Un lenguaje formal para especificación y refinamiento de Objetos de Negocio

Raquel Anaya
Universidad EAFIT. Medellín, Colombia
ranaya@sigma.eafit.edu.co

Isidro Ramos
Universidad Politécnica de Valencia. Valencia, España
iramos@dsic.upv.es

Resumen

La descomposición, tanto horizontal como vertical, es una estrategia para enfrentar la complejidad del problema y construír componentes software reusables. El lenguaje extendido de Oasis permitirá formalizar el proceso de especificación y refinamiento de los componentes, incorporando dos tipos de clases : Las clases ***tipo actividad*** *que establecen la comunicación desde y hacia el entorno, implementan las reglas del negocio del comportamiento global y definen el mecanismo de sincronización de servicios. Las clases* ***tipo recurso*** *que describen la estructura y el compromiso (rol) de los objetos en un comportamiento. Según el nivel de abstracción, estos dos tipos representan conceptos abstractos en el dominio del problema o conceptos específicos en el dominio de la solución. El proceso de refinamiento representa un cambio de granularidad que transforma un requerimiento abstracto en un diseño concreto, proceso formalmente soportado por la técnica de* ***reificación****.*

Palabras Claves *: Lenguajes de Especificación, Refinamiento, Comportamiento inter-objetual, Objetos de Negocio*

1. Introducción

El reuso es una estrategia prometedora para enfrentar la demanda de sistemas de información cada vez más complejos que apoyen las operaciones del negocio en entornos organizacionales altamente competitivos y dinámicos [Pri93] [FF95]. La descomposición es uno de los principios básicos de la reutilización ; para que una solución software construida para satisfacer una necesidad actual, pueda ser reutilizada en un necesidad futura, se requiere desarrollar piezas de software flexibles y con el nivel de granularidad adecuado para representar una funcionalidad relevante en el dominio del problema. Dichas piezas ("*features*") deben permitir adecuadamente su integración posterior ("*feature integration*") en otro sistema, sin producir problemas ("*feature interaction problem*") [FIRE]. Se evoluciona entonces de una aproximación orientada al desarrollo de un sistema completo, a una aproximación orientada al desarrollo de componentes [Gog86] [NT95].

La descomposición del espacio del problema puede realizarse de manera horizontal y vertical. La perspectiva *horizontal* esta apoyada en el viejo principio de modularización en el que la funcionalidad total del sistema se descompone en funcionalidades menores en un mismo nivel de abstracción. La perspectiva *vertical* hace énfasis en el aspecto de refinamiento por pasos que involucra más de un nivel de abstracción [Kru92] [ED93][Eck98b].

Las metodologías tradicionales OO permiten la descomposición horizontal y vertical del problema [Rum91][SM91][Jac93][UML]. La descomposición horizontal a través de técnicas como los Casos de Uso y los Diagramas de Colaboración, facilitan desde una perspectiva funcional, el modelado de los objetos del negocio de cara a una funcionalidad útil en el espacio del problema [Fow97]. La descomposición vertical se presenta de manera intuitiva mediante el enfoque iterativo del análisis OO que parte de unas clases abstractas y llega a clases de diseño e implementación. Sin embargo, el tratamiento de estos dos tipos de descomposición se realiza de manera informal ; los entornos de desarrollo generalmente no proveen mecanismos para conservar versiones relevantes del proceso de refinamiento o de la descomposición por funciones, en los que se maneje de forma automática la trazabilidad en un mismo nivel de abstracción (integración de aspectos) o en diferentes niveles de abstracción (de los objetos de negocio a los objetos de diseño).

Por otra parte, existen tendencias importantes de acercamiento de los lenguajes de especificación formal a modelos gráficos convencionales [JWH+95] [MC94] [BW94] [PIP+97]. Esta aproximación logra entornos de desarrollo de software robustos con una semántica y una sintaxis bien definidas y a la vez facilita un acercamiento amigable a la especificación del problema. OASIS y su extensión metodológica OO-Method, se encuentran en esta línea La plantilla de definición de clases en el lenguaje Oasis, se traduce a un conjunto de fórmulas en lógica dinámica, con una semántica declarativa dada por una estructura Kripke [LRS+98].

El objetivo de este trabajo es presentar una versión extendida de Oasis 3.0, para adecuarlo a la construcción de componentes reutilizables que representen diferentes niveles de abstracción y que describan el compromiso que los objetos del negocio adquieren para satisfacer una funcionalidad útil en el espacio del problema.

La segunda parte presenta el aspecto de descomposición horizontal a través de clases tipo actividad en la que se modelan objetos de negocio desde una perspectiva funcional, la tercera parte presenta el aspecto de refinamiento formalmente soportado por el principio de reificación. En cada uno de estos tópicos se relacionan los principales trabajos existentes ; las extensiones propuestas al lenguaje se ilustran con el caso de estudio de alquiler de vehículos; en la presentación del lenguaje se omiten propiedades no relevantes para el propósito de este trabajo. La cuarta parte, a manera de conclusión, sintetiza los principales aportes de este trabajo e introduce el formalismo gráfico que será utilizado para dirigir el proceso de recopilación de requisitos y refinamiento de los objetos de negocio.

2. Aproximación funcional para el modelado de objetos de negocio

2.1 Trabajos en el área

Son diversos los lenguajes formales que tratan el problema del comportamiento cooperativo de objetos desde una perspectiva funcional. Liu et.al utiliza la lógica temporal de primer orden para definir la actividad como un constructor básico que expresa patrones de comportamiento interobjetual y sus mecanismos de agregación y composición [LM96]. El trabajo de [MKB97] define primitivas para

expresar colaboraciones entre objetos en un lenguaje de definición (DDL) por medio de clases de comportamiento (*bclass*). Los trabajos de [HHG90] y [Hol92] introducen el concepto de *contrato* como los constructores de alto nivel para especificar interacciones entre grupos de objetos. Los contratos definen unos participantes con sus obligaciones contractuales (métodos) e incluyen una declaración de los invariantes o reglas que condicionan su comportamiento.

Otros trabajos enfatizan el concepto de rol del objeto. Un rol provee una vista del comportamiento de un objeto en un contexto particular [BFP95]. Aproximaciones medodológicas como IOOM [DP95], propuestas formales como [AR92], entornos complejos de producción de software como [BGK+97] y proyectos grandes de reutilización como Ithaca [BFP95], introducen este concepto como estrategia de modelado para representar objetos con comportamiento complejo ; el rol provee un nivel de granularidad más fino que el objeto mismo, aumentando la potencialidad de reuso. Desde esta perspectiva, un objeto de negocio puede ser definido como una agregación compleja de estructuras y comportamientos no disjuntos construído de manera incremental y conservativa.

2.2 Aproximación funcional en Oasis

Se propone diferenciar entre dos tipos de clases : Clases que representan recursos y clases que representan actividades. Una clase tipo actividad sirve para declarar la colaboración que se establece entre objetos participantes; una clase tipo recurso es la plantilla que describe la estructura y comportamiento de un objeto participante. Estos dos tipos de clase podrían ser declarados con la semántica básica que maneja OASIS que a través del mecanismo de agregación puede dar cuenta de objetos que participan en un comportamiento complejo ; desde nuestro enfoque, más que una agregación se trata de una relación de inclusión. Las extensiones que se proponen, ya sea desde el punto de vista sintáctico o semántico, tienen por objetivo dar mayor riqueza expresiva al lenguaje desde la perspectiva de desarrollo de componentes reusables.

Clase tipo actividad

Una clase tipo actividad puede definirse como la composición concurrente de una sociedad de objetos para un comportamiento específico, en la cual la signatura de los objetos participantes es visible completamente. Esta clase cumple tres funciones básicas : Primera, establecer la comunicación desde y hacia el entorno de cara a este comportamiento; segunda, implementar las reglas del negocio que rigen el comportamiento global y tercero, servir como mecanismo coordinador de los servicios que requieren sincronización. Los objetos tipo actividad representan instancias de ejecución de una actividad global que para nuestro propósito se corresponde con un caso de uso[Ana99].

El cuadro no. 1 presenta la especificación de la actividad *AlquilarVehículo* . Se tiene un bloque para declaración de los objetos participantes. Cada participante es definido con un nombre de variable, nombre de la clase a la que corresponde y el nombre del rol. Los participantes de esta actividad son los objetos de negocio *cliente* y *vehículo*, con los roles de *alquilador* y *objeto_alquiler*, respectivamente. *SolicitarVehiculo* y *RecibirVehiculo* son estímulos recibidos del entorno que aparecen en la signatura de la clase, los cuales originan una solicitud de servicio a los objetos participantes (a) y (b).

Los atributos constantes de esta clase representan los parámetros generales asociados al comportamiento. En el ejemplo, el número de días límite que puede estar un vehículo alquilado

(*PlazoLimAlquiler*) y el número máximo de vehículos que puede tener un cliente a cargo (*NroVehCliente*) son parámetros definidos para cada instancia de ejecución de la actividad en el momento de su creación. Estos parámetros describen propiedades volátiles de la actividad que generalmente se asocian a las reglas de negocio que rigen el comportamiento global. Las reglas del negocio pueden ser declaradas a través de precondiciones, restricciones de integridad, etc.; en este caso la precondición indica que se puede ejecutar el servicio *SolicitarVehiculo* si el cliente no ha llegado al límite de vehículos que puede tener a cargo (c).

```
Class AlquilarVehiculo
participants
        c : Cliente01 as alquilador;
        h : Vehiculo01 as objeto_alquiler;
constants attributes
        PlazoLimAlquiler : nat ;
        NroVehCliente : nat ;
variable attributes
events
        alta new
        SolicitarVehiculo calling with members                          (a)
                c.SolicitarVeh;
                h.EntregarVeh;
        RecibirVehiculo calling with members                            (b)
                h.RecibirVeh;
                c.DevolverVeh;
preconditions
        SolicitarVehiculo if c.TotalContratos < NroVehCliente           (c)
End Class AlquilarVehiculo
```

Cuadro no. 1 declaración de la actividad abstracta *AlquilarVehículo*

Clases tipo recurso

Son las clases tradicionales que se define en una especificación OO y que normalmente se representan en esquemas de una base de datos relacional. Para aquellas clases que representen entidades del dominio del problema con comportamientos diversos y complejos en más de un contexto (producto, cliente, venta, etc.) se introduce el concepto de rol o aspecto de la clase. Es decir, la definición de tipo declara las propiedades del objeto relevantes para el comportamiento y guarda silencio acerca de propiedades no relevantes [AR92].

El cuadro no. 2 presenta la especificación para los objetos de negocio que participan en el comportamiento *AlquilarVehículo*. La declaración de rol se hace explícita en el encabezado de la clase agregando la palabra clave **played by**, seguido por el nombre de la clase base de la cual forma parte y el nombre del rol que desempeña en este comportamiento. Por ejemplo *Vehiculo01*, está asociado a la clase base Vehículo y describe la estructura y comportamiento relevante para el alquiler en el que desempeña el rol de *objeto_alquiler*.

```
Class Vehiculo01 played by Vehiculo (objeto_alquiler)
...
variable attributes
```

```
        tarifa : nat;
        estado : string;
        condiciones : string
events
        EntregarVeh ;
        RecibirVeh ;
valuations
        [EntregarVeh] estado = 'alquilado';
        [RecibirVeh] estado = 'devuelto'
preconditions
        EntregarVeh if {estado='disponible'};
end Class VehiculoRol01


Class Cliente01 played by Cliente (alquilador)
...
variable attributes
        totalcontratos : nat(0);
events
        SolicitarVeh;
        DevolverVeh;
valuations
        [SolicitarVeh] totalcontratos = totalcontratos + 1;
        [DevolverVeh] totalcontratos = totalcontratos – 1;
end Class Cliente01
```

Cuadro no.2 Declaración abstracta de *Cliente* y *Vehículo*

Modelado del comportamiento interobjetual

El clase tipo actividad representa una sociedad o sistema de objetos que existen concurrentemente. Cada objeto participante posee un ciclo de vida local que para propósitos de este comportamiento debe sincronizarse en diferentes puntos del tiempo para formar un ciclo de vida global distribuído [DE95a].

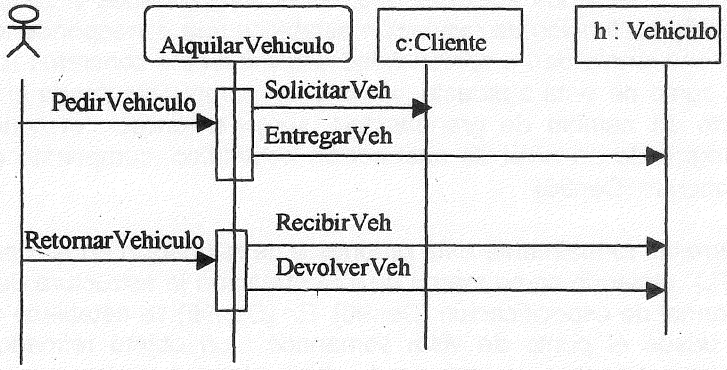

Figura no. 1 Sincronización abstracta para *AlquilarVehiculo*

Los diagramas de secuencia, sirven como formalismo gráfico para capturar los puntos en el tiempo donde los ciclos de vida de la sociedad de objetos deben sincronizarse, como resultado de los requisitos del domino del problema y de la semántica asociada a los respectivos servicios. La figura no. 1 presenta parte del Diagrama de Secuencia para la actividad *AlquilarVehiculo* : cargar un vehículo a un cliente (c.*SolicitarVeh*) debe sincronizarse con modificar el estado del vehículo (h.*EntregarVeh*). En este caso la comunicación es sincrónica de ocurrencia forzada (*event calling*), puesto que los servicios deben ocurrir en los dos objetos o en ninguno de ellos. La coordinación de la comunicación sincrónica es responsabilidad de la clase actividad ; los participantes de la actividad pueden a la vez establecer entre ellos comunicaciones de tipo asincrónico (*action waiting, action sending*) [LRS+98].

Hasta este momento, se tiene una especificación correcta de la especificación en un primer nivel de abstracción en el que se describe el papel que los objetos del negocio *cliente* y *vehículo* desempeñan en la funcionalidad *AlquilarVehículo*. Se trata ahora de describir el proceso de refinamiento de la especificación y los principios en los cuales se fundamenta.

3. Refinamiento de la especificación

3.1 Trabajos en el área

La reificación es una técnica formal que estudia la reducción de la complejidad de software y que está siendo investigada bajo la aproximación de orientación por objetos. La propuesta de refinamiento presentada en este trabajo está apoyada básicamente en la propuesta de Denker et.al [ED93][Den95][Den95a][Den96][HWD96]. El uso del término **reificación** más que refinamiento o implementación hace énfasis en el *cambio de granularidad* de una especificacion. Significa partir de una especificación inicial para construír una secuencia de especificaciones que van reemplazando conceptos abstractos de la especificación por conceptos más cercanos a un programa concreto.

El proceso de reificación puede verse como una aproximación de tres niveles en la que se tienen tres tipos de objetos : Objetos abstractos (*abstract object*), que para nuestro caso se corresponde con los objetos del negocio, que representan comportamientos no deterministicos del dominio del problema, objetos base (*building block*) que describen los comportamientos básicos del problema y los objetos refinados (*middle object*) que son una composición de los objetos abstractos y objetos base, enriquecidos con una especificación de cómo los atributos abstractos son refinados a estructuras de base de datos (reificación de datos) y cómo las acciones abstractas son refinadas a transacciones base (reificación de procesos) [ED93]. La función de reificación establece una correspondencia de los eventos abstractos a procesos de secuencialidad u opcionalidad sobre eventos concretos (base). Un evento que era atómico desde el punto de vista abstracto, viene a ser compuesto desde el punto de vista reificado. Asociado al efecto de cambio de granularidad, surge entonces el concepto de **transacción**, como una unidad lógica de un nivel de abstracción específico, compuesto de varios eventos simples a un nivel más concreto [Den96].

Los trabajos de Denker et.al desarrollan formalmente esta propuesta de reificación en el contexto de la lógica temporal de eventos (DTL) teniendo como marco de interpretación la estructura de eventos y utilizando Troll como lenguaje formal de especificación [Den96]. En [Den95] se establece el criterio de correctitud de la reificación, desde el punto de vista semántico : un objeto refinado es una reificación correcta de un objeto abstracto sobre un conjunto de objetos base cuando se cumplen dos condiciones : Primera, el objeto reificado sólo puede adicionar restricciones al comportamiento del

objeto base, pero no puede modificar su comportamiento y segunda, los objetos reificados pueden ser utilizados sin pérdida de información en lugar de los objetos base. Estas dos condiciones se expresan con la semántica de la estructura de eventos para significar que el conjunto de ciclos de vida que representan el comportamiento del objeto reificado, debe proyectarse tanto en la estructura abstracta como en las estructuras base. Desde el punto de vista sintáctico [DE95a] la prueba de reificación se establece cuando las fórmulas que se derivan de la reificación deben ser satisfechas tanto para los objetos abstractos como para los objetos concretos a través de una sustitución sintáctica.

3.2 Refinamiento de la clase recurso en Oasis

El refinamiento se describe por la palabra **reified** y representa el mecanismo de transformación de objetos abstractos en objetos concretos, tanto en su estructura como en su comportamiento. El cuadro no. 3 presenta una vista simplificada de la reificación de Vehiculo01. El tratamiento formal y exhaustivo del refinamiento en el marco de la Lógica Dinámica y con la semántica de intrepretación de la estructura Kripke, se encuentra fuera del objetivo de este trabajo.

```
Class Vehiculo01R reified Vehiculo01 aggregation of
        static inclusive  Contrato toward(0,1) from( 1,1) ;
        static relational  Tarifa toward(posee : 1,1) from(aplicada_a : 0,*) ;
...
variable attributes
        tarifa : nat derived;                   /* tarifa pasa a ser un atributo derivado
        estado : string;
        combustible : nat;
        kilometros : nat;
        condiciones = {combustible : nat , kilometros : nat}  /*equivalencia del atributo abstracto
events                                                        /* en atributos concretos
        CrearContrato calling to members            /* el objeto se encarga de sincronizar los servicios
                Contrato.Crear;                     /* de sus componentes
        PagarFianza calling to members
                Contrato.Pagar;
        DarEstadoAlq;
        PagarSaldo calling to members
                Contrato.Pagar
        DarEstadoDev;
        EntregarVeh process ENTREGARVEH;        /* ecuaciones de equivalencia de eventos abstractos
        RecibirVeh  process RECIBIRVEH;         /* en procesos concretos
valuations
        [DarEstadoAlq] estado = 'alquilado';
        [RecibirVeh] estado = 'devuelto';
derivations
        tarifa = posee.precio;                  /* fórmula de derivación del atributo refinado
preconditions
        DarEstadoAlq if {estado='disponible'};
operations
ENTREGARVEH :                                   /* descripción de procesos que han sido refinados
EntregarVeh = CrearContrato.VEH0;
VEH0        = PagarFianza.VEH1;
VEH1        = ::DarEstAlq;
```

```
RECIBIRVEH transaction:
RecibirVeh = PagarSaldo.REC0;
REC0 = ::DarEstadoDev;
end Class Vehiculo01R
```

Cuadro no. 3 Declaración refinada de *Vehiculo01*

```
Class Contrato
...
constant attributes
        numero : string ;
        fecha_desde : date;
        nro_dias : nat
variable attributes
        totalpagado : nat(0);
        valorcontrato : nat;
        fecha_hasta : date;
events
        Crear new;
        Pagar ;
        Cerrar destroy;
valuations
        [Pagar(cant)] total_pagado = total_pagado + cant;
end class Contrato


Class Tarifa aggregation of
        static relational Vehiculo01R toward(aplicada_a : 0,*) from(posee : 1,1) ;
identification
        codigo : (codigo)
constant attributes
        codigo : nat;
        descripción : nat
variable attributes
        precio ;
events
        crear_tarifa new ;
        cambiar_precio (cant nat) ;
        borrar_tarifa destroy;
...
valuations
        [cambiar_precio(cant)] precio_alq = cant;
protocols
TARIFA :
        TARIFA0 =
                crear_tarifa TARIFA1
        TARIFA1 = cambiar_precio.TARIFA1 +
                borrar_tarifa;
end Class Tarifa
```

Cuadro no. 4 Especificación de objetos base para *Vehiculo01R*

En la propuesta de Denker et.al la transacción es el concepto que representa un cambio de granularidad ; una transacción comprende un conjunto de acciones ejecutadas consecutivamente sin observabilidad intermedia [Den96]. Es importante resaltar la riqueza semántica y expresiva que posee Oasis para describir el objeto como un proceso observable [RPC+93] [LRS+98]. La versión 3 establece diferencia entre procesos que *pueden* ocurrir en la traza del objeto (protocolos), procesos que *deben* ocurrir (operaciones) y procesos que tienen una ocurrencia atómica sin observabilidad intermedia (transacción) ; utiliza un subconjunto de CCS como lenguaje para especificar la diversidad de procesos. De esta manera se pueden establecer funciones de reificación más ricas que en la propuesta original, facilitando el modelado de problemas reales de gestión de objetos de negocio.

Las clases *Contrato* y *Tarifa* surgen en el proceso de refinamiento de *Vehiculo01* (cuadro no.4). *Contrato* es un objeto agregado de *Vehiculo01*, lo cual indica que todos sus servicios serán coordinados a través de Vehículo01. *Tarifa* es un objeto secundario asociado con *Vehículo* que se maneja en Oasis como una agregación de tipo relacional ; este objeto puede suscribir en su signatura local, eventos con el entorno expresados en su sección de protocolo.

3.3 Refinamiento de la clase actividad en Oasis

Vale la pena considerar ahora, el efecto que el proceso de reificación tiene en la sincronización que se declaró a nivel abstracto. Como ya fue mencionado, la clase actividad será la encargada de establecer la coordinación de eventos entre el objeto complejo (actividad) y los objetos agregados (participantes) a través de una comunicación sincrónica y de ocurrencia forzada. Teniendo en cuenta que los eventos a nivel abstracto se reifican como procesos (operaciones, protocolos, transacciones) en el nivel básico, se hace necesario analizar el efecto que el refinamiento tiene sobre esta sincronización inicial.

En la propuesta original de reificación, Denker et.al establece una regla que declara que la especificación de una interacción abstracta significa que hay un 'punto de comunicación' durante la ejecución de las acciones [Den96]. Los planteamientos dados por Denker et.al, están orientados a establecer modelos de sincronización en los que se optimiza la concurrencia de acciones involucradas en una transacción considerando aspectos de control de concurrencia y paralelismo de transacciones. Para el propósito práctico que persigue este trabajo se hace necesario establecer los principios que determinen en qué momento es relevante redefinir la sincronización sobre los objetos reificados.

En situaciones donde las acciones abstractas se refinan en procesos complejos y específicamente cuando los pasos del proceso contemplen estímulos del entorno (existe observabilidad intermedia) , es deseable que el diseñador pueda establecer los puntos de sincronización adecuados ; esperar que éstos se sincronicen en su punto de finalización podría ser una restricción fuerte para el modelo de comunicación. A continuación se intenta explicar mediante el ejemplo en estudio.

La clase *AlquilerVehículo* declara dos casos de sincronización :

(1) happens c.SolicitarVeh $\Rightarrow$ happens h.EntregarVeh

(2) happens h.RecibirVeh $\Rightarrow$ happens c.DevolverVeh

y dos funciones de refinamiento :

ρ (EntregarVeh) = Operation{CrearContrato, PagarFianza, DarEstAlq}

ρ (RecibirVeