la pila debe entender tal mensaje y además requiere la comprobación del estado asociado (no vacía) en la misma; de este modo es más eficiente y práctico utilizar el elemento especial. Entonces, el determinante raíz delega la evaluación en su único determinante de fuerza hijo. De la misma manera que su padre, este determinante le pide a su tipo de fuerza que recupere el objeto adecuado del ECS, en este caso, el emisor del mensaje subjetivo. Posteriormente, busca al hijo que predomina. El primero de éstos no predomina, ya que su valor no satisface al emisor (es decir, el emisor no tiene la misma clase que figura como único elemento del conjunto asociado al valor del determinante). El segundo de sus hijos, sí predomina, por estar configurado con el elemento especial. De esta

manera, por tratarse de un determinante de implementación culmina la evaluación del árbol de fuerzas y la pila subjetiva dará como respuesta el desapilado de un elemento.

Conclusiones de la estructura y evaluacion del Árbol de Fuerzas

Un aspecto importante que se observa en la estructura de un árbol de fuerza es que los determinantes están organizados por niveles. Por ejemplo, podría encontrarse un árbol con el siguiente esqueleto:



Todos los determinantes agrupados en un mismo bloque de un nivel poseen el mismo determinante de fuerza padre. Es fundamental en la evaluación del árbol que al menos uno de ellos predomine si su determinante padre ha predominado. Debido a que la formación del árbol de fuerzas depende del usuario programador, puede cometer errores tales como no definir correctamente los valores de los determinantes de un mismo bloque de manera tal que al menos uno de ellos predomine en la evaluación (si su padre ha predominado). Por tal motivo, se ha establecido que el último de los hijos de un determinante de fuerza (es decir, el que se encuentra en la parte inferior del bloque) debe tener como valor el elemento especial; es responsabilidad del programador diseñar adecuadamente a los determinantes para que el comportamiento subjetivo que "codifican" tenga sentido.

Adicionalmente, pueden sacarse las siguientes conclusiones en la evaluación de un árbol:

- El número de nivel donde se encuentra el determinante de implementación que conduce a la respuesta subjetiva (aquel que predominó en la evaluación del árbol) indica la cantidad de determinantes de fuerzas evaluados, que podría interpretarse como la cantidad de fuerzas que influyeron en el comportamiento del objeto.

- Generalmente, el árbol no se evalúa en su totalidad. La evaluación llega hasta el primer determinante de implementación predominante; por lo tanto, pueden existir determinantes que no se tienen en cuenta en una cierta evaluación del árbol.
- Si dentro de un mismo bloque existe más de un determinante que predomina, sólo se tendrá en cuenta al primero de ellos, es decir, al que se encuentra en la parte más superior del bloque; un buen diseño del árbol implica que sólo un determinante dentro de un mismo bloque debe predominar en cada evaluación, lo cual depende del programador.

A continuación, se explicará el modelo de clases asociado al comportamiento subjetivo. En primer instancia, se presentarán las tarjetas CRCs (Clase-Responsabilidades-Colaboraciones) para describir las clases del modelo, luego se presentarán los diagramas de clase, y finalmente un diagrama de interacción que refleja la evaluación de un mensaje subjetivo.

CRCs

Las tarjetas CRC permiten denotar las clases con su superclases/subclases, sus responsabilidades y colaboraciones. Dichas tarjetas se utilizarán para describir las clases involucradas en la definición del comportamiento subjetivo.

SubjectiveBehaviorCompiledMethod (Clase concreta)	
DESCRIPCION	
Representa a los mensajes subjetivos.	
SUPERCLASE	
❖ *CompiledMethod*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Inicializarse a partir del código compilado y de los valores de ciertas variables de una instancia de la clase CompiledMethod.	❖ *CompiledMethod* ❖ *SmallInteger* ❖ *SubjectiveBehaviorCompiledMethod*
Reemplazar una instancia de CompiledMethod consigo misma (aquella que implementa el mismo mensaje), en el diccionario de métodos de la clase a la cual pertenece.	❖ *Class* ❖ *ByteString* ❖ *Classorganizer*
Determinar si es un mensaje múltiple[8].	❖ *True*
Retornar un código fuente para presentar al usuario/programador.	❖ *ByteString* ❖ *ImplementationDeterminant*

[8] Mensaje múltiple: Mensaje que tiene asociada más de una implementación.

SubjectiveBehaviorDeterminant (Clase abstracta)	
DESCRIPCION	
Representa a los determinantes que forman el árbol de fuerzas.	
SUPERCLASE	
❖ *Object*	
SUBCLASES	
❖ *ForceDeterminant* ❖ *ImplementationDeterminant*	
RESPONSABILIDADES	*COLABORACIONES*
Retornar el determinante de implementación que predomina bajo un entorno del comportamiento subjetivo dado. Nota: Debe implementarse en las subclases.	
Participar en la tarea de agregarse como hijo de un determinante de fuerza dado.	❖ *SubjectiveBehaviorDeterminantType* ❖ *ForceDeterminant* ❖ *SubjectiveBehaviorDeterminant*
Determinar si posee hijos. Nota: Dado que no todos los determinantes poseen hijos, deberá reimplementarse en las subclases que corresponda.	❖ *False*
Eliminarse de la lista de hijos de su determinante de fuerza padre.	❖ *ForceDeterminant* ❖ *SubjectiveBehaviorDeterminant*
Retornar el código fuente de su método subjetivo asociado. Nota: Dado que no todos los determinantes poseen implementaciones asociadas, deberá reimplementarse en las subclases que corresponda.	❖ *ByteString*
Retornar una cadena de caracteres para describirse.	❖ *ByteString* ❖ *SubjectiveBehaviorDeterminant*

ForceDeterminant (Clase concreta)	
DESCRIPCION	
Representa a los determinantes de fuerza.	
SUPERCLASE	
❖ *SubjectiveBehaviorDeterminant*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Buscar y retornar dentro de su lista de hijos el determinante de implementación que predomina bajo un entorno del comportamiento subjetivo dado.	❖ *OrderedCollection* ❖ *SubjetiveBehaviorForce* ❖ *SubjectiveBehaviorDeterminantType* ❖ *SubjectiveBehaviorDeterminant* ❖ *DefaultSubjectiveMessageEnvironment*
Determinar si tiene un hijo definido por defecto.	❖ *Boolean* ❖ *SubjectiveBehaviorDeterminant*
Determinar si tiene hijos.	❖ *Boolean* ❖ *OrderedCollection* ❖ *ForceDeterminant*
Agregar un determinante como un nuevo hijo.	❖ *SubjectiveBehaviorDeterminant*
Retornar a todos sus hijos.	❖ *OrderedCollection* ❖ *ForceDeterminant*
Agregar un determinante al final de su lista de hijos.	❖ *SubjectiveBehaviorDeterminant* ❖ *ForceDeterminant* ❖ *OrderedCollection*
Eliminar a un determinante hijo de su lista de hijos.	❖ *SubjectiveBehaviorDeterminant* ❖ *ForceDeterminant* ❖ *OrderedCollection*

ImplementationDeterminant (Clase concreta)	
DESCRIPCION	
Representa a los determinantes de implementación.	
SUPERCLASE	
❖ *SubjectiveBehaviorDeterminant*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Retornarse como determinante predominante.	❖ *ImplementationDeterminant*
Retornar el código fuente de su método subjetivo asociado.	❖ *ByteString*

SubjectiveBehaviorForce (Clase abstracta)	
DESCRIPCION	
Representa a los tipos de fuerza.	
SUPERCLASE	
❖ *Object*	
SUBCLASES	
❖ *CollaboratorForce* ❖ *StateForce* ❖ *ContextForce* ❖ *SenderForce*	
RESPONSABILIDADES	*COLABORACIONES*
Participar en la evaluación de predominio de un determinante dado, bajo un entorno del comportamiento subjetivo específico.	❖ *SubjetiveBehaviorForce* ❖ *SubjectiveBehaviorDeterminantType*
Recuperar y retornar el objeto adecuado desde un entorno del comportamiento subjetivo dado. Nota: Debido a que cada tipo de fuerza se centra en diferentes aspectos del entorno del comportamiento subjetivo, debe implementarse en las subclases.	

CollaboratorForce (Clase abstracta)	
DESCRIPCION	
Representa a la fuerza del colaborador.	
SUPERCLASE	
❖ *SubjectiveBehaviorForce*	
SUBCLASES	
❖ *InternalCollaboratorForce* ❖ *EnternalCollaboratorForce*	
RESPONSABILIDADES	*COLABORACIONES*
Mantener una referencia al colaborador (externo o interno) de un objeto subjetivo.	❖ *Integer*

InternalCollaboratorForce (Clase concreta)	
DESCRIPCION	
Representa a la fuerza del colaborador interno.	
SUPERCLASE	
❖ *CollaboratorForce*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Recuperar y retornar a un colaborador interno de un objeto subjetivo desde un entorno del comportamiento subjetivo dado.	❖ *DefaultSubjectiveMessageEnvironment* ❖ *SubjectiveBehaviorDeterminantType* ❖ *Object*

ExternalCollaboratorForce (Clase concreta)	
DESCRIPCION	
Representa a las fuerzas del colaborador externo.	
SUPERCLASE	
❖ *CollaboratorForce*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Recuperar y retornar a un colaborador externo de un objeto subjetivo desde un entorno del comportamiento subjetivo dado.	❖ *DefaultSubjectiveMessageEnvironment* ❖ *SubjectiveBehaviorDeterminantType* ❖ *Array*

StateForce (Clase concreta)	
DESCRIPCION	
Representa a la fuerza del estado.	
SUPERCLASE	
❖ *SubjectiveBehaviorForce*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Recuperar y retornar al objeto subjetivo desde un entorno del comportamiento subjetivo dado.	❖ *DefaultSubjectiveMessageEnvironment*

ContextForce (Clase concreta)	
DESCRIPCION	
Representa a la fuerza del contexto.	
SUPERCLASE	
❖ *SubjectiveBehaviorForce*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Recuperar y retornar a un objeto del contexto de un objeto subjetivo.	❖ *PathToAnObject* ❖ *SubjectiveBehaviorDeterminantType*

SenderForce (Clase concreta)	
DESCRIPCION	
Representa a la fuerza del emisor.	
SUPERCLASE	
❖ *SubjectiveBehaviorForce*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Recuperar y retornar al emisor de un mensaje subjetivo enviado a un objeto, desde un entorno del comportamiento subjetivo dado.	❖ *DefaultSubjectiveMessageEnvironment* ❖ *SubjectiveBehaviorDeterminantType*

SubjectiveBehaviorDeterminantType (Clase abstracta)	
DESCRIPCION	
Representa a los valores de un determinante.	
SUPERCLASE	
❖ *Object*	
SUBCLASES	
❖ *DeterminantTypeWithValues* ❖ *DeterminantByDefault*	
RESPONSABILIDADES	*COLABORACIONES*
Determinar si un objeto del entorno del comportamiento subjetivo es predominante. Nota: Debe implementarse en las subclases.	
Retornar una cadena de caracteres que represente a sus valores. Nota: Debido a que no todos los valores de determinante poseen valores, deberá reimplementarse en las subclases que corresponda.	❖ *ByteString*

DeterminantTypeWithValues (Clase abstracta)	
DESCRIPCION	
Representa a los valores del determinante que son conjunto de elementos.	
SUPERCLASE	
❖ *SubjectiveBehaviorDeterminantType*	
SUBCLASES	
❖ *DeterminantByCondition* ❖ *DeterminantByProtocol* ❖ *DeterminantByIdentity*	
RESPONSABILIDADES	*COLABORACIONES*
Agregar un elemento a su conjunto de valores.	❖ *OrderedCollection*
Eliminar un elemento de su conjunto de valores.	❖ *OrderedCollection*
Retornar una cadena de caracteres que represente a sus valores. Nota: Debido a que no todos los valores del determinante poseen la misma clase de elementos, deberá reimplementarse en las subclases que corresponda.	❖ *ByteString*

DeterminantByCondition (Clase concreta)	
DESCRIPCION	
Representa a los valores del determinante que especifican conjuntos de condiciones.	
SUPERCLASE	
❖ *DeterminantTypeWithValues*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Determinar si un objeto dado cumple con alguna de las condiciones que especifica en su conjunto.	❖ *OrderedCollection* ❖ *PathToAnObject*
Retornar una cadena de caracteres que represente a sus elementos (mensajes).	❖ *ByteString* ❖ *DeterminantTypeWithValues*

DeterminantByProtocol (Clase concreta)	
DESCRIPCION	
Representa a los valores del determinante que especifican conjuntos de clases.	
SUPERCLASE	
❖ *DeterminantTypeWithValues*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Determinar si la clase de un objeto dado está incluida en el conjunto de clases que conserva o si alguna de dichas clase es superclase de la misma.	❖ *OrderedCollection* ❖ *Behavior*
Recuperar y retornar la clase de un objeto dado.	❖ *Object*
Retornar una cadena de caracteres que represente a sus elementos (clases).	❖ *ByteString* ❖ *OrderedCollection* ❖ *Object*

DeterminantByIdentity (Clase concreta)	
DESCRIPCION	
Representa a los valores del determinante que especifican conjuntos de identidades de objetos.	
SUPERCLASE	
❖ *DeterminantTypeWithValues*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Determinar si la identidad a un objeto dado está incluida en el conjunto de identidades que conserva.	❖ *OrderedCollection* ❖ *PathToAnObject*

DeterminantByDefault (Clase concreta)	
DESCRIPCION	
Representa a los valores del determinante especiales.	
SUPERCLASE	
❖ *SubjectiveBehaviorDeterminantType*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Determinar si predomina.	❖ *True*

SubjectiveMessageEnvironment (Clase abstracta)	
DESCRIPCION	
Representa a los entornos del comportamiento subjetivo (ECS).	
SUPERCLASE	
❖ *Object*	
SUBCLASES	
❖ *DefaultSubjectiveMessageEnvironment*	
RESPONSABILIDADES	*COLABORACIONES*
Inicializarse a partir de un contexto de ejecución. Nota: Debe implementarse en las subclases.	

DefaultSubjectiveMessageEnvironment (Clase concreta)	
DESCRIPCION	
Representa a los entornos del comportamiento subjetivo en un ambiente orientado a objetos tradicional.	
SUPERCLASE	
❖ *SubjectiveMessageEnvironment*	
SUBCLASES	
No posee	
RESPONSABILIDADES	*COLABORACIONES*
Inicializarse a partir de un contexto de ejecución.	❖ *MethodContext* ❖ *CompiledMethod* ❖ *Array* ❖ *BlockClosure* ❖ *SmallInteger*

DIAGRAMAS DE CLASES

El modelo de clases asociado al comportamiento subjetivo se mostrará en cuatro diagramas de clase. En primer lugar, se dará una breve descripción de cada uno de ellos.

- Diagrama 1: **TIPOS DE FUERZAS**

En este diagrama se refleja el modelo de clases que representa a los tipos de fuerza (fuerza del emisor, fuerza del estado, fuerza del contexto, fuerza del colaborador interno y fuerza del colaborador externo).

- Diagrama 2: **TIPOS DE DETERMINANTES Y ENTORNO DEL COMPORTAMIENTO SUBJETIVO**

En este diagrama se refleja el modelo de clases que representa a los tipos de determinantes (por condición, por identidad, por protocolo y por defecto) y al entorno del comportamiento subjetivo (ECS).

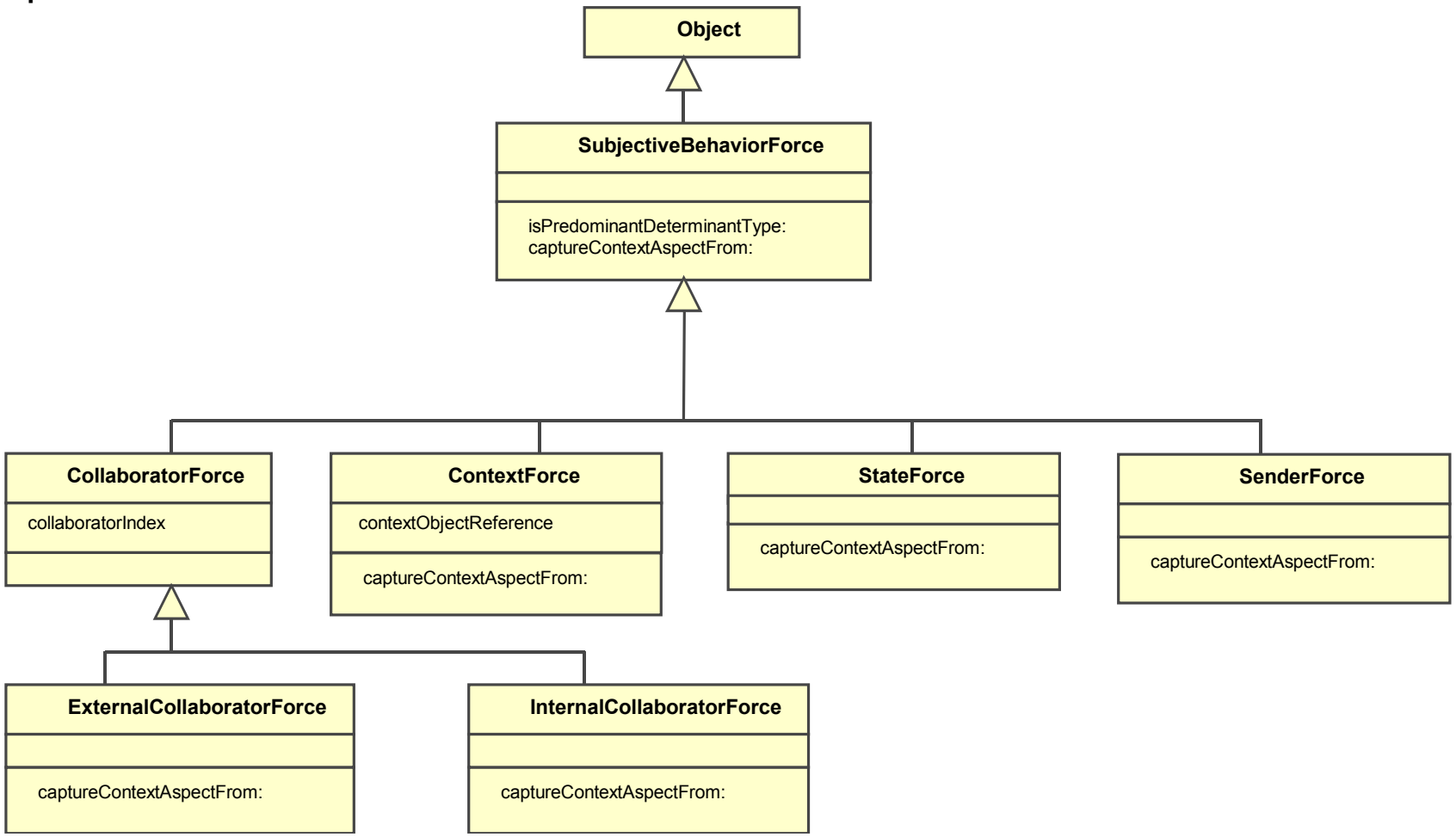
- Diagrama 3: **DETERMINANTES**

En este diagrama se refleja el modelo de clases que representa a los determinantes (determinante de fuerza y determinante de implementación) y al mensaje subjetivo. Se muestra también las relaciones existentes entre estos elementos.

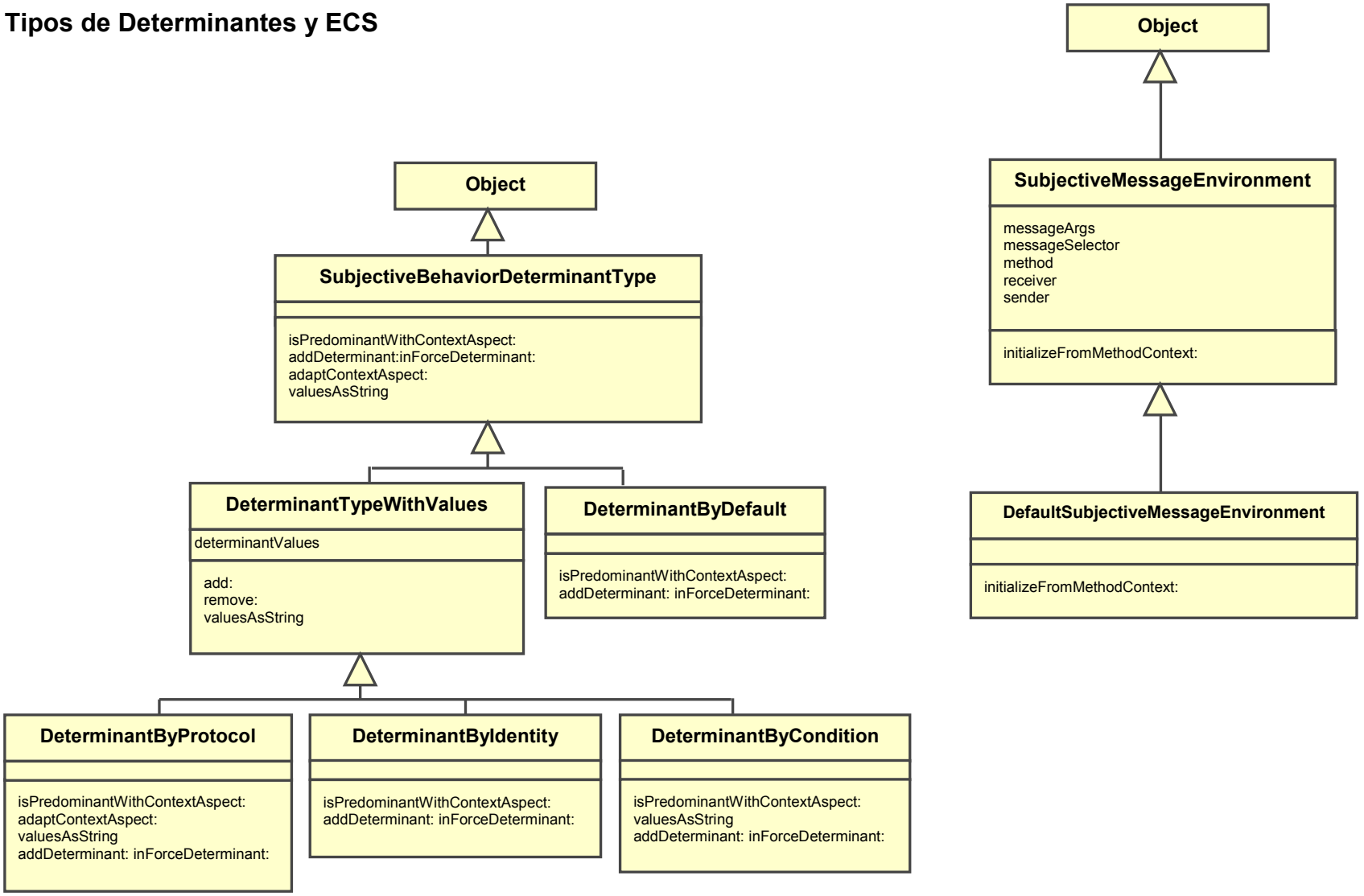
- Diagrama 4: **RELACIONES**

En este diagrama se refleja el modelo de clases que representa a todos los objetos que conforman un mensaje subjetivo y participan en su evaluación. Se refleja también las relaciones existentes entre estas distintas clases.

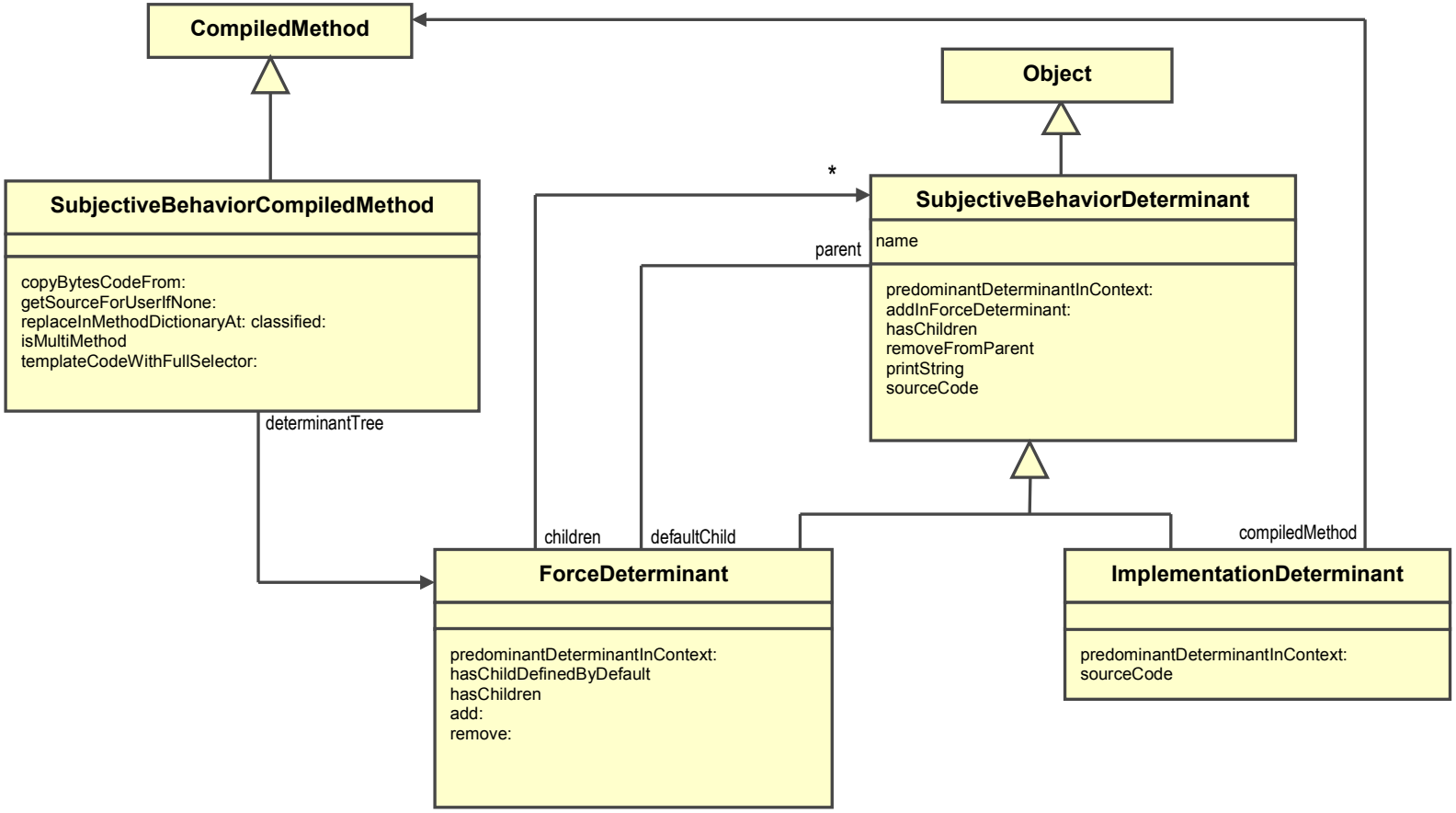
Tipos de Fuerzas



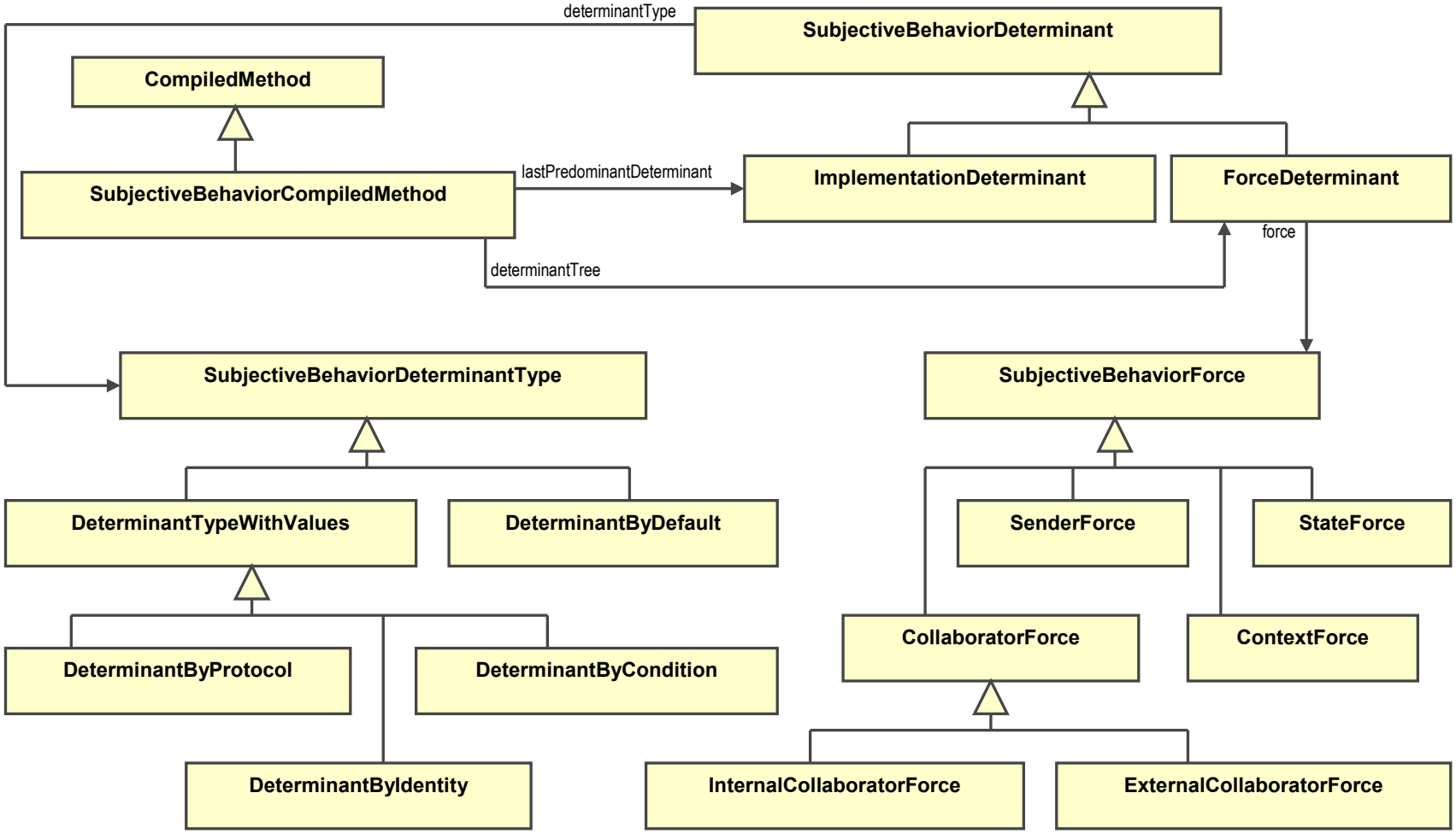
Tipos de Determinantes y ECS



Determinantes



Relaciones entre clases

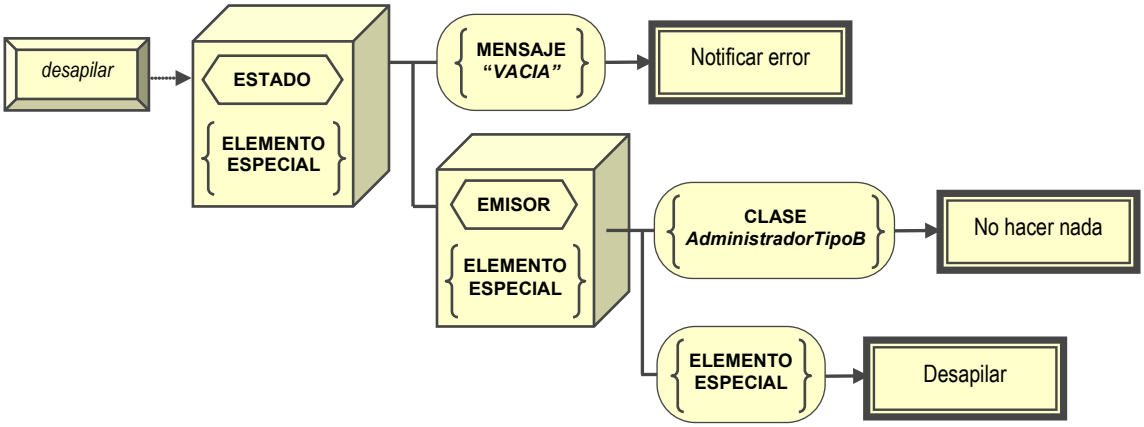


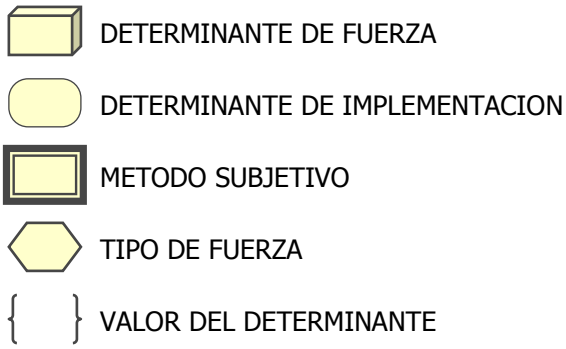
DIAGRAMAS DE INTERACCIONES

Se mostrará en seis diagramas de interacción la evaluación de un mensaje subjetivo. Para esto, se seleccionó el mensaje subjetivo *desapilar* del ejemplo de la pila subjetiva utilizado en el desarrollo de la tesis. Dicho mensaje tiene la siguiente definición:

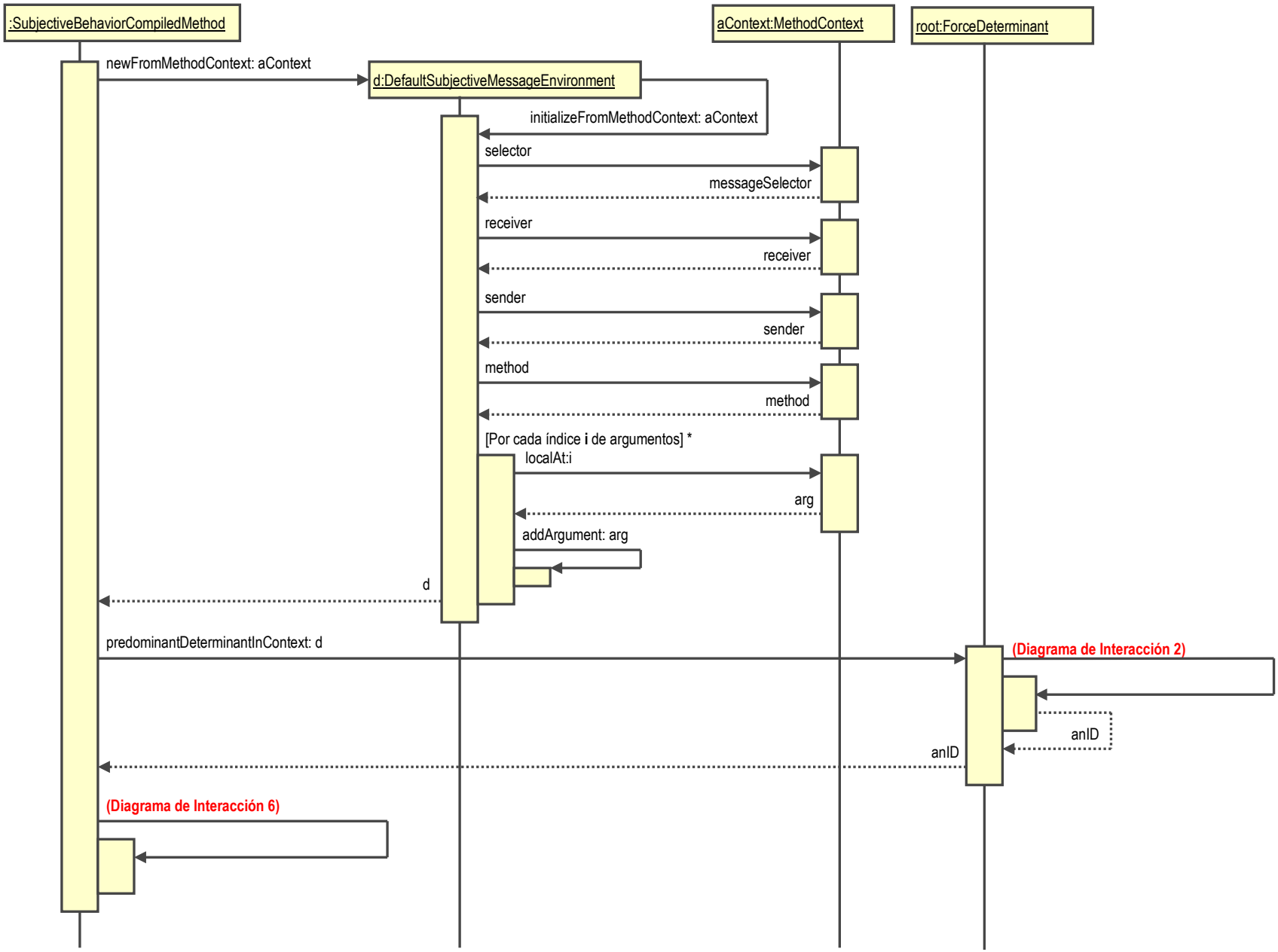
- Si la pila **está vacía** → notifica un error.
- Si la pila no **está vacía** →
  - Si el emisor **entiende el protocolo** que define la clase *AdministradorTipoB* → no desapila el último elemento.
  - Si el emisor **no entiende el protocolo** que define la clase *AdministradorTipoB* → desapila el último elemento.

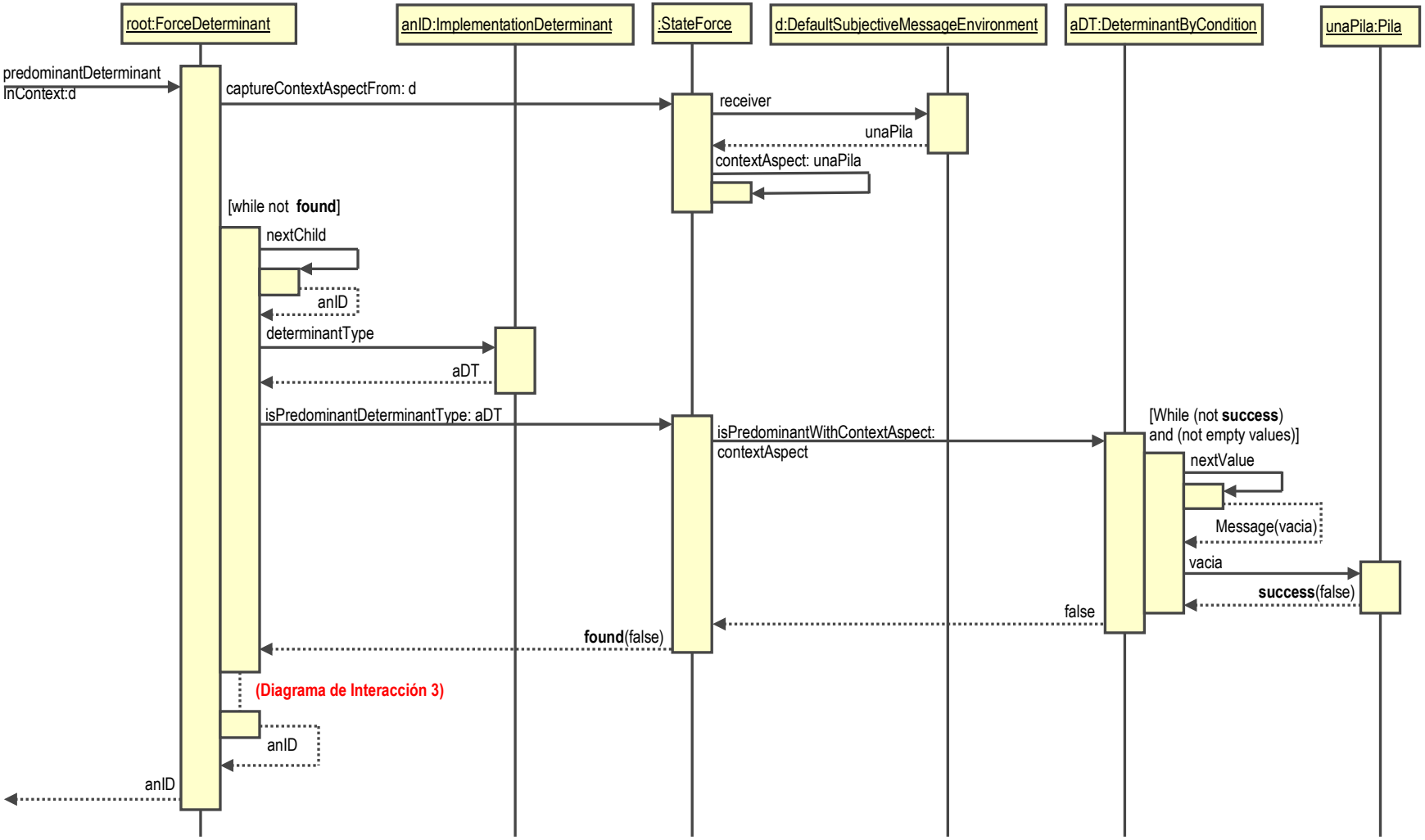
El árbol de fuerzas asociado al mensaje *desapilar* se diagrama de la siguiente manera:



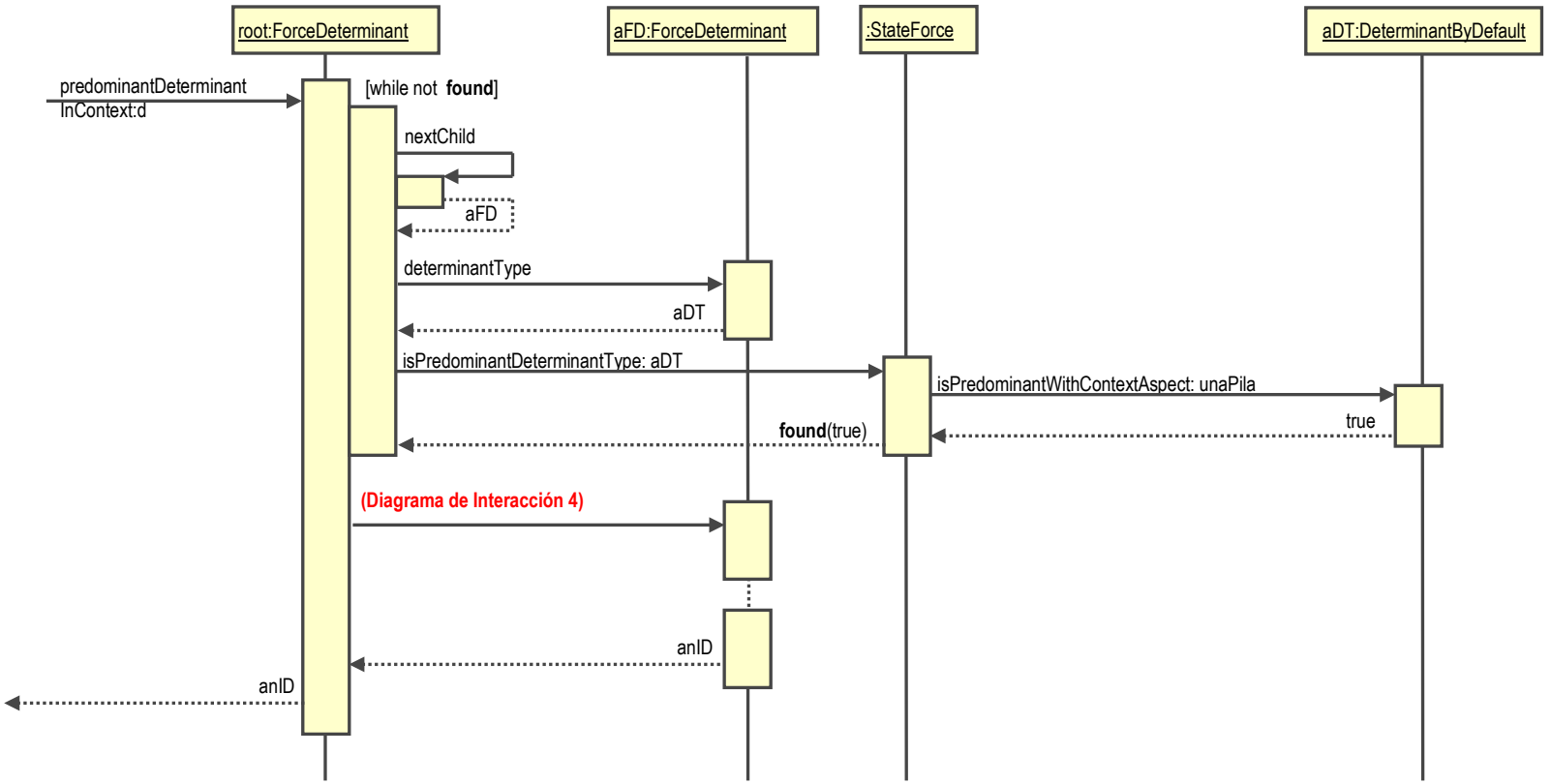


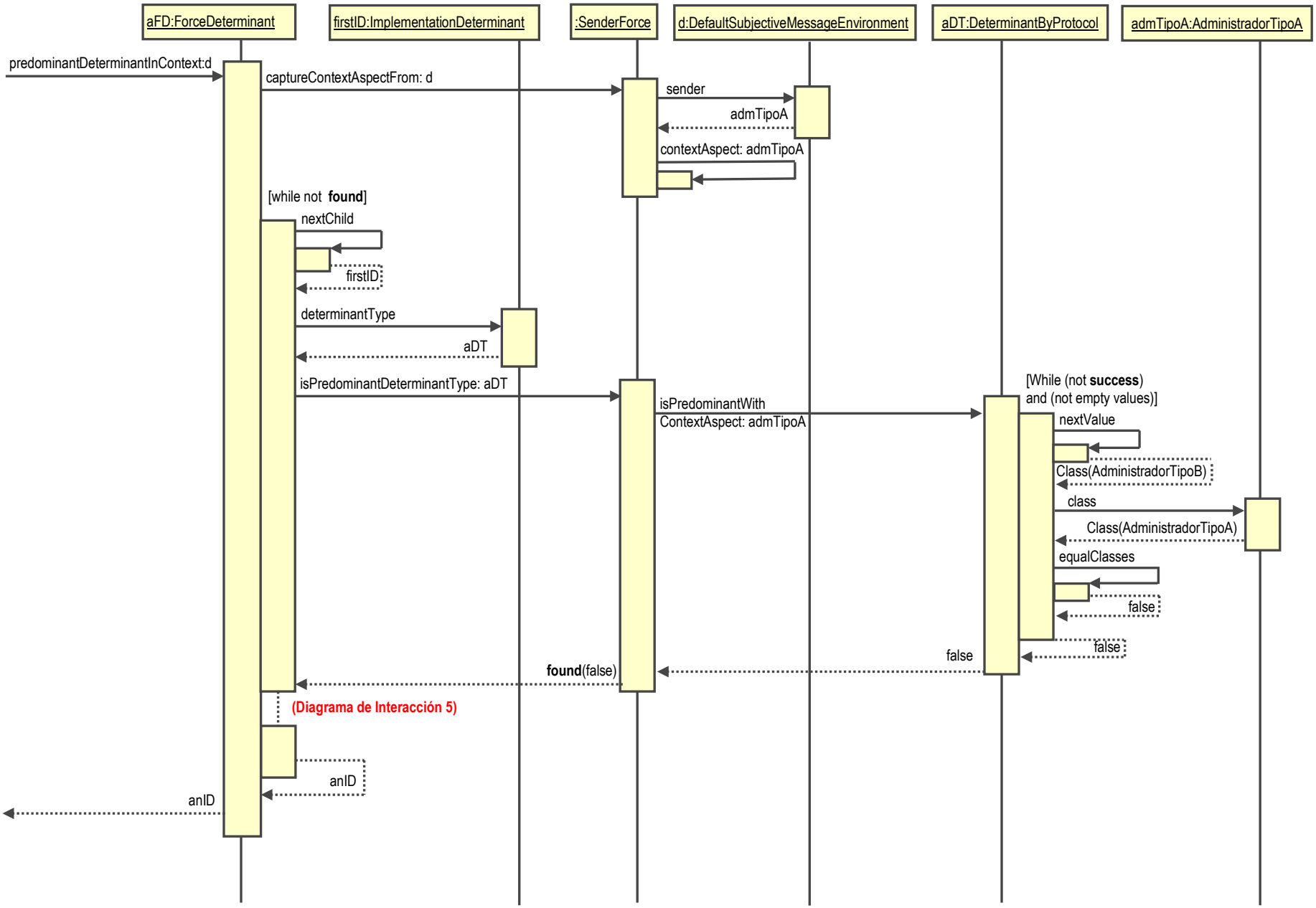
Se asume además que la pila, al momento de recibir el mensaje *desapilar*, no está vacía y el emisor del mensaje es una instancia de la clase *AdministradorTipoA*.

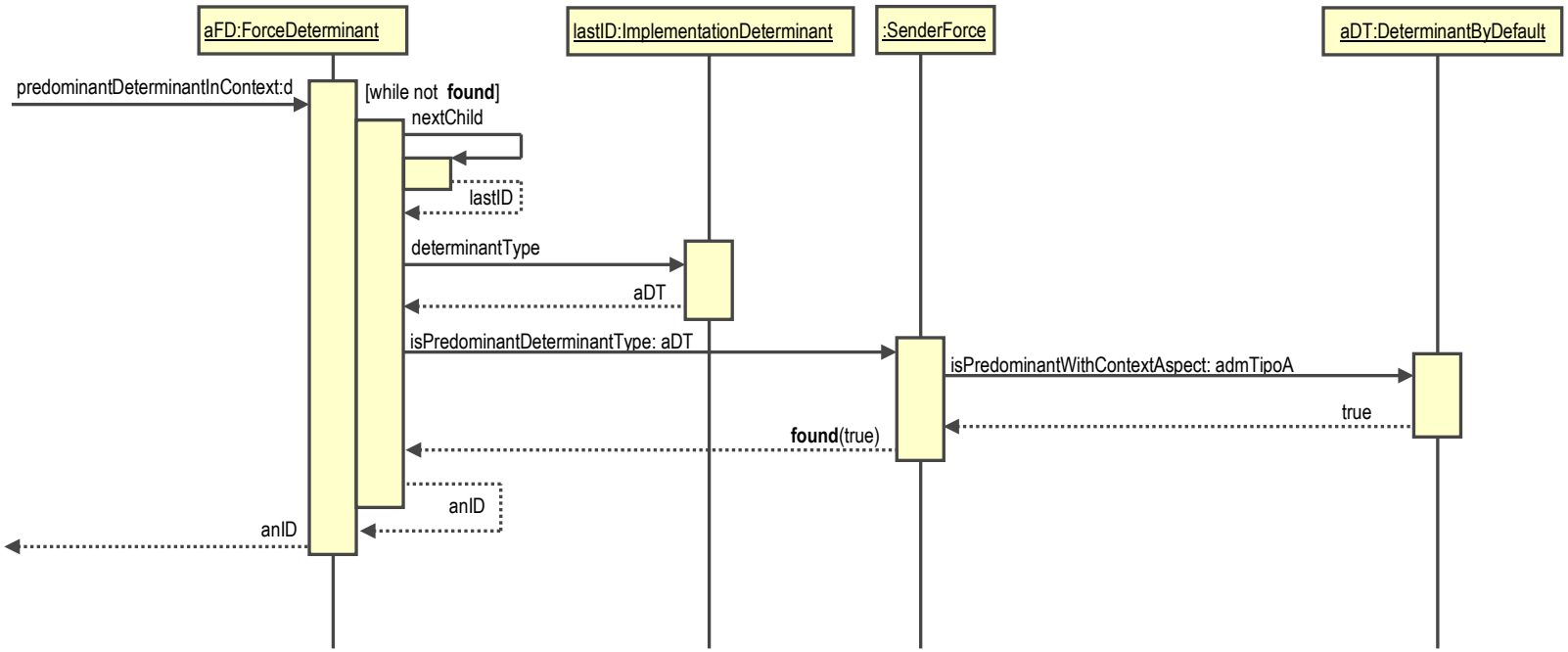

:SubjectiveBehaviorCompiledMethod
aContext:MethodContext
root:ForceDeterminant
newFromMethodContext: aContext
d:DefaultSubjectiveMessageEnvironment
initializeFromMethodContext: aContext
selector
messageSelector
receiver
receiver
sender
sender
method
method
[Por cada índice i de argumentos] *
localAt:i
arg
addArgument: arg
d
predominantDeterminantInContext: d
(Diagrama de Interacción 2)
anID
anID
(Diagrama de Interacción 6)


root:ForceDeterminant
anID:ImplementationDeterminant
:StateForce
d:DefaultSubjectiveMessageEnvironment
aDT:DeterminantByCondition
unaPila:Pila
predominantDeterminant
InContext:d
captureContextAspectFrom: d
receiver
unaPila
contextAspect: unaPila
[while not found]
nextChild
anID
determinantType
aDT
isPredominantDeterminantType: aDT
isPredominantWithContextAspect:
contextAspect
[While (not success)
and (not empty values)]
nextValue
Message(vacia)
vacia
success(false)
false
found(false)
(Diagrama de Interacción 3)
anID
anID

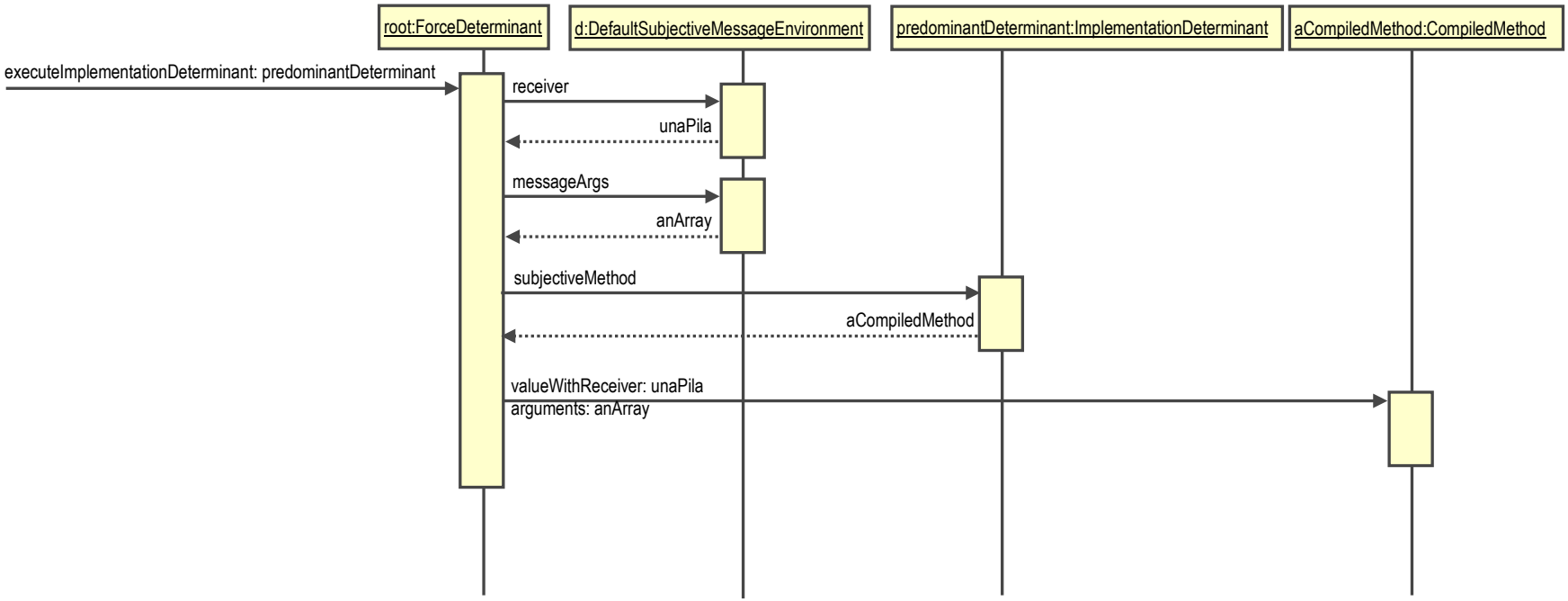
COMPORTAMIENTO SUBJETIVO – Desarrollo




aFD:ForceDeterminant
firstID:ImplementationDeterminant
:SenderForce
d:DefaultSubjectiveMessageEnvironment
aDT:DeterminantByProtocol
admTipoA:AdministradorTipoA
predominantDeterminantInContext:d
captureContextAspectFrom: d
sender
admTipoA
contextAspect: admTipoA
[while not found]
nextChild
firstID
determinantType
aDT
isPredominantDeterminantType: aDT
isPredominantWith
ContextAspect: admTipoA
[While (not success)
and (not empty values)]
nextValue
Class(AdministradorTipoB)
class
Class(AdministradorTipoA)
equalClasses
false
false
false
found(false)
(Diagrama de Interacción 5)
anID
anID


aFD:ForceDeterminant
lastID:ImplementationDeterminant
:SenderForce
aDT:DeterminantByDefault
predominantDeterminantInContext:d
[while not found]
nextChild
lastID
determinantType
aDT
isPredominantDeterminantType: aDT
isPredominantWithContextAspect: admTipoA
true
found(true)
anID
anID

COMPORTAMIENTO SUBJETIVO – Desarrollo



Para comprobar el verdadero valor del comportamiento subjetivo en un ambiente de objetos, se propuso como uno de los objetivos de esta tesis aplicar tal concepto a un dominio de aplicación particular; para tal propósito, se seleccionó a los *ambientes virtuales cooperativos orientados a objetos.* Tales ambientes poseen una definición muy precisa y, por sus particularidades, resulta muy interesante la idea de introducir "una nueva forma de comportarse" en los objetos que forman tales ambientes.

A continuación, se definirán los ambientes virtuales cooperativos y los ambientes virtuales cooperativos orientados a objetos, junto con sus características principales.

AMBIENTE VIRTUAL COOPERATIVO

Los ambientes virtuales cooperativos (*Collaborative Virtual Environments – CVEs)* surgen de las investigaciones realizadas en diferentes ramas de la computación, tales como gráficos en tiempo real, realidad virtual y espacios de trabajo cooperativos. Posteriormente, se expone una definición de CVE.

Definición

Un CVE es un espacio o mundo virtual compartido por participantes conectados a través de una red de computadoras. Dicho espacio puede estar habitado simultáneamente por individuos distribuidos, que generalmente tienen como objetivo realizar una o varias actividades en forma cooperativa con otros individuos que ocupan el espacio.

Un CVE proporciona un entorno adecuado para interactuar con objetos reales o irreales, y para realizar una gran variedad de maniobras, incluso algunas que en el mundo real resultan imposibles efectuar.

Ejemplos de mundos virtuales similares a los CVEs son los MUDs y los MOOs. Un MUD (Multi-User Domain) es una clase de comunidad virtual en la cual los usuarios participan mediante un mundo de texto; son realidades virtuales sociales basadas en texto. Básicamente, existen dos tipos de MUDs:

- Los del tipo "aventura", donde el objetivo de los mismos es similar al de los juegos.
- Los del tipo "social", donde el objetivo es interactuar con otros habitantes y, en algunos casos, ayudar a construir el mundo virtual creando objetos.

En este último tipo de MUD, la construcción del mundo virtual asociado es particularmente fácil y son los que se tendrán en cuenta en esta tesis. En particular, la tesis se basará en una clase especial de MUD social orientado a objetos, conocida como MOO. Un MOO es un ambiente orientado a objetos programable, basado en texto y multi-usuario, fácil de personalizar. Por tales características, los MOOs son ideales para formar realidades virtuales sociales, sistemas de conferencias, de aprendizaje y juegos.

Características básicas de un CVE

En esta sección se presentan los rasgos más distintivos de un CVE, a saber:

- Quiénes conforman un CVE (**Habitantes**).
- Cómo es la estructura de un CVE (**Espacio**).
- Cómo es la "dinámica" de un CVE (**Navegación**).

Habitantes

Un CVE actúa como un contenedor de objetos, los cuales son los habitantes de su espacio asociado. Cada habitante posee una representación física, es decir, está personificado de alguna forma gráfica o textual (como en el caso de los MOOs) dentro del CVE.

Básicamente, se pueden distinguir dos tipos de habitantes: *objetos virtuales* y *participantes*.

Un participante es un usuario del CVE. Normalmente, los usuarios participan en estos ambientes para realizar alguna tarea cooperativa junto a los demás usuarios participantes.

Por su parte, un objeto virtual puede ser cualquier otro objeto, como un carrito de compras, un documento, un colectivo, un agente, etc. Algunos objetos virtuales facilitan la realización de ciertas actividades cooperativas entre los habitantes, como por ejemplo un objeto que simula una mesa de conferencia o una pizarra. Asimismo, en la mayoría de los CVEs existen objetos virtuales que simulan habitaciones, para representar por ejemplo salas de encuentros, salas de conferencias, etc.

En el caso particular de los participantes, su personificación se denomina *avatar*, y por lo general, es el medio que transmite a los demás participantes su identidad, presencia, localización y las actividades que realiza.

Desde la perspectiva del comportamiento, existen tres tipos de habitantes:

- Reactivos
- Activos
- Pasivos

Un habitante reactivo es un objeto cuyo comportamiento se manifiesta cuando se lo estimula mediante el envío de un mensaje. Un habitante activo posee comportamiento autónomo, por lo cual, puede no precisar que lo estimulen otros habitantes para comportarse de alguna determinada manera. En cambio, un habitante pasivo no posee comportamiento visible y no reacciona ante el envío de mensajes.

Dentro de un CVE pueden encontrarse objetos virtuales que tienen la capacidad de contener a un cierto número de habitantes, por ejemplo, un colectivo, una habitación de conferencias, etc. De este punto de vista, es posible distinguir entre objetos virtuales *contenedores* y objetos virtuales *no contenedores*. Incluso, un mismo objeto virtual puede pertenecer a ambos tipos, por ejemplo, un monedero que contiene monedas y que al mismo tiempo puede estar contenido en una cartera.

Espacio

De acuerdo a su estructura, el espacio virtual asociado a un CVE puede pertenecer a uno de los siguientes grupos: *espacios ilimitados* o *espacios limitados*.

La primera clasificación involucra aquellos espacios virtuales que representan un área abierta o ilimitada. En cambio, un espacio virtual limitado está formado por un conjunto de pequeños subespacios virtuales conectados. Esta particularidad ofrece una buena forma de estructuración de un CVE, y favorece a la formación de metáforas de habitación, debido a que es posible diseñar cada subespacio con características particulares, por ejemplo, diseñarlos como habitaciones de conversación, habitaciones privadas, habitaciones para encuentros, etc. Asimismo, resulta más sencillo organizar actividades grupales con otros habitantes en espacios limitados y razonablemente más pequeños que en los espacios abiertos o ilimitados, debido a que dan la sensación de una mayor contención y localidad.

Navegación

En cualquier instante, los habitantes de un CVE siempre se encuentran ubicados en algún punto dentro del espacio virtual. En su gran mayoría, dichos habitantes tiene la

capacidad de trasladares (o ser trasladados por otros habitantes) a distintas ubicaciones dentro del espacio. Por ejemplo, un avatar puede navegar a través de un conjunto de habitaciones conectadas en un espacio virtual limitado.

Todos los habitantes de un CVE satisfacen la propiedad de permanecer siempre en un único lugar, es decir, no pueden estar en dos o más habitaciones distintas al mismo tiempo, ni tampoco tener dos o más representaciones en un mismo espacio. Es una de las propiedades del mundo real que se respeta en los ambientes virtuales cooperativos.

Hasta este punto se presentaron las características básicas de un CVE. Dado que el comportamiento subjetivo se define en un ambiente orientado a objetos, se debe restringir la aplicación de tal concepto a ambientes virtuales cooperativos orientados a objetos. A continuación, se extenderá la definición anterior de CVE para tal propósito como así también se desarrollarán ciertas características particulares de los CVEs orientados a objetos.

AMBIENTE VIRTUAL COOPERATIVO ORIENTADO A OBJETOS

Un CVE es orientado a objetos (OOCVE) cuando se utilizó los principios, técnicas y herramientas del paradigma orientado a objetos para desarrollarlo.

Básicamente, un OOCVE puede considerase como un ambiente en el cual existe uno o varios objetos que modelan un espacio físico habitado por diversos otros objetos, y cuyo medio de interacción es el envío de mensajes.

Estructura de los habitantes

Todos los OOCVEs tienen un modelo en objetos asociado. Los MOOs por ejemplo, están diseñados con objetos: cualquier cosa en un MOO es un objeto con una única identidad; es posible cambiar el mundo en un MOO creando y adaptando objetos, los cuales continuarán existiendo en el mismo aún cuando no se esté conectado a dicho ambiente.

Al igual que en un ambiente orientado objetos, los usuarios de un MOO pueden crear objetos a partir de otros, es decir, existen objetos “genéricos” a partir de los cuales se pueden construir nuevos objetos, heredando las características y comportamiento de los primeros. Todos los objetos en un MOO se crean heredando de otros objetos existentes en el ambiente, de modo que cada uno de éstos tiene un objeto “padre” del cual hereda. Por lo general, el objeto clase raíz de un MOO es “object”, el cual provee a todos los objetos del ambiente con un cierto comportamiento común, tal como la habilidad de tener un nombre y una descripción.

Dado que los MOOs son ambientes basados en texto, la única manera de interactuar con dichos ambientes es mediante comandos textuales. La mayoría de los MOOs utilizan una sintaxis común para su manipulación, a continuación se verán algunos pasos básicos necesarios para crear, personalizar e interactuar con objetos en un MOO.

Creando nuevos objetos en un MOO

Para crear un nuevo objeto en un MOO, debe usarse el comando:

@create <nombre o número del objeto padre> **named** “<nombre del nuevo objeto>”

Por ejemplo, para crear un objeto alumno llamado “Alumno”, se podría tipear:

@create Persona **named** “Alumno”

Asumiendo que existe el objeto clase Persona en el MOO. De esta manera, un nuevo objeto “Alumno” cobra existencia en el MOO, cuyo padre será el objeto Persona.

Crear una nueva habitación es otra forma de creación, pero se utiliza otro comando. De la forma más sencilla, es posible crear una habitación de la siguiente manera:

@dig <nombre de la nueva habitación>

Por ejemplo, para crear una habitación llamada “Auditorio”, se podría tipear:

@dig Auditorio

De esta forma se crea una nueva habitación desconectada del resto y la clase padre se asume que es una habitación genérica.

Una manera de crear una habitación conectada a otras es la siguiente:

@dig <especificación de la/s puerta/s> **to** <nombre o número de la habitación>

“especificación de la/s puerta/s” es el o los nombres que se asignarán a las puertas de la nueva habitación.

Por ejemplo, para crear una habitación con una puerta de entrada y salida a la misma, se podría tipear:

@dig entrada **|** salida **to** Auditorio

De este modo se crea una nueva habitación llamada “Auditorio”. Además, se crea una puerta denominada “entrada” desde la habitación actual a la recientemente creada (“Auditorio”), y una puerta desde el “Auditorio” hacia la habitación actual denominada “salida”.

Personalizando objetos en un MOO

Nuevas propiedades en objetos

A un objeto recientemente creado se le puede especificar nuevas propiedades, adicionales a las que hereda de su clase padre. La forma de lograr esto es mediante el comando “property”, de la siguiente manera:

@property <objeto>**.**<nombre_propiedad>

@property <objeto>**.**<nombre_propiedad> <valor_inicial>

Este comando agrega una nueva propiedad llamada “nombre_propiedad” a “objeto”. Por ejemplo, para agregar la propiedad “numerodealumno” a la clase Alumno, podría tipearse:

@property Alumno**.**numerodealumno

Nuevos mensajes en objetos

La forma de especificar nuevos mensajes en una clase es mediante el comando “verb”:

@verb <objeto>**:**<nombre_mensaje>

Este comando agrega un nuevo mensaje o “verbo” llamado “nombre_mensaje” a “objeto”, lo cual significa que en adelante dicho objeto entenderá tal mensaje. Por ejemplo, para especificar el mensaje “incrementarMateriasFinalizadas” a la clase Alumno, podría tipearse:

@verb Alumno**:** incrementarMateriasFinalizadas

La forma de escribir el código de un mensaje en un objeto, es a través del comando “program”:

@program <objeto>**:**<nombre_mensaje>

El comando “program” permite ingresar código para un mensaje y compilarlo, reemplazando cualquier código existente del mensaje en cuestión. Cuando se tipea dicho comando, todo lo que se ingrese por teclado se asume que forma parte del código del mensaje en edición. Por ejemplo, asumamos que la clase Alumno define una propiedad “materias_finalizadas”, se podría definir el código del mensaje “incrementarMateriasFinalizadas” de la siguiente manera:

@program Alumno**:** incrementarMateriasFinalizadas

This.materias_finalizadas = this.materias_finalizadas + 1

En este ejemplo, “this” hace referencia al objeto en el cual se ejecutará dicho mensaje.

Interactuando con objetos en un MOO

La forma de enviar un mensaje a un objeto en un MOO es sencilla, simplemente se debe conocer el nombre o número del objeto con el cual se desea interactuar y el mensaje que se le desea enviar. Por ejemplo, si se desea enviar el mensaje “incrementarMateriasFinalizadas” al objeto Alumno, simplemente se debe tipear:

@incrementarMateriasFinalizadas Alumno

Responsabilidades de los habitantes

Cualquiera sea su tipo, las actividades que puede realizar cada habitante de un OOCVE se efectúan en ocasiones gracias a la colaboración de otros habitantes. Cada habitante puede ofrecer un cierto servicio que contribuye al desarrollo de algún otro servicio, bajo la responsabilidad de otro habitante.

Debido a que el único medio de comunicación es el envío de mensajes, una actividad que efectúa un objeto virtual se especifica con uno o varios mensajes, donde uno de estos mensajes dará inicio al desarrollo de la actividad en el mismo. Posiblemente, para el cometido de la actividad, el objeto virtual deberá enviar mensajes a otros habitantes cuando requiera de ellos algún servicio.

A continuación, se comenzará con el desarrollo del comportamiento subjetivo en los OOCVEs. En un principio se mostrará un caso de estudio donde es factible la introducción de comportamiento subjetivo en los objetos que habitan un MOO, siguiendo con el análisis del comportamiento subjetivo en los OOCVEs en general.

COMPORTAMIENTO SUBJETIVO EN UN OOCVE

Hasta este punto, se introdujo a los OOCVEs con sus características esenciales para la aplicación del comportamiento subjetivo en tales dominios. Es importante destacar que se analizará la introducción de comportamiento subjetivo sólo en los objetos virtuales reactivos, es decir, en aquellos objetos que responden a estímulos (envío de mensajes) de otros habitantes. No se considerará el comportamiento subjetivo en los objetos virtuales activos ni en los participantes, ya que el comportamiento de ambos tipos de habitantes es programado (para el caso de los objetos activos) o depende de los usuarios (en el caso de los participantes). Tampoco se tendrán en cuenta los objetos virtuales pasivos, dado que no poseen comportamiento.

Los OOCVEs son ambientes ideales para la introducción de comportamiento subjetivo en sus habitantes, dado que en la mayoría de ellos la comunicación o interacción es un medio fundamental para cumplir con sus objetivos, y las situaciones que se establecen en tales interacciones son factores determinantes para las reacciones ante los estímulos recibidos por los distintos habitantes.

Posteriormente, se comenzará con el desarrollo de un caso de estudio en donde el comportamiento subjetivo tiene un papel importante.

Motivación

El conjunto de mensajes o requerimientos que puede satisfacer un objeto virtual forma su comportamiento; así mismo, cada requerimiento representa una determinada actividad que puede realizar, y se define a través de un mensaje. Siempre que un habitante necesite la colaboración de un objeto virtual en una cierta actividad, simplemente debe enviarle el mensaje que la define, y éste último responderá de la forma establecida en la implementación del mensaje en cuestión.

A simple vista, se podría suponer que estas propiedades de interacción de los objetos virtuales son suficientes para representar con facilidad objetos del mundo real u objetos irreales. Pero en ciertas ocasiones, es factible un mecanismo de comportamiento aún más complejo, el cual permita que las distintas situaciones en la que se encuentra un cierto objeto en su ambiente puedan influenciarlo de manera tal que provoquen que dicho objeto brinde diferentes respuestas ante un mismo mensaje, es decir, que sea capaz de reaccionar de diversas formas cuando se le solicita que satisfaga un determinado servicio.

Caso de estudio

Como se mencionó anteriormente, los MOOs son una variedad de realidad virtual basada en texto. Algunos de ellos están diseñados para facilitar la enseñanza a través de medios informáticos, como por ejemplo *Collage Town (http://www.bvu.edu/ctown)* o *Diversity University (http://www.du.org).* Esta clase de MOO, se basa principalmente en actividades de aprendizaje sincrónicas, aunque también considera a otro tipo de actividad asincrónica: envío y recepción de mensajes. En su mayoría, cada uno de estos MOOs tiene su propio sistema interno de mensajes conocido como moomail, el cual permite la comunicación asincrónica entre los estudiantes y maestros.

Considere un MOO diseñado para la enseñanza, tal como Collage Town. En el mismo se desea crear un sistema de mensajes de correo, en el cual exista un aula con buzones de mensajes. Cada buzón debe pertenece a un determinado participante (un alumno o un profesor), quien tendrá acceso exclusivo al contenido de dicho buzón. Los buzones serán cargados de mensajes por un participante “mensajero”, el cual es responsable de llenar de mensajes cada buzón.

A continuación se presentará la creación del sistema de correo en un MOO utilizando los comandos anteriormente listados.

Aula de buzones

El aula de buzones será una habitación la cual contendrá los buzones de mensajes. Se le especificará una puerta de entrada y salida, a través de la cual los participantes (maestros y alumnos) podrán ingresar. Para esto, deberá tipearse:

@dig entrada **|** salida **to** aulaDeBuzones

Con lo cual, se habrá creado una nueva habitación denominada “aulaDeBuzones” con una puerta de entrada y salida que permitirá que cualquier habitante ingrese y salga de la misma.

Buzones

Los buzones pueden considerarse contenedores de objetos (mensajes), por lo cual, se definirán los buzones como habitaciones, de la siguiente manera:

@dig entrada **|** salida **to** Buzon1
....
....
@dig entrada **|** salida **to** BuzónN

Cada buzón debe ser capaz de mostrar su contenido, es decir, mostrar los mensajes que posee en su interior. Para esto, se definirá el mensaje "mostrarMensajes" en una clase genérica desde la cual heredan todos los buzones de la siguiente manera:

@verb Buzon**:**mostrarMensajes

Al enviar el mensaje "mostrarMensajes" a un objeto buzón, el mismo deberá mostrar su contenido, y esta acción estará especificada en el código asociado a dicho mensaje.

Para introducir los objetos buzones al aula de buzones, considere que la clase genérica desde la cual hereda "aulaDeBuzones" define un mensaje, tal como "agregarObjeto" el cual permite agregar los objetos buzones en la misma:

@agregarObjeto aulaDeBuzones Buzon1
....
....
@agregarObjeto aulaDeBuzones BuzonN

Mensajes

Los mensajes pueden considerarse objetos que contienen un encabezado y contenido asociado. Se podría definir una clase genérica de mensaje la cual contenga dichas propiedades y defina algunos mensajes para el acceso a los valores de las mismas:

@create Object **named** "Mensaje"

Se crea la clase genérica de los mensajes.

@property Mensaje**.**encabezado

@property Mensaje**.**contenido

Se determinan las propiedades de todos los mensajes.

@verb Mensaje**:** mostraEncabezado

@verb Mensaje**:** mostraContenido

Se determinan los mensajes que entienden todos los mensajes.

La creación de cada uno de los mensajes podría efectuarse de la siguiente manera:

@create Mensaje **named** "Mensaje1"

Para ingresar los mensajes a un buzón, considere que cada buzón entiende el mensaje:

@agregarMensaje BuzonX mensajeY

Una vez definido el sistema de mensajes de correo, es posible utilizarlo en el MOO.

Una posible interacción en el aula de buzones podría originarse cuando un alumno solicita ver el contenido de un determinado objeto buzón, enviándole el mensaje *mostrarMensajes*. La reacción más pensada a tal requerimiento es que el buzón responda mostrando sus mensajes, pero no siempre debe ser de esta manera, dado que el alumno podría no ser el propietario de buzón. Por lo cual, es ideal que cada buzón sea capaz de reaccionar adecuadamente de acuerdo a quien le solicite su contenido: si se trata del alumno propietario, muestra su contenido, si se trata de cualquier otro participante, no muestra su contenido.

Considere el caso en que un alumno solicita sus mensajes a su buzón y el mismo no tiene mensajes en su interior. En este caso, sería ideal que el buzón reaccione notificando a su dueño de tal hecho, en lugar de abrir su casilla y mostrar que no tiene mensajes en su interior.

Por otra parte, imagínese el caso de que un alumno solicita los mensajes a su buzón, y dentro del aula se encuentra el participante "mensajero", el cual se encarga del cargado de mensajes en los buzones. En este caso, sería bueno que el buzón además de mostrar su contenido (si tiene mensajes), notifique de alguna manera a su dueño que el mensajero podría tener más mensajes para él.

Podrían surgir muchas otras situaciones que conduzcan a distintas reacciones o respuestas de un buzón cuando recibe el mensaje *mostrarMensajes*; por lo cual, sería deseable extender la forma en que se comporta ante tal requerimiento, para que brinde una respuesta adecuada de acuerdo a la situación que se establece en su ambiente.

Esta clase de problema, donde un objeto debe ser capaz de responder adecuadamente ante un mismo estímulo dependiendo de la situación de su entorno, es el que intenta resolver el comportamiento subjetivo de manera natural. Sin comportamiento subjetivo, es posible su resolución, pero es probable que se requiera un mayor esfuerzo en lograrla, y quizá no sea tan clara y directa como la que se logra utilizando subjetividad en el comportamiento de los objetos.

Previamente se definió comportamiento subjetivo en el paradigma orientado a objetos como la habilidad en un objeto de poder brindar ante un mismo mensaje diferentes respuestas, dependiendo de la manera en que ciertos factores internos y/o externos lo afectan en ese momento. Se extiende esta definición en un OOCVE de la siguiente manera.

Definición de comportamiento subjetivo en un OOCVE

Un objeto virtual en un OOCVE tiene comportamiento subjetivo si es capaz de brindar ante un mismo estímulo diferentes respuestas, dependiendo de la situación en la que se encuentre inmerso.

Los conceptos del paradigma orientado a objetos que fueron extendidos por la incorporación del comportamiento subjetivo también tienen validez en los OOCVEs, admitiendo además que objeto virtual subjetivo aludirá a un objeto virtual que tiene la capacidad de comportarse subjetivamente.

ENTORNO DEL COMPORTAMIENTO SUBJETIVO EN UN OOCVE

Como se vio anteriormente, en los ambientes virtuales surgen frecuentemente situaciones en las cuales se espera que la respuesta asociada a un determinado mensaje en

un objeto virtual no sea siempre la misma, es decir, que dicho objeto sea capaz de dar una respuesta adecuada, dependiendo de la situación que se establezca en su entorno, en el instante que recibe dicho mensaje.

Para definir el concepto de situación de influencia, es necesario establecer en primer lugar cuáles son los elementos que la determinan. En los ambientes subjetivos orientados a objetos, se vio que los objetos que conformaban el ECS eran los que determinaban los factores de influencia (o fuerzas). De manera similar, en los ambientes virtuales orientados a objetos se extiende el concepto de ECS, y sus integrantes son los que determinan situaciones de influencia.

Cuando se desarrolló el comportamiento subjetivo tradicional[9] se definió el entorno del comportamiento subjetivo (ECS) como el conjunto de objetos que cumple un determinado rol en relación con un objeto subjetivo y que alguna de sus *características del objeto*[10] puede influir en el comportamiento del mismo. Dichos objetos eran:

- El **emisor** de un mensaje,
- Los **objetos del contexto** de aplicación del objeto subjetivo,
- (si existen) los **colaboradores internos** del objeto subjetivo,
- (si existen) los **colaboradores externos** del objeto subjetivo.

En un OOCVE, también se define para cada objeto virtual su propio ECS, dado que, cuando un objeto virtual recibe un mensaje, existen habitantes con roles similares a los anteriormente listados: emisor, colaboradores internos y colaboradores externos (cuando el mensaje lleva ciertos argumentos). En el caso particular de los objetos del contexto, se considerarán a aquellos que se encuentren en el entorno del objeto virtual subjetivo, es decir, los objetos relativamente próximos los cuales pueden ser percibidos por dicho objeto virtual. Para evitar confusiones, se denominarán objetos del entorno y se ampliará su definición en la sección que define la fuerza asociada a dichos objetos.

Adicionalmente, en los OOCVEs es posible considerar a un objeto virtual con un rol distinto de aquellos objetos que conforman el ECS descrito hasta el momento, y que eventualmente alguna de sus características puede influir en el comportamiento de un objeto virtual. Dado que cada habitante está contenido en algún espacio dentro de un ambiente virtual (una habitación, un colectivo, un monedero, etc.), un objeto virtual contenedor (un objeto que modela un espacio) puede considerarse como un factor más de influencia. En la sección que define las fuerzas en un OOCVE se mostrará su relevancia en el comportamiento subjetivo, por lo pronto, se incluirá en la definición del entorno del comportamiento subjetivo dentro de estos ambientes.

A continuación, se extiende la definición de ECS para los OOCVEs.

Definición de ECS

El ECS es un conjunto de habitantes, donde cada uno cumple un rol específico con relación a un objeto virtual subjetivo, y el cual podría afectar su comportamiento por poseer una determinada *característica del objeto*. Dichos habitantes son considerados al momento de evaluarse un mensaje subjetivo que recibe un objeto virtual, y son:

- El habitante que le envía el mensaje (**emisor**),
- Habitantes que son percibidos por el objeto virtual subjetivo (**objetos del entorno**),
- Objetos que forman parte de su estructura (**colaboradores internos**),
- (si existen) objetos virtuales que forman parte del mensaje (**colaboradores externos**),
- El objeto virtual contenedor del objeto virtual subjetivo (**contenedor**).

De esta manera, quedan establecidos los habitantes que forman el ECS de un objeto virtual subjetivo, y cualquier *característica del objeto* de ellos puede ser relevante en la forma de comportarse de dicho objeto virtual.

---

[9] El comportamiento subjetivo tradicional se refiere al comportamiento subjetivo definido en el paradigma orientado a objetos.

[10] Las características definidas como *Identidad, Protocolo* y *Condición.*

LAS FUERZAS

En la definición del comportamiento subjetivo en un ambiento orientado a objetos tradicional, se estableció que, para incluir subjetividad en la evaluación de un mensaje, es necesario fijar previamente, en la definición del mismo, cuáles son los factores que influirán, y a qué respuesta subjetiva conducirán.

A su vez, se determinó que un factor de influencia en el comportamiento de un objeto subjetivo puede ser:

- Una *característica del objeto* que se pretende que posea alguno de los objetos del ECS cuando se evalúe un mensaje subjetivo (factor externo), o bien
- Una determinada condición que debe establecerse en los objetos que conoce internamente el objeto subjetivo cuando se evalúe dicho mensaje (factor interno).

A cada uno de estos factores se le atribuyó una fuerza, ya que ciertas veces tiene el poder de variar el comportamiento o respuesta de un objeto subjetivo ante un mismo mensaje. En un OOCVE, se extiende el concepto de factor de influencia de la siguiente manera:

- Una *característica del objeto* que se pretende que posea alguno de los habitantes que forman el ECS cuando se evalúe un mensaje subjetivo (factor externo), o bien
- Una determinada condición que debe establecerse en los objetos que conoce internamente el objeto virtual subjetivo cuando se evalúe dicho mensaje (factor interno).

Asimismo, se redefine el concepto de fuerza de la siguiente manera.

Definición de Fuerza

Una fuerza es un factor de influencia que tiene el poder de participar en la elección de una respuesta subjetiva, cuando un objeto virtual subjetivo recibe un mensaje subjetivo.

Como puede verse, la mayoría de las fuerzas definidas para un ambiente orientado a objetos tradicional son aplicables en un OOCVE, dado que los elementos que determinan dichas fuerzas (los objetos que conforman el ECS) están presentes en cualquier aplicación orientada a objetos, independientemente del dominio del que se trate; por tal motivo, tales fuerzas pueden considerarse en un OOCVE, ya que se trata de una aplicación orientada a objetos. El motivo por el cual la fuerza del contexto no se considera tal cual fue definida anteriormente, se debe a que es más significativo en un OOCVE tener en cuenta como factor de influencia alguna *característica del objeto* de algún habitante en el entorno de un objeto subjetivo, en lugar de alguna *característica del objeto* de algún objeto del contexto de aplicación.

De este modo, cada factor externo de influencia estará asociado a uno de los objetos del ECS, y de acuerdo a cuál sea su rol con relación al objeto virtual subjetivo, quedará representado por la *fuerza del emisor*, la *fuerza del entorno*, la *fuerza del colaborador interno*, la *fuerza del colaborador externo* o la *fuerza del contenedor*; el único factor interno quedará representado por la *fuerza del estado.*

Definición de Predominio de Fuerza

De la misma manera que en un ambiente subjetivo tradicional, en la evaluación de un mensaje subjetivo dentro de un OOCVE, una o más fuerzas pueden predominar. Se dice que una fuerza:

- Externa (emisor, entorno, colaborador interno, colaborador externo o contenedor) predomina cuando la *característica del objeto* pretendida para su habitante asociado dentro

del ECS (establecida en la definición del mensaje), coincide con la característica que tiene dicho habitante en el momento de la evaluación del mensaje;
- Interna (estado) predomina cuando la condición pretendida en los objetos que conoce internamente el objeto virtual subjetivo (establecida en la definición del mensaje), se concreta en el momento de la evaluación del mensaje.

Por lo tanto, cuando un objeto virtual recibe un mensaje subjetivo, se analizan las fuerzas que fueron configuradas como influyentes en el comportamiento del mismo. Si alguna de estas fuerzas predomina, conducirá en algún sentido específico el camino para la elección de la respuesta correcta.

A continuación, se desarrollarán las fuerzas en un OOCVE.

LA FUERZA DEL COLABORADOR

Los objetos que conforman la estructura interna de un habitante forman el conjunto de colaboradores internos. Los objetos virtuales que conoce un habitante mediante conocimiento externo (es decir, aquellos que son parámetros de algún estímulo) forman el conjunto de colaboradores externos. Al igual que en el comportamiento subjetivo para un ambiente orientado a objetos tradicional, un colaborador (interno o externo) puede afectar el comportamiento de un objeto virtual cuando recibe un mensaje, por alguna de las siguientes características:

- Su identidad
- Su protocolo
- Cierta condición que cumple

Ejemplos del buzón de mensajes

Considere que existe la posibilidad de que cualquier habitante solicite ver los mensajes de un buzón, siempre y cuando tenga una clave o código de acceso adecuado. De esta manera, cada buzón debería tener asociado un código, y los habitantes que solicitan ver sus mensajes, deben conocer dicho código. Podría agregarse una propiedad más a cada buzón que represente el código de acceso, con un valor inicial:

@property BuzonX.codigoDeAcceso 45156

Y en la clase genérica “Buzon” se especifica el siguiente mensaje:

@verb Buzon:mostrarMensajesCodigo unCodigo

Donde el parámetro “*unCodigo*” (colaborador externo) debe ser el que tiene asociado el buzón para que el mismo muestre satisfactoriamente los mensajes. Por lo tanto, si un habitante le envía el mensaje “*mostrarMensajesCodigo”* a un buzón, el mismo tendrá dos posibles formas de comportarse, de acuerdo a si el código es:

- El mismo que tiene asociado → entonces el buzón muestra los mensajes,
- Distinto al que tiene asociado → entonces el buzón no muestra los mensajes.

Así, puede observarse que un colaborador externo de un mensaje enviado a un objeto virtual, puede tener influencia en la forma de responder de dicho objeto al momento de recibir ese mensaje.

Entonces... ¿Cual es la Fuerza del Colaborador?

La *fuerza del colaborador* es uno de los factores externos de influencia. Se basa en uno de los colaboradores (interno o externo) de un objeto virtual, y tiene en cuenta una de las *características del objeto* de dicho colaborador. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL ESTADO

Dado que los objetos virtuales son en definitiva objetos, el estado de un objeto virtual se define de la misma manera que para un objeto común: como el conjunto de colaboradores internos que posee en un determinado instante durante su tiempo de vida. Ya que los colaboradores internos de un objeto virtual forman un conjunto variante, es posible que en algún momento, se establezcan ciertas características en uno o varios de dichos colaboradores que simbolice un determinado *estado lógico* para el objeto virtual. La presencia de un determinado estado lógico puede influenciar el comportamiento del objeto, afectando la elección de la respuesta a brindar.

Ejemplo del buzón de mensajes

Considere el caso en que un habitante solicita ver el contenido de su correspondiente buzón de mensajes, enviándole el mensaje *mostrarMensajes.* El buzón podría tener o no mensajes en su interior, por lo cual, sería ideal que:

- En el caso de que el buzón tenga mensajes → muestre a su dueño los mensajes,
- En el caso de que el buzón no tenga mensajes → notifique de alguna forma a su dueño que no contiene mensajes en su interior.

De esta manera, el estado de un objeto virtual puede ser importante en el momento de dar una respuesta adecuada cuando dicho objeto recibe un mensaje.

Entonces... ¿Cual es la Fuerza del Estado?

La *fuerza del estado* es el factor interno de influencia que se basa en uno de los posibles estados lógicos en el que puede encontrarse el objeto virtual subjetivo. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si se establece o no tal estado lógico; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL EMISOR

Un habitante que invoca un mensaje en un objeto virtual cumple el rol de *emisor* del mensaje. El mismo puede afectar el comportamiento del respectivo objeto virtual receptor, por alguna de sus siguientes características:

- Su identidad
- Su protocolo
- Cierta condición que cumple

Ejemplo del buzón de mensajes

Dado que los buzones están ubicados en un aula pública, puede suceder que cualquier habitante solicite ver los mensajes contenidos en cualquier buzón. Asuma que un habitante le envía el mensaje "*mostrarMensajes"* a un buzón. El buzón debe mostrar sus mensajes sólo a su dueño. Por lo tanto, la respuesta de buzón ante dicho mensaje debe ser:

- Si el habitante que solicita ver los mensajes es su dueño → muestra los mensajes,
- Si el habitante que solicita ver los mensajes no es su dueño → no muestra los mensajes, notificando quizás el error.

Entonces, el emisor de un mensaje puede ser otro factor de influencia en el comportamiento de un objeto virtual.

Entonces... ¿Cual es la Fuerza del Emisor?

La *fuerza del emisor* es otro de los factores externos de influencia. Se basa en el habitante emisor de un mensaje subjetivo, y tiene en cuenta una de las *características del objeto* de dicho emisor. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

Las tres fuerzas anteriores se definen de manera muy similar que en el comportamiento subjetivo tradicional, y se incluyen en forma casi inmediata dentro del conjunto de fuerzas del comportamiento subjetivo en un OOCVE. En cambio, la fuerza del contexto no se mantiene de la forma que anteriormente fue definida, debido a que los objetos del contexto considerados en estos ambientes no son los mismos que los considerados en la primera definición. Posteriormente, se define el concepto de entorno de un objeto virtual y se redefine la fuerza del contexto como la fuerza del entorno.

LA FUERZA DEL ENTORNO

Definición de Entorno

La mayoría de los habitantes tienen la capacidad de interactuar con otros habitantes para requerir ciertos servicios de ellos. Para lograrlo, un habitante debe tener algún medio de poder determinar cuales son los habitantes "cercanos" a él, de manera de poder comunicarse con ellos para solicitarle alguna colaboración. La mayoría de los objetos virtuales (especialmente los reactivos y los activos) están provistos de un mecanismo que les permite determinar el conjunto de habitantes en su entorno, aquellos que "virtualmente" perciben y, por ende, que pueden enviarles mensajes. Si el espacio virtual donde habita un objeto virtual es limitado, podría considerarse que los habitantes percibidos son aquellos que se encuentra en la misma habitación; en cambio, si el espacio es ilimitado seguramente existirá otro mecanismo más complejo para determinar cuales son los habitantes en su entorno.

Independientemente del mecanismo de percepción utilizado, se concluye que el entorno de un objeto virtual está formado por el conjunto de habitantes, normalmente variable, que percibe en un determinado momento, y por lo cual, puede comunicarse mediante el envío de mensajes.

Características de un objeto del Entorno que pueden influir en el comportamiento

Un habitante que forma parte del entorno de un objeto virtual puede influir en la respuesta a un mensaje enviado a dicho objeto, por alguna de sus siguientes características:

- Su identidad
- Su protocolo
- Cierta condición que cumple

Ejemplo del buzón de mensajes

Para que los buzones se recarguen con mensajes, es necesario que algún objeto se encargue de esta tarea. Asuma que existe un participante "*mensajero*" que tiene esta responsabilidad, es decir, cargar los buzones cuando existen mensajes para sus respectivos dueños. Dado que la presencia del objeto mensajero no es constante en el aula de buzones (solo entra al aula cuando va a cargar uno o más buzones), sería ideal que, cuando un habitante está solicitando ver sus mensajes a su buzón:

- Si el participante mensajero está dentro del aula → el buzón muestra sus mensajes notificando además a su dueño de que el objeto mensajero podría tener más mensajes para él,
- Si el participante mensajero no está dentro del aula → sólo muestra sus mensajes.

Así, es posible que un habitante del entorno de un objeto virtual afecte su comportamiento ante un estímulo.

Entonces... ¿Cual es la Fuerza del Entorno?

La *fuerza del entorno* es un factor externo más de influencia. Se basa en uno de los habitantes que forman el entorno de un objeto virtual subjetivo, y tiene en cuenta una de sus *características del objeto*. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

LA FUERZA DEL CONTENEDOR

Definición de Contenedor

Indistintamente del tipo de ambiente virtual del que se trate (limitado o ilimitado), todos los habitantes de un OOCVE se alojan en un determinado espacio dentro del ambiente. El contenedor de un habitante queda representado por el espacio donde reside, es un objeto virtual con la capacidad de contenerlo.

Características del Contenedor que pueden influir en el comportamiento

Un objeto virtual que contiene a otro objeto virtual puede influir en la respuesta de un mensaje enviado a este último, por alguna de sus siguientes características:

- Su identidad

- Su protocolo
- Cierta condición que cumple

Ejemplo del buzón de mensajes

Considere que el aula donde se encuentran los buzones tiene un horario de consulta el cual debe ser respetado por los alumnos. Para esto, la clase aulaDeBuzones define la siguiente propiedad:

@property aulaDeBuzones.horarioDeConsulta

Los habitantes alumnos sólo pueden verificar sus mensajes cuando el aula se encuentra en horario de consulta de mensajes, más allá de que el aula siempre se encuentre abierta, accesible a cualquier habitante. Por lo cual, cuando un habitante alumno le envía el mensaje *mostrarMensajes* a su buzón, puede ocurrir que:

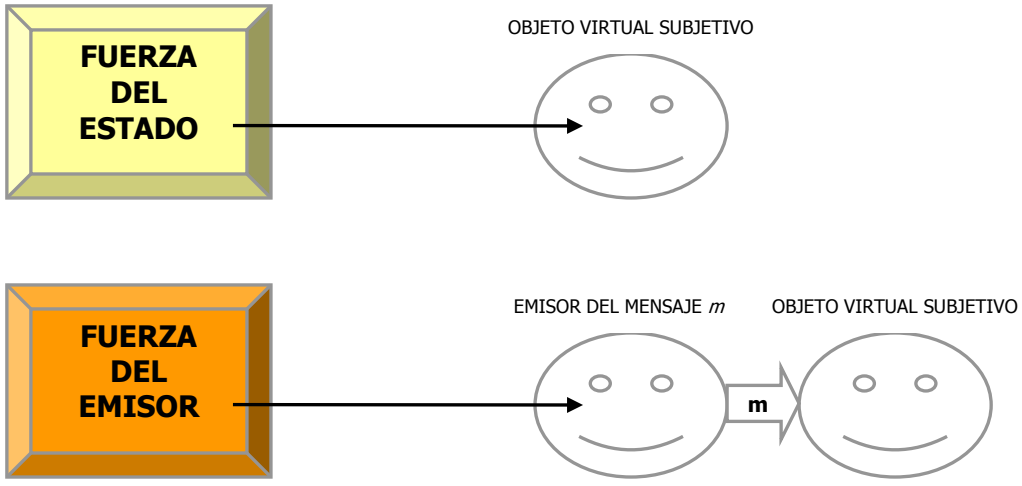
- Si el aula está en horario de consulta de mensajes → el buzón muestre sus mensajes sin inconvenientes,
- Si el aula no está en horario de consulta de mensajes → el buzón no muestre sus mensajes y notifique al alumno cuál es el horario de consulta.

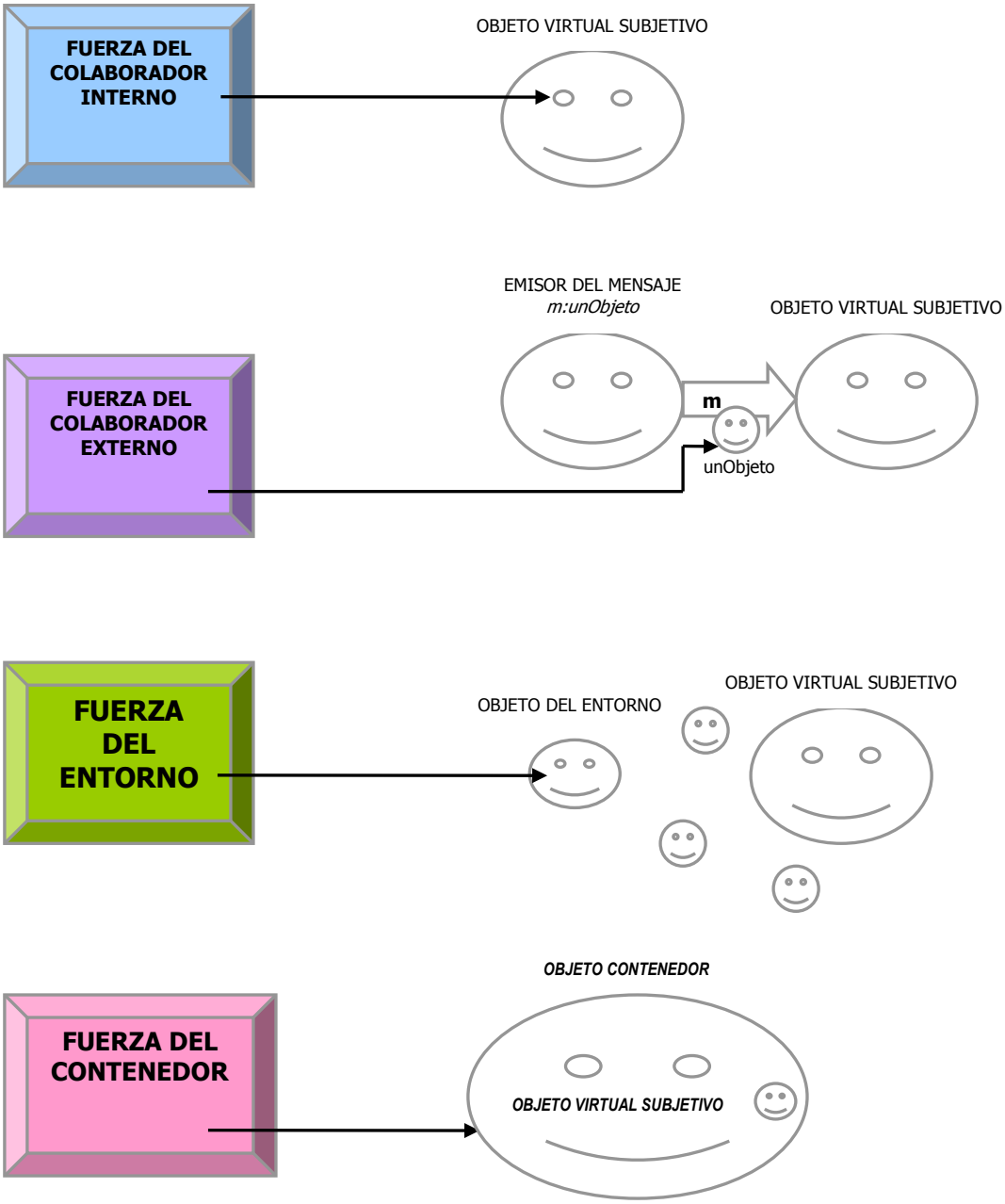
De este modo, puede verse que el objeto contenedor de un objeto virtual puede influenciar en la forma de responder ante un estímulo recibido por dicho objeto.

Entonces... ¿Cual es la Fuerza del Contenedor?

La *fuerza del contenedor* es un factor externo de influencia. Se basa en el objeto virtual que contiene a un objeto virtual subjetivo, y tiene en cuenta una de las *características del objeto* de dicho objeto contenedor. Al momento de evaluarse el mensaje subjetivo donde está definida la fuerza, la misma determina si predomina o no tal característica; basándose en este resultado, conduce a la elección de la respuesta subjetiva correcta.

El siguiente diagrama muestra en forma resumida todos los tipos de fuerzas que forman parte del comportamiento subjetivo en un OOCVEs y los objetos del ECS que cada una de éstas tiene asociados.





Hasta este punto se definieron las fuerzas que se consideran relevantes en un OOCVE. Posteriormente se explicará la organización de un mensaje subjetivo, junto con el concepto de situación de influencia y lógica de evaluación de un mensaje subjetivo en un OOCVE.

ESTRUCTURA DE UN MENSAJE SUBJETIVO EN OOCVE

La estructura de un mensaje subjetivo en un OOCVE es igual a la definida para un ambiente tradicional orientado a objetos: el árbol de fuerzas, el cual establece el orden de evaluación de las fuerzas involucradas en un mensaje subjetivo. El hecho de emplear la misma

estructura se debe a que el árbol de fuerzas puede configurarse para satisfacer cualquier caso de influencia de fuerzas en el comportamiento de algún objeto virtual. No debe olvidarse que los OOCVE no dejan de ser aplicaciones en ambientes orientados a objetos, por lo cual, el diseño del comportamiento subjetivo puede utilizarse sin inconvenientes en cualquier aplicación orientada a objetos.

Por lo tanto, el árbol de fuerzas para los mensajes subjetivos en un OOCVE estará compuesto por determinantes de implementación y determinantes de fuerza, con la misma estructura y funcionalidad que los definidos anteriormente.

En el caso de los determinantes de fuerza, se vio que los mismos tienen asociados un objeto *tipo de fuerza*, el cual es responsable de capturar un objeto dependiendo de qué tipo de fuerza represente. Dado que en un OOCVE el conjunto de fuerzas varia en cierta medida del definido para un ambiente tradicional orientado a objetos, se reconstruye el cuadro explicativo de los tipos de fuerzas:

TIPO DE FUERZA	ACCION DE CAPTURA
Fuerza del emisor	Captura al habitante emisor de un mensaje subjetivo.
Fuerza del colaborador interno	Captura a un colaborador interno de un objeto virtual subjetivo.
Fuerza del colaborador externo	Captura a un colaborador externo de un objeto virtual subjetivo, el cual forma parte del conjunto de parámetros de un mensaje subjetivo enviado a dicho objeto.
Fuerza del entorno	Captura a un habitante del entorno de un objeto virtual subjetivo.
Fuerza del estado	Captura al objeto virtual receptor de un mensaje subjetivo.
Fuerza del contenedor	Captura al objeto virtual contenedor de un objeto virtual subjetivo.

SITUACIONES DE INFLUENCIA

Como se describió en la definición del comportamiento subjetivo en un OOCVE, un objeto virtual puede variar su comportamiento cuando recibe un mensaje, dependiendo de la situación en la que se encuentra en ese momento.

Imagínese la siguiente situación. En el aula virtual donde se encuentran los buzones de mensajes, hay un habitante profesor y un alumno. Dicho alumno le solicita a su respectivo buzón ver su correspondencia (es decir, le envía el mensaje *mostrarMensajes*). Pero en ese instante, el aula no se encuentra en horario de consulta de mensajes para los alumnos. Dado que en el interior del aula se encuentra un profesor, podría notificársele al mismo que un alumno intenta ver mensajes fuera de horario. En esta situación, existen varias fuerzas cuyo predominio en conjunto determina esa respuesta específica del buzón. Esto significa que, cuando un buzón recibe el mensaje *mostrarMensajes*:

- Si el emisor es un alumno
- Si el aula (contenedor) se encuentra fuera de horario de consulta, y
- Si en el entorno del buzón se encuentra un profesor

⇒ El buzón notifica al profesor que un alumno intenta ver sus mensajes fuera de horario.

De este modo, se observa que el comportamiento de un objeto virtual ante un mensaje podría variar por la influencia simultanea de dos o más fuerzas, como en el caso del buzón, donde la influencia de tres fuerzas llevan a un comportamiento especifico por parte del mismo.

Por lo tanto, una situación de influencia puede estar determinada por una o más fuerzas que simultáneamente predominan.

En el caso del comportamiento subjetivo para un ambiente tradicional orientado a objetos, también podría tener sentido la existencia de situaciones de influencia de este tipo, es decir, situaciones en donde dos o más fuerzas predominen y que dicho predominio en conjunto conduzca en algún sentido la búsqueda de la respuesta subjetiva. No obstante, el árbol de fuerzas es una estructura lo suficientemente flexible y satisface sin problemas estos casos. Considere el caso en que se desea que el predominio (simultaneo) de tres fuerzas conduzca a una determinada respuesta por parte de un objeto cuando recibe un determinado mensaje. La forma de lograr dicha configuración, se consigue anidando tales fuerzas, es decir, haciendo que una de ellas determine la evaluación de otra cuando predomina; la última de las fuerza anidadas debe conducir, cuando predomina, a la evaluación de la respuesta subjetiva. De esta manera, si las tres fuerzas predominan, equivale conceptualmente al predominio de una situación determinada por dichas tres fuerzas.

De este modo, el concepto de situación de influencia es una manera de referirse a casos en donde el predominio de dos o más fuerzas conduce en algún sentido la búsqueda de la respuesta subjetiva. Una situación de influencia puede ser representada sin problema con un árbol de fuerzas.

LÓGICA DE EVALUACIÓN DE UN MENSAJE SUBJETIVO

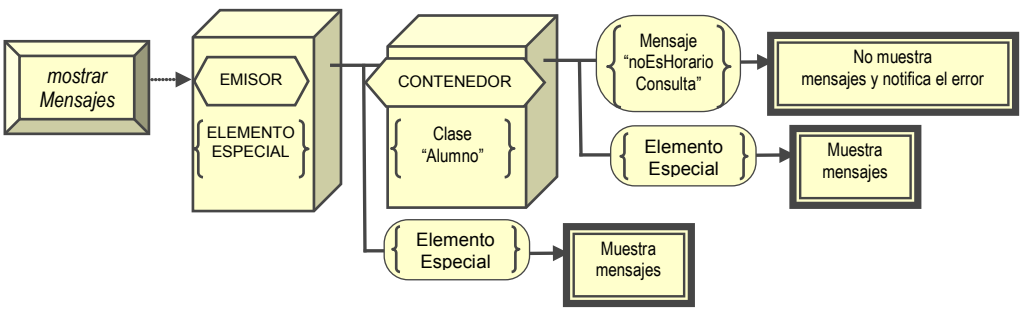
Al igual que en el comportamiento subjetivo tradicional, cuando un objeto virtual recibe un mensaje subjetivo, implícitamente se evalúa la estructura asociada al mismo para determinar cuál es la respuesta subjetiva que debe brindar. El proceso que implica esta evaluación no se modifica en un OOCVE y sigue siendo transparente al objeto virtual subjetivo, debido a que en ningún momento se requiere de su colaboración en tal proceso.

De esta forma, en la evaluación de un árbol de fuerza se producirán dos sucesos:

- El establecimiento del ECS → el medio que permite capturar a todos aquellos habitantes que pueden "forzar" de alguna forma el comportamiento de un objeto virtual subjetivo.
- La evaluación de la estructura asociada al mensaje → comenzando con la evaluación del determinante de fuerza raíz del árbol de fuerzas asociado al mensaje subjetivo, y continuando la evaluación por alguna de las ramas de dicho árbol para encontrar el método subjetivo a ejecutar.

EJEMPLO DE EVALUACIÓN DE UN ÁRBOL DE FUERZA

Para recordar la evaluación de un árbol de fuerzas, se mostrará la evaluación de un árbol asociado al mensaje *mostrarMensajes* . Se asume la siguiente configuración del mismo:




DETERMINANTE DE FUERZA


DETERMINANTE DE IMPLEMENTACION


METODO SUBJETIVO


TIPO DE FUERZA


VALOR DEL DETERMINANTE

Este árbol de fuerzas representa el caso en que la influencia simultanea de dos fuerzas (emisor y contenedor) llevan a una respuesta específica por parte del buzón. Considere el caso en que un buzón de mensajes recibe el mensaje *mostrarMensajes,* cumpliéndose lo siguiente:

- El emisor del mensaje es un alumno, y
- El aula no se encuentra en horario de consulta de mensajes,

En primer instancia se determina el ECS del buzón de mensajes. El determinante de fuerza raíz es el que conoce al tipo de fuerza emisor y el mismo predominará por tener al elemento especial como valor. Luego, solicitará a su objeto tipo de fuerza que capture al objeto correcto, que en este caso se trata del emisor del mensaje: el alumno. Así, el determinante fuerza del emisor busca al hijo que predomina. En primer lugar solicita a su único hijo determinante de fuerza la verificación de su predominio, el cual compara la clase que conserva como valor del determinante con la clase del emisor del mensaje para determinarlo. Dado que dichas clases coinciden, este determinante predomina y el determinante raíz delega la evaluación en este determinante de fuerza. De la misma manera que su padre, este determinante le pide a su tipo de fuerza que recupere el objeto adecuado del ECS, en este caso, el contenedor del buzón de mensajes: el objeto aula. Posteriormente, busca al hijo que predomina. El primero de éstos predomina, ya que su valor satisface al contenedor (es decir, la condición de que el aula no está en horario de consulta se hace verdadera). De esta manera, por tratarse de un determinante de implementación, culmina la evaluación del árbol de fuerzas y el buzón no mostrará sus mensajes y notificará el motivo.

EVALUACION DE LA INTRODUCCION DE COMPORTAMIENTO SUBJETIVO EN UN OOCVE

El comportamiento subjetivo fue modelado para ser utilizado en cualquier aplicación orientada a objetos. Por tal motivo, fue diseñado para considerar los aspectos generales a todos los objetos (las características del objeto) y los factores generales de influencia en el comportamiento de los mismos (la fuerza del emisor, del colaborador interno, del colaborador externo, del estado y del contexto). Por tal motivo, el comportamiento subjetivo puede ser utilizado en cualquier dominio de aplicación que sea modelado y implementado en un ambiente de programación orientado a objetos. Sin embargo, en algunos casos, los elementos que define pueden resultar insuficientes para una determinada aplicación orientada a objetos y tiene sentido extenderlo para que se adapte de mejor forma al dominio de aplicación que se esté modelando, como en el caso de los OOCVEs. En este tipo de aplicación, tuvo significado extender las fuerzas, debido que el dominio mismo define ciertos tipos de objetos con “roles especiales” (contenedor y objeto del entorno) cuyas características pueden afectar el comportamiento de otros objetos virtuales.

Por lo tanto, la introducción del comportamiento subjetivo puede tener una gran utilidad en ciertos dominios, siendo aún mayor si se introducen nuevas fuerzas de influencia para los

objetos que lo modelan. A medida que se comprenda mejor cómo se comportan los objetos del mundo real o irreal (como en el caso de los OOCVEs), es muy posible que aparezcan nuevas fuerzas a modelar dentro del ambiente subjetivo.

CONCLUSIONES

El comportamiento subjetivo es un área de interés más en la comunidad orientada a objetos. En sistemas de gran envergadura, es importante que ciertos objetos se comporten de diferentes maneras cuando se los utiliza en distintos contextos, por distintos clientes, y en tiempos distintos. El principal objetivo de tal "flexibilidad" en el comportamiento es la reusabilidad, ya que de esta manera se logran objetos capaces de adecuar su forma de responder a un mensaje ante distintas situaciones.

A continuación, se listan conclusiones obtenidas de la introducción del comportamiento subjetivo en el mundo de los objetos.

- Es posible modelar de manera más clara y natural objetos del mundo real, disminuyendo la distancia o "gap semántico" que hay entre estos objetos y los modelos computacionales. En el mundo real existen ciertas entidades con comportamiento complejo las cuales son capaces de varias sus respuestas ante un mismo requerimiento, dependiendo de la influencia de factores internos y/o externos. Dicho comportamiento complejo no es trivial modelar con los conceptos básicos que ofrece el paradigma orientado a objetos. El comportamiento subjetivo en los objetos trata de resolver de manera simple la programática de dicho comportamiento complejo, ya que otorga a los objetos la capacidad de comportarse subjetivamente ante un mismo mensaje.

- Se extiende el paradigma orientado a objetos, sin restringirlo, debido a que no se modifican los conceptos básicos que define y, dado que se añade una nueva capacidad de comportamiento en los objetos, aparecen nuevos conceptos propios del comportamiento subjetivo. Por lo tanto, el paradigma crece en conceptos, respetándose los conceptos más importantes:

  - **Encapsulamiento**: los objetos con comportamiento subjetivo encapsulan el mecanismo necesario para evaluar los mensajes subjetivos; en la definición de uno de estos mensajes sólo hay que especificar el comportamiento deseado del objeto, sin entrar en detalles en la mecánica de evaluación. De este modo se mantiene la transparencia en la evaluación y despacho de mensajes subjetivos y no subjetivos.

  - **Ocultamiento de Información**: los clientes de clases subjetivas desconocen el hecho de que dichas clases definen comportamiento subjetivo. El protocolo que define una clase subjetiva no revela la verdadera naturaleza de sus instancias, haciendo que el comportamiento subjetivo sea transparente para los clientes de los objetos subjetivos.

  - **Herencia**: El mecanismo de herencia (desde el punto de vista conceptual) no se preocupa por el hecho de que el comportamiento heredado sea subjetivo o no, por lo cual, permanece intacto en un ambiente con comportamiento subjetivo. Todo el comportamiento subjetivo es heredado por todas las subclases de una clase dada.

  - **Polimorfismo**: Se extiende este concepto logrando que un objeto subjetivo sea polimórfico consigo mismo de manera no trivial, debido a que es capaz de variar la forma de responder ante un mismo mensaje. Esto se debe a que un mensaje subjetivo se implementa con distintos métodos subjetivos polimórficos entre sí.

- Es posible modelar objetos con cierta "personalidad", ya que pueden responder de diferentes maneras de acuerdo con el o los factores que los influyen. Por lo tanto, se logra modelar objetos más flexibles y adaptables a su ambiente, preparados para brindar una respuesta adecuada cuando reciben un mensaje.

- No se fuerza a que todos los objetos sean subjetivos, ya que es posible definir comportamiento subjetivo sólo en las clases deseadas. De esta manera, se puede atribuir características subjetivas en el comportamiento de aquellos objetos que tenga sentido tal atribución. Esto implica que se mantiene el ambiente, extendiéndose posiblemente con objetos subjetivos y no subjetivos.

- En los modelos donde se incluyen clases subjetivas, se disminuye la cantidad de clases, dado que la introducción de comportamiento subjetivo reemplaza a la subclasificación en algunos casos, y evita la introducción de nuevas jerarquías en otros, reduciendo de esta manera una explosión de la jerarquía de clases. Esto se vio en los ejemplos de la fuerza del colaborador interno y la fuerza del estado, donde la resolución con comportamiento subjetivo implicó una reducción en el número de clases.

- En algunos casos se eliminan muchas de las sentencias condicionales ("if" y "case"). El polimorfismo elimina muchas de estas sentencias durante la programación, sin embargo, con comportamiento subjetivo, se reduce otra gran cantidad de sentencias condicionales, ya que en el momento que un objeto debe responder a un mensaje, de acuerdo con el o los factores que lo están influyendo, automáticamente se ejecuta el método subjetivo que corresponde. De esta manera se logra código más claro y legible, sin tantas sentencias condicionales y doble disptaching que no siempre son fáciles de seguir y entender.

- El comportamiento subjetivo puede ser utilizado en cualquier dominio de aplicación que sea modelado y implementado en un ambiente de programación orientado a objetos. En algunos casos, los elementos que define el comportamiento subjetivo básico [11] son suficientes para ser utilizados en una determinada aplicación orientada a objetos. En otros casos, tiene sentido extender el comportamiento subjetivo (las fuerzas) para que se adecue de mejor manera al dominio de la aplicación que se esté modelando, como en el caso de los OOCVEs. De todos modos, el modelo asociado al comportamiento subjetivo es lo suficientemente flexible como para extenderse sin mayores complicaciones.

- Es posible introducir nuevas fuerzas de influencia para los objetos. A medida que se comprenda mejor cómo se comportan los objetos del mundo real o irreal (como en el caso de los OOCVEs), es muy posible que aparezcan nuevas fuerzas a modelar dentro del ambiente subjetivo.

FUTUROS TRABAJOS

Dado que el comportamiento subjetivo es un tema poco investigado, existen algunos puntos a analizar y perfeccionar con relación al mismo.

En primer lugar, debería ponerse a prueba el uso del comportamiento subjetivo en distintos dominios de aplicaciones orientadas a objetos, tanto en la parte de análisis y diseño como en la implementación. De esta forma, se lograría visualizar de forma más clara las ventajas y desventajas del uso de este nuevo concepto en el desarrollo de sistemas, permitiendo además encontrar nuevos conceptos (en la parte de análisis y diseño) y herramientas (en la parte de implementación) que serían útiles y necesarias.

Con relación a la implementación del comportamiento subjetivo, sería bueno buscar nuevas alternativas en el diseño de la arquitectura que lo soporta, para mejorar el rendimiento y la eficiencia de su funcionamiento. Por ejemplo, podría resultar conveniente para lograr este objetivo definir una máquina virtual de Smalltalk especializada en la evaluación de mensajes subjetivos, de manera de que este proceso se realice a más bajo nivel, consiguiendo mayor velocidad y eficiencia. Otro tema importante a analizar, que sería de gran utilidad en la fase de diseño, está relacionado con los patrones de diseño. Es posible que ciertos problemas donde se aplican los patrones de diseño puedan ser resueltos utilizando el comportamiento subjetivo en lugar de dicho patrones, reduciendo quizá complejidad en el código, número de clases, jerarquías de clase, etc.

También, resulta de gran interés analizar el comportamiento subjetivo en la rama de la Inteligencia Artificial. Dado que este concepto agrega la capacidad de un comportamiento más complejo en los objetos, sería bueno investigar su influencia en la definición de entidades inteligentes o Agentes.

[11] El comportamiento subjetivo básico se refiere al comportamiento subjetivo definido para un ambiente orientado a objetos tradicional.

BIBLIOGRAFÍA

Bibliografía general

[BOO/94] Grady Booch
"Análisis y Diseño Orientado a Objetos con Aplicaciones" (Segunda Edición)
Addison-Wesley/Diaz de Santos
1994

[GAM/95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
"Design Patterns - Elements of Reusable Object-Oriented Software"
Addison Wesley
1995

[BOO/01] Grady Booch, Ivar Jacobson, Jim Rumbaugh
"*OMG-Unified Modeling Language* " – Versión 1.4
Septiembre 2001
Rational Software Corporation

"Basic Aspects of Squeak and the Smalltalk-80 Programming Language"
http://kaka.cosc.canterbury.ac.nz/~wolfgang/cosc205/smalltalk1.html

[BEC/94] Kent Beck, Ralph Johnson.
"Patterns Generate Architectures"
Proceedings of the European Conference on Object-Oriented Programming (ECOOP'94)
Bologna, Italia. Julio 1994

[BOO/99-R] G.Booch, I. Jacobson, J.Rumbaugh
"Unified Modeling Language – Reference Manual"
Addison-Wesley
1998/1999

[DIE/87] Jim Diederich, Jack Milton
"Experimental Prototyping in Smalltalk"
Revista "IEEE Software"
Mayo 1987

[FER/89] Jacques Ferber
"Computational Reflection in Class based Object Oriented Languages"
Proceedings of the Conference on Object-Oriented Programming Systems, Languages And Applications (OOPSLA'89)
New Orleans, LA, USA. Octubre 1989

[GLEN/] Glenn Krasner
"The Smalltalk-80 Virtual Machine"
Revista "BYTE"
Agosto, 1981

[GOL/83] Adele Goldberg, David Robson
"Smalltalk-80: The Language and its Implementation"
Addison Wesley
1983

[HAL/87] Daniel C. Halbert, Patrick D. O'Brien
"Using Types and Inheritance in Object-Oriented Programming"
Revista "IEEE Software"
Septiembre 1987

[HAY/94] Wayne Haythorn
"What is object-oriented design?"
Revista "Journal of Object Oriented Programming" (JOOP)
Marzo-Abril 1994

[HEL/93] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
"Design Patterns: Abstraction and Reuse of Object-Oriented Design"
Proceedings of the European Conference on Object-Oriented Programming (ECOOP'93)
Kaiserslautern, Alemania. Julio 1993

[JOH/88] Ralph Johnson, Brian Foote
"Designing Reusable Classes"
Revista "Journal of Object-Oriented Programming" (JOOP)
Junio/Julio 1988

[JOH/92] Ralph Johnson
"Documenting Frameworks using Patterns"
Proceedings of the Conference on Object-Oriented Programming Systems, Languages And Applications (OOPSLA'92)
Vancouver, B.C., Canadá. Octubre 1992

[MEY/87] Bertrand Meyer
"Reusability: The Case for Object-Oriented Design"
Revista "IEEE Software"
Marzo 1987

[SIE/95-InfoNote 007] Linda Siener
"VisualWorks Portability"
VisualWorks InfoNote 007
Agosto 1995

[SIE/95-InfoNote 010] Linda Siener
"The Benefits of Object-Oriented Technology"
VisualWorks InfoNote 010
Agosto 1995

[STE/88] Lynn Andrea Stein, Henry Lieberman, David Ungar
"A shared view of sharing: The Treaty of Orlando"
Brown University Technical Report
Octubre, 1988
(También en el Capítulo 3 de "Object Oriented Concepts, Applications and Databases", Won Kim, Fred Lochovsky, editors, ACM Press, 1988)

[UNG/87] David Ungar, Randall B. Smith
"Self: The power of simplicity"
Proceedings of the Conference on Object-Oriented Programming Systems, Languages And Applications (OOPSLA'87)
Orlando, FL, USA. Octubre 1987

[WIL/95] Nancy Wilkinson
"Using CRC Cards - An Informal Approach to Object Oriented Development"
SIGS Books
1995

[WIR/90] Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener
"Designing Object-Oriented Software"
Prentice Hall
1990

[WOO/97] Bobby Woolf
"Polymorphic Hierarchy: Subimplementro methods are the key"
Revista "The Smalltalk Report"
Enero, 1997

Bibliografía sobre CVE

[BEN/01] Steve Benford, Chris Greenhalgh, Tom Rodden, y James Pycock
"Collaborative Virtual Environments" – Communications of the ACM
Vol. 44, No.7
July 2001

[DIA/98] Alicia Díaz - Alejandro Fernández
"A pattern Language for Virtual Environment"
Position Paper – 1998

[GRE/98] Soul Greenberg and Mark Roseman
"Using a room metaphor to ease transitions in groupware"
Soul Greenberg and Mark Roseman
Research report - 1998

[HAY/99] Cynthia Haynes and Jan Rune Holmevik
"High Wired – On the design, use, and theory of educational MOOs"
The university of Michigan Press
1999

Bibliografía sobre Comportamiento Subjetivo

Tesis de Licenciatura:"Subjetividad en un Ambiente de Objetos"
Callegari, Claudia – Eliaschev, Laura – Tanner, Carla.
UBA

[FOO/95] Brian Foote
"An Objective Look al Subjectivity"
Object-Oriented Programming Systems Languages And Applications, Workshop sobre "Subjectivity in Object Oriented Systems" (OOPSLA'95)
Austin, TX, USA. Octubre 1995

[HAR/93] William Harrison, Harold Ossher
"Subject-Oriented Programming (A Critique of Pure Objects)"
Proceedings of the Conference on Object-Oriented Programming Systems, Languages And Applications (OOPSLA'93)
Washington, DC, USA. Septiembre 1993

[HAR/94] William Harrison, Harold Ossher, Randall B. Smith, David Ungar
"Subjectivity in Object-Oriented Systems – Workshop Summary"
Object-Oriented Programming Systems Languages And Applications, Workshop sobre "Subjectivity in Object Oriented Systems" (OOPSLA'94)
Portland, OR, USA. Octubre 1994

[LIE/86] Henry Lieberman
"Using Prototypical Objects to Implement Shared Behaviour in Object Oriented Systems"
Proceedings of the Conference on Object-Oriented Programming Systems, Languages And Applications (OOPSLA'86)
Portland , OR, USA. Noviembre 1986

[MIL/96] Hafedth Mili, William Harrison, Harold Ossher
"Supporting Subject-Oriented Programming in Smalltalk"
Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS'96)
USA. Agosto 1996

[OSS/99] Harold Ossher, Peri Tarr
"*Using Subject-Oriented Programming to Overcome Common Problems in Object-Oriented Software Development and Evolution*"
Proceedings of the 1999 International Conference on Software Engineering.
Los Angeles, CA, USA. 16-22 de Mayo de 1999

[PRI/95] Máximo Prieto, Pablo Victory
"Real World Object Behaviour"
Object-Oriented Programming Systems Languages And Applications, Workshop sobre "Subjectivity in Object Oriented Systems" (OOPSLA'95)
Austin, TX, USA. Octubre 1995

[PRI/96] Máximo Prieto, Pablo Victory
"Subjectivity: Towards True Polymorphism"
Object-Oriented Programming Systems Languages And Applications, Workshop sobre "Subjects and Viewpoints throughout the Life Cycle" (OOPSLA'96)
San Jose, CA , USA. Octubre 1996

[PRI/97] Máximo Prieto, Pablo Victory
"Subjective Object Behaviour"
Revista "Object Expert"
Vol.2/3. Marzo/Abril 1997

[RIE/95] Dirk Riehle
"Subjectivity in Object-Oriented Systems"
Object-Oriented Programming Systems Languages And Applications, Workshop sobre "Subjectivity in Object Oriented Systems" (OOPSLA'95)
Austin, TX, USA. Octubre 1995

[TAI/93] Antero Taivalsaari
"Object-oriented programming with modes"
Revista "Journal of Object Oriented Programming" (JOOP)
Junio 1993

Referencias

[Agora]
P. Steyaert, W. Codenie, T. D' Hondt, K. D. Hondt, C. Lucas, y M. V. Limberghen. *Nested mixin-methods in Agora*. En Proc. of ECOOP '93, LNCS 707. Springer-Verlag. 1993.

[Brain]
http://brain.Sourceforge.net

[Chambers, 1993]
Craig Chambers. *The Cecil language: Specification and rationale.* Technical report tr 93-03-05, Department of Computer Science and Engineering. University of Washington. Marzo 1993.

http://www.cs.washington.edu/research/projects/cecil/www/cecil-home.html

[Ingalls, 1981]
Dan Ingalls. *Design Principles Behind Smalltalk.* Byte Vol. 6(8), p. 290.

[Keene, 1989]
S. Keene. *Object-Oriented Programming in Common Lisp*. p. 125. Reading, MA. Addison-Wesley. 1989.

[Python]
http://www.python.org

[Ungar et al., 1987]
David Ungar y Randall B. Smith. *Self: The Power of Simplicity*. Proceedings of the OOPSLA´87, Orlando, FL, October, 1987, pp. 227–242. A revised version appeared in the Journal of Lisp and Symbolic Computation, 4(3), Kluwer Academic Publishers. Junio 1991.