(Draft) Actas de la VII Conferencia de la Asociación Española para la Inteligenccia Artificial: CAEPIA´97. 1997, p. 135-144

Utilización de técnicas de programación basadas en frames para incrementar la potencia de representación en clases de C++ para aplicaciones de sistemas de información

J. Valiño J. Zarazaga S. Comella J. Nogueras P. R. Muro-Medrano
Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior, Universidad de Zaragoza
María de Luna 3, 50015 Zaragoza
{juanv, javy, prmuro@posta.unizar.es}

Séptima Conferencia de la Asociación Española parala Inteligencia Artificial, CAEPIA 97, celebrada en Málaga, España, del 12-14 Noviembre de 1997

Resumen

En este trabajo se pone de relieve el interés de la utilización de técnicas de representación basadas en frames para la creación de los modelos de objetos de sistemas de información. Se ilustran las ideas básicas del desarrollo de un esquema de representación basado en frames en lenguaje C++. Los problemas derivados de la utilización de este lenguaje de programación son abordados y resueltos. La bondad de la aproximación ha sido demostrada en el desarrollo de un sistema de información de uso comercial.

Abstract

This work is focused on the interest of the use of representation techniques based on frames for the design of object model for information systems. The basic ideas of the development of a frame based representation schema in C++ are ilustrated. The problems derived from the use of such programming language are addressed and solved. The goodness of approach has been demostrated in the development of an information system of comercial use.

(Draft) Actas de la VII Conferencia de la Asociación Española para la Inteligenccia Artificial: CAEPIA´97. 1997, p. 135-144

Utilización de técnicas de programación basadas en frames para incrementar la potencia de representación en clases de C++ para aplicaciones de sistemas de información

J. Valiño J. Zarazaga S. Comella J. Nogueras P. R. Muro-Medrano
Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior, Universidad de Zaragoza
María de Luna 3, 50015 Zaragoza
{ juanv, javy, prmuro@posta.unizar.es}

Palabras clave: representación del conocimiento, frames, sistemas de información, orientación a objetos, C++, persistencia de objetos, interfaces gráficos de usuario
Keywords: knowledge representation, frames, information systems, object oriented, C++, object persistence, GUI

RESUMEN: *En este trabajo se pone de relieve el interés de la utilización de técnicas de representación basadas en frames para la creación de los modelos de objetos de sistemas de información. Se ilustran las ideas básicas del desarrollo de un esquema de representación basado en frames en lenguaje C++. Los problemas derivados de la utilización de este lenguaje de programación son abordados y resueltos. La bondad de la aproximación ha sido demostrada en el desarrollo de un sistema de información de uso comercial.*

ABSTRACT: *This work is focused on the interest of the use of representation techniques based on frames for the design of object model for information systems. The basic ideas of the development of a frame based representation schema in C++ are ilustrated. The problems derived from the use of such programming language are addressed and solved. The goodness of approach has been demostrated in the development of an information system of comercial use.*

1. INTRODUCCIÓN.

A medida que los sistemas de información son más avanzados, los clientes requieren mayor variedad de servicios de información lo que conduce a su vez a mayores y más complejos sistemas de información. Para desarrollar nuevos servicios con rapidez, el software debe construirse en base a esquemas de representación y diseños que proporcionen una potente capacidad expresiva y resulten fáciles de comprender, extender y reutilizar.

Una de las aproximaciones tradicionales en la Inteligencia Artificial se ha focalizado en los aspectos de representación de la información/conocimiento (Brachman & Levesque 85). Las ideas básicas de estos esquemas de representación (Minsky 81), que provenían de las redes semánticas (Brachman 79), consisten en proporcionar formas de expresar la estructura lógica del conocimiento para dar flexibilidad en la asociación de procedimientos con piezas de conocimiento específicas y controlar la accesibilidad relativa de diferentes hechos y descripciones. Estos formalismos, con la denominación genérica de frames, están basados en entidades conceptuales estructuradas con descripciones asociadas (Bobrow &

Winograd 77). Estas entidades forman una red de unidades de memoria con diferentes tipos de conexiones, teniendo cada una implicaciones bien definidas para el proceso de recuperación. Los procedimientos pueden estar asociados directamente con la estructura interna de una entidad conceptual. Estos formalismos proporcionan buenos mecanismos de modularización y reutilización del conocimiento (centralizando conocimiento asociando comportamiento procedimental o a través de relaciones entre entidades y utilizando mecanismos de inferencia como el de herencia) que ya han sido transferidos a entornos más especializados de sistemas de información con mayores requisitos de ingeniería del software a través de los lenguajes y métodos de programación orientados a objeto (Stefik & Bobrow 85).

Una parte del éxito obtenido por los frames desde el punto de vista de su utilidad para la programación de sistemas, son sus posibilidades para describir distintos aspectos (facets) de los atributos (slots) de las entidades, es decir conocimiento sobre el propio conocimiento de la entidad. Además del metaconocimiento específico de las entidades derivado del dominio de aplicación, existiría de partida cierto metaconocimiento originado por su uso en un sistema de información. El conocimiento necesario para su persistencia (tipos de dato, nombre de tablas y de campos si la persistencia es en una base de datos relacional) sería un ejemplo de metainformación útil para la entidad. Los requerimientos del interfaz gráfico de usuario proporcionan otra fuente de utilización de este metaconocimiento. En este sentido, con frecuencia es necesario adquirir información del usuario para completar la información de las entidades, y puede resultar útil asociar el conocimiento de como adquirir esta información en la propia entidad: texto a salir por pantalla, incluso objetos gráficos necesarios, chequeo de datos de entrada, apariencia de salida (p.e. lista desplegable con valores para un tipo de dato enumerado), …

Aunque no existe una definición formal de las características de un frame (muchas veces denota objetos muy diferentes), en lo que si hay un acuerdo es en que su organización está siempre basada en una estructura en tres niveles *frame-slot-facet*[1] (Masini et al. 89). El propósito del presente trabajo es desarrollar un sistema basado en frames, en base a la mencionada organización a tres niveles, como esquema de representación para sistemas de información desarrollados en C++.

C++ (Stroustrup 97) es un lenguaje de programación que, tomando como base el lenguaje C, añade utilidades específicas para la programación orientada a objetos. Su compatibilidad con el muy popular lenguaje C (lo que facilita un cambio no traumático), su potencia y su eficiencia han hecho de él el lenguaje orientado a objetos más empleado en las aplicaciones profesionales. Sin embargo, C++ conlleva una serie de problemas para la programación de los aspectos necesarios para soportar el concepto de frame que no surgen cuando se utilizan otros lenguajes de programación más clásicos en el área de IA como el Common Lisp. La realización de la implementación en C++ nos lleva a añorar algunas utilidades de programación de nuestra experiencia en el desarrollo sistemas basados en frames en Common Lisp entre los que cabe destacar los siguientes:

1. posibilidad de manejar los programas (tipos, clases, código) como datos;
2. posibilidad de crear nuevas clases en tiempo de ejecución;
3. muy fácil manipulación de estructuras de datos dinámicas;
4. posibilidad de trabajo de forma no tipada, frente a la programación fuertemente tipada de C++ (con las que obtiene sus ventajas de eficiencia de ejecución); …

Estas características de C++ hacen que el soporte para las utilidades de frames sean mucho más dificultosas de desarrollar de forma adecuada. Para ilustrar nuestra aproximación a este desarrollo se ha estructurado el resto del trabajo de la siguiente forma. En la sección 2

[1] En este trabajo utilizaremos la terminología inglesa de frames y slots pues creemos que está muy extendida y no encontramos traducciones al castellano suficientemente adecuadas y aceptadas.

se describen algunos aspectos de los frames que sería interesante soportar en el esquema de representación propuesto, mientras que la sección 3 describe la infraestructura utilizada para soportar estos aspectos de interés. El trabajo finaliza con una sección de conclusiones.

2. ALGUNOS ASPECTOS DE LOS FRAMES DE INTERÉS PARA SISTEMAS DE INFORMACIÓN.

La Figura 1 ilustra parcialmente un ejemplo típico de frame con la organización de información en base a tres niveles y con una serie de facets para el slot “nombre”. A continuación se enumeran algunos componentes básicos de un esquema de representación basado en frames típico con objeto de ilustrar las características más relevantes y proporcionar el contexto donde encuadrar la infraestructura de frames desarrollada en la siguiente sección. El desglose se ha dividido en dos categorías (Masini et al. 89):

- *Declarativos*. Definen aspectos estructurales de los slots.
- *Procedurales*. Especifican comportamiento asociado localmente a los slots y personalizan su acceso. Son utilizados normalmente para realizar efectos laterales como calcular o actualizar el valor del slot.

```
Persona
  is-a object
  nombre
     value :
     restriccion :
     valueClass : string
     maxLeng : 20
     if-needed :
     if-added :
     defaultValue : 'Innombrable'
     persistence : si
     slotName : 'nombre'
     slotLongName : 'Nombre: '
  ...
```

Figura 1: Frame de Persona.

2.1 Características estructurales.

- **Descripciones taxonómicas**. Una característica imprescindible de los lenguajes de representación basados en frames consiste en sus constructores para describir individuos y clases de individuos en el dominio de aplicación formando taxonomías. Un frame representando una clase puede contener descripciones prototípicas de los miembros de la clase, así como descripciones de la clase misma (Fickes &. Kehler 85). Debe proporcionar, así mismo, los mecanismos necesarios para implementar la estrategia básica de inferencia por herencia.
- **Descripciones de atributos**. Una parte del éxito obtenido por los frames desde el punto de vista de su utilidad para la programación de sistemas, son su potencia expresiva para describir los atributos de los objetos.
    - **Valores por defecto en los atributos**. Los frames disponen de diversos mecanismos (valores propios o inferidos por herencia u otros mecanismos de atadura procedural) para inferir el valor de los slots. Sin embargo, existen ocasiones en las que dichos mecanismos no proporcionan ningún valor útil. En estos casos los frames proporcionan todavía otro mecanismo de inferencia por defecto por el que se proporciona el valor a utilizar en esta situación.

- **Clase de valores y restricciones en el rango de los valores de los atributos**. Los frames proporcionan también la posibilidad de especificar la clase de valores que puede almacenar el slot. Incluso dada una clase, no todos los posibles valores de la clase a la que pertenece un atributo pueden no ser adecuados. Generalmente los frames proporcionan también la posibilidad de especificar el rango de los posibles valores. En el caso de sistemas de información en los que es frecuente que el usuario introduzca valores, resulta particularmente útil disponer de mecanismos automáticos que permitan detectar el hecho para reaccionar adecuadamente y preservar, de esta forma, la integridad semántica del sistema de información.
- **Otros aspectos de los atributos**. Además de ciertas propiedades estándar de los atributos como las descritas anteriormente, los frames proporcionan la infraestructura para integrar otro tipo de conocimiento más dependiente del dominio de aplicación. Por ejemplo, dentro del contexto de sistemas de información, puede resultar útil centralizar en el atributo informaciones relacionadas con su apariencia en el interfaz gráfico de usuario o con el acceso al gestor de base de datos. Así por ejemplo, disponer de una propiedad con un string conteniendo el nombre textual del atributo, o el de la clase a la que pertenece, facilitaría la realización de tareas de entrada/salida (mostrando ese nombre al usuario le identificamos el atributo) o el trabajo con su tabla de la base de datos donde tiene persistencia el objeto (nombre del campo dentro de la tabla). De esta forma el propio frame “sabe” cómo representarse o cómo darse persistencia ofreciendo así a los otros frames una funcionalidad más “inteligente”. Esta centralización del conocimiento relacionado con una entidad, además de constituir una de las recomendaciones de trabajo provenientes ya de la época de las redes semánticas, proporciona interesantes propiedades desde el punto de vista de ingeniería al limitar la duplicidad y facilitar la modificabilidad del código.
- **Relaciones entre entidades**. La centralización del conocimiento específico de una entidad en una estructura de datos propia y el establecimiento de asociaciones con cierta carga semántica con otras entidades con las que tiene algún tipo de relación, son utilidades importantes propuestas desde los orígenes de las redes semánticas (Woods 75) (Brachman 79).

2.2 Propiedades comportamentales.

- **Programación orientada al acceso**. Los frames aportaron un estilo de programación especial denominado programación orientada al acceso.
    - **Daemons para lectura y escritura**. En esta forma de programación, cada acción es el resultado de un acceso a los datos. Los daemons son procedimientos ligados a slots que son invocados cuando se producen acciones de lectura o escritura del valor del slot. Los daemons pueden ser utilizados también para computar valores dinámicamente en base a “cuando-se-necesiten”.
    - **Encapsulación del acceso a los atributos**. Los usuarios de una clase no deben tener acceso directo a un atributo de un objeto. Impidiendo el acceso exterior a la implementación de un atributo se gana independencia entre los distintos frames y posibilita modificar la implementación de la estructura de datos del valor del atributo sin que se vean afectados el resto de los objetos. Si bien esta es una cualidad más requerida por cuestiones de ingeniería del software, es posibilitada (aunque no exigida) por los lenguajes basados en frames a través de los daemons.

- **Envío de mensajes**. Es también casi general que los lenguajes basados en frames proporcionen una peculiaridad típica de programación orientada a objetos mediante la cual, los objetos representados por los frames pueden responder a mensajes.

3. INFRAESTRUCTURA DE FACETS.

C++ es un lenguaje que soporta la programación orientada a objetos para la que proporciona estructuras construidas para declarar clases y objetos que pertenecen a las clases declaradas. Como es habitual en los lenguajes de programación orientada a objetos, una clase de C++ consta de atributos que describen la información y el estado de los objetos en una estructura de datos a dos niveles (objeto-slot). Las clases constan así mismo de métodos que proporcionan servicios y gestionan el acceso a los atributos a través de envío de mensajes. Sin embargo no existe un soporte directo para una infraestructura a tres niveles (objeto-slot-facet) donde los facets almacenan las informaciones sobre el slot y que es la requerida para soportar la idea de frame desde el punto de vista de programación. Resulta necesario por tanto enriquecer la estructura de atributos de C++ para que pueda contener y gestionar el meta-conocimiento relacionado con él[2].

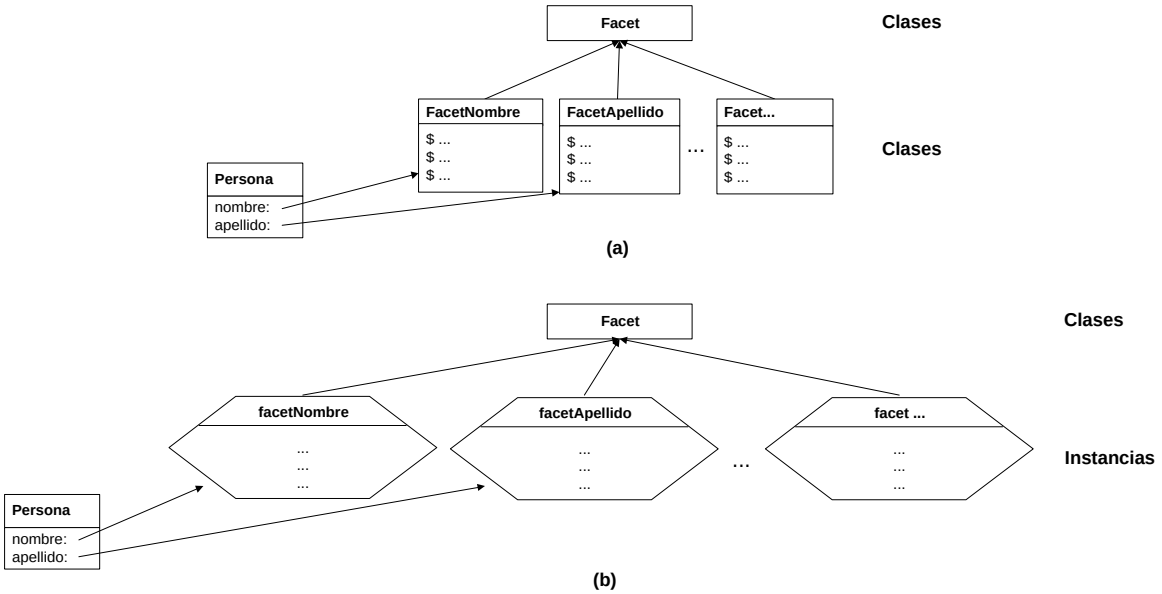

Figura 2: Posibilidades de estructura a tres niveles (objeto-slot-facet).

Una primera aproximación para integrar el nivel de facet podría consistir simplemente en hacer que el atributo de la clase sea un objeto que soportara, tanto la información del valor, como del resto de facets. Sin embargo, esta aproximación simplista tiene el problema del incremento exponencial de declaraciones de clases que serían necesarias (una por slot por clase) como puede verse en la Figura 2a[3].

La solución adoptada para soportar slots consiste en separar la información que es particular de cada instancia (facet con el valor específico) con la que es común a toda la clase. Así se organizará la estructura para almacenar la información del slot con una zona para guardar el valor específico para cada instancia y otra que incluye una referencia a una

[2] Una explicación exhaustiva de las clases de C++ y su composición y código más formal excede el marco del presente trabajo, además de enturbiar posiblemente algunas de las ideas que se quieren resaltar. Los autores se han tomado la libertad de agrupar, eliminar o adaptar clases, funciones, datos y otros aspectos del código cuando han considerado que esto podía contribuir a mejorar la legibilidad del trabajo.

[3] Para hacer la descripción de los diagramas de objetos se ha adoptado la notación gráfica de UML (Unified Modeling Language).

variable con la información común, es decir a una variable de una clase `facet`. La Figura 2b muestra esta nueva situación en la que se aprecia en este caso la utilización de la misma instancia de facet por todas las instancias de la clase (el consumo en memoria resulta similar a la situación anterior debido a las variables estáticas de la clase).
Otro problema adicional proviene del fuerte tipado del lenguaje C++ que impone severas restricciones a la hora de la codificación[4] (por este motivo, la jerarquía de facets sigue un diseño orientado por los tipos de dato de los valores). Para cumplir con los requisitos de codificación se hace necesario templatizar la clase que va a guardar el valor del atributo con una variable de la clase `facet`.

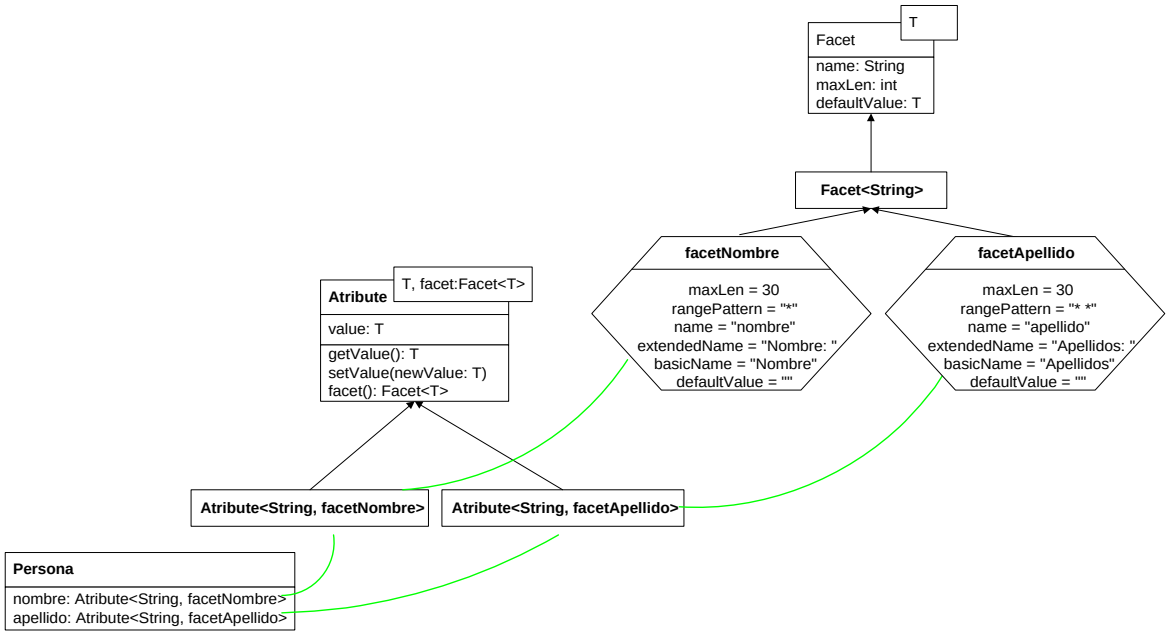

Figura 3: Jerarquía de slots y facets para el ejemplo de Persona.

Volviendo al ejemplo de la clase persona podemos plantear ahora la creación del slot `nombre` con conocimiento sobre patrón de validez, longitud máxima, valor por defecto y nombre básico y extendido del atributo. La Figura 3 muestra la jerarquía necesaria para su creación, las clases más abstractas corresponden a templatizaciones de los objetos `Facet` y `Atribute` para el tipo de dato involucrado (`String`).
Para crear el slot `nombre` en la clase persona `Persona` habría que escribir el siguiente código:

```
RangeFacet<String> facetNombre("Sin nombre", ....)
//La instancia de RangeFacet que asociaremos al atributo nombre

class Persona {
public:
 BasicAtribute<RangeFacet<String>,  facetNombre> nombre;
 ...
}
```

La Figura 4 muestra un esquema con las relaciones entre las estructuras de datos a nivel de instacias de `Persona`, así como las relaciones de herencia. Obsérvese que los slots comparten la misma instancia de `Facet`.

[4] Llegados a este punto siempre se añora la flexibilidad del Common Lisp.

A continuación se repasa como algunos de los aspectos de interés de los frames están soportados específicamente por el esquema de representación propuesto.

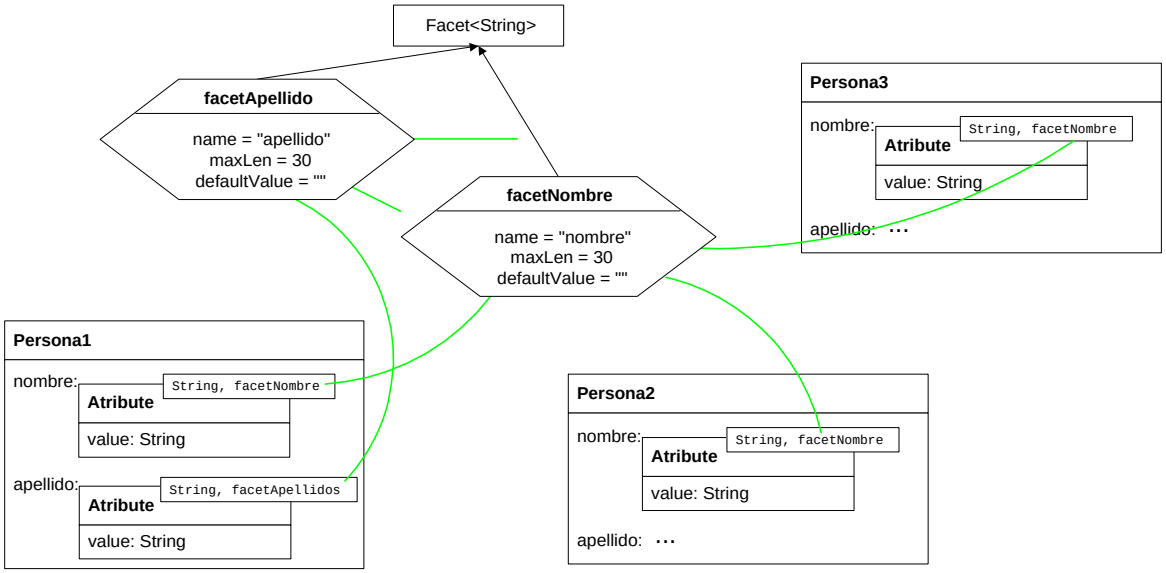

Figura 4: Organización de las estructuras de datos para instancias de Persona.

3.1 Integración de características estructurales.

- **Descripciones taxonómicas**. Son utilizadas las propias de C++ para los casos en que no exista especialización de facets del slot en los descendientes. En caso de que si existan, resulta necesario crear una especialización del objeto facet que cubra los nuevos requisitos y asociarlo nuevamente. Se está trabajando en la mejora de este proceso de herencia para que resulte más simple.
- **Valores por defecto en los atributos**. Basta con incluirlo como un campo del facet del slot en cuestión.
- **Clase de valores y restricciones en el rango de los valores de los atributos**. El parámetro del template de la clase del facet determina el tipo de valores que almacenan los atributos. Las restricciones a los valores válidos se establece en los campos del facet (dependiendo del tipo de facet existen campos como `patron`, `min`, `max` y `maxLen`).
- **Otras descripciones**. Para añadir a un slot facets adicionales se ha de derivar una nueva clase con las adiciones y utilizar una instancia de esta para crear el atributo.
- **Gestión de relaciones entre objetos**. El esquema de representación dispone de una jerarquía de facets especializada en la representación y manipulación de relaciones. En el diseño de esta jerarquía se ha tenido también presente el punto de vista de ingeniería del software (Linenbach 96). En este sentido se ha conseguido que la utilización de las relaciones por parte del programador sea un proceso similar a la utilización de cualquier otro tipo de atributos (lecturas y escrituras homogéneas con el resto de atributos) y todo el proceso de gestión (coherencia en los dos sentidos de la relación, utilización de contenedores, información en memoria o persistente, …) queda encapsulado. La Figura 5 muestra las clases básicas de la jerarquía de facets que dan soporte a las relaciones. Los atributos de relación vienen templatizados por un facet especial que guarda información de aspectos como la cardinalidad de la relación, métodos para insertar y borrar elementos de la relación, nombre de la relación, etc., denominado role. Las clases relación (`Relation`) se encuentran especializadas en base a su cardinalidad y templatizadas por las clases de los objetos que intervienen en ella (`Relation11np <Left, Right>`, `Relation1Nnp <Left, Right>`, `RelationNNnp <Left, Right>`).

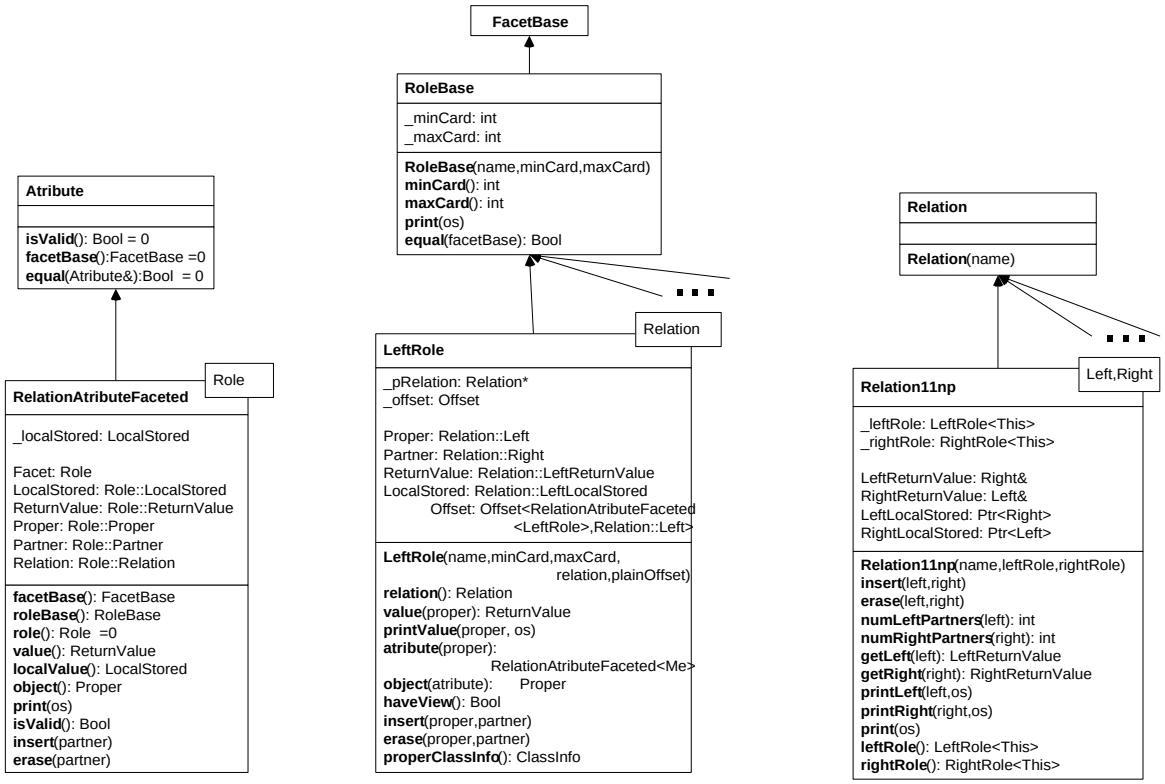

Figura 5: Vista parcial de la jerarquía de facets para relaciones.

3.2 Integración de características comportamentales

- **Programación orientada al acceso. Encapsulación del acceso a los atributos**. La infraestructura de facets propuesta se dispone de un mecanismo uniforme para acceder al valor del atributo mediante los métodos `getValue` y `setValue`, que son quienes integran la información de gestión del valor del slot por defecto (conocimiento de la localización y la estructura de datos y su validación). De forma general, el acceso se realiza de la siguiente forma:

```
Persona p;
p.nombre.setValue(“pepe”);
cout << p.nombre.getValue();
```

- **Daemons para lectura y escritura**. Al igual que con nuevos facets estructurales, es posible integrar nuevos daemons para lectura y escritura derivando una nueva clase del objeto facet involucrado y reescribiendo el método `getValue` o `setValue` oportuno. Si dentro del daemon se desea acceder al valor para leer o escribir se puede utilizar el método `getValue` o `setValue` del padre de la clase.
- **Envío de mensajes**. El esquema de representación soporta el mecanismo de envío de mensajes propio de C++. Adicionalmente, a la hora de acceder al valor que hay dentro del slot puede resultar más elegante usar el idioma Property descrito en (Hastie 95) que en lugar de `getValue` y `setValue` utiliza el operador de llamada a función:

```
Value operator() const {return _value;}
void operator()(const string& newValue) {...}
```

Con el que es posible acceder y modificar el valor del slot utilizando el mismo tipo de notación que en las llamadas a los métodos:

```
Persona p;
p.nombre("Pepe");        //Llama a operator()(string&);
cout << p.nombre();      //Llama a operator()
```

Con esto se crea la ilusión de que nombre es directamente un método de la clase `Persona` que sirve para acceder al valor.

4. CONCLUSIONES

El presente trabajo ha puesto de relieve el interés que puede tener el uso de técnicas estructuradas de representación del conocimiento basadas en frames para la creación de los modelos de objetos en sistemas de información. De esta forma se consigue potenciar los aspectos de representación con mayores posibilidades expresivas, centralizar la localización del conocimiento, así como hacer más sencilla y automatizable su manipulación.

El trabajo describe las ideas básicas para la creación de la infraestructura de frames sobre el lenguaje C++. Las limitaciones impuestas por este lenguaje son las que han dirigido y complicado sustancialmente el desarrollo del software. Nuestra experiencia al respecto es que el trabajo ha sido muy laborioso, ha requerido profundos conocimientos del lenguaje C++ y el software interno ha resultado bastante "enrevesado". Sin embargo, estamos bastante contentos con los resultados, los sistemas se pueden modelar rápida y fácilmente (ayudados de las macros adecuadas) y son fáciles de modificar (las modificaciones de la base de datos y, parcialmente, del interfaz de gráfico de usuario son automáticas), mientras que la complejidad del código queda encapsulada. Por otra parte, para proporcionar generalidad se ha tenido que hacer un uso extensivo de los templates de C++, lo que provoca que el código requiere fuertes recursos de compilación.

El esquema de representación desarrollado ha utilizado con éxito en el desarrollo de un sistema de gestión de redes de radiotelefonía trunking comercial.

AGRADECIMIENTOS.

Este trabajo ha estado parcialmente financiado por la CICYT bajo el proyecto TAP95-0574.

REFERENCIAS.

R.J. Brachman & H.J. Levesque.(1985): *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Inc. 1985.

D.G. Bobrow & T. Winograd (1977): An Overview of KRL, a Knowledge Represenation Language". *Cognitive Science*, Vol. 1, Num. 1, pp. 3-46. 1977.

R.J. Brachman (1979): "On the Epistemological Status of Semantic Networks". *Associative Networks: Representation and Use of Knowledge by Computers*, pp. 3-50. Ed. N.V. Findler, New York: Academic Press, 1979.

R. Fickes & T. Kehler (1985): "The Role of Frame-Based Representation in Reasoning", *Communications of the ACM*. Vol. 28, Num. 9, pp. 904-920. Sept. 1985.

C. Hastie (1995): "A property template for C++". *C++ Report*, Nov-Dec 1995.

T. Linenbach (1996): "Implementing Reusable Binary Associations in C++". *C++ Report*, May 1996.

M. Minsky (1981): Framework for Representing Knowledge". *Mind Design*, pp. 95-128. Ed. J. Haugeland, MA: The MIT Press, 1981.

G. Masini, A. Napoli, D. Colnet, D. Leonard, K. Tombe (1989): *Object Oriented Languages*. The APIC Series, Vol. 34. Academic Press. 1989.

M. Stefik & D. G. Bobrow (1985): "Object Oriented Programming: Themes and Variations", *The AI Magazine*, pp. 40-62. 1985.

B. Stroustrup (1997): *The C++ language, Third edition*, Addison-Wesley. 1997.

W.A. Woods (1975): “What´s in a Link: Foundations for Semantic Networks”. *Representation and Understanding: Studies in Cognitive Science*, pp. 35-82. Ed. D.G. Bobrow & A.M. Collins, New York: Academic Press, 1975.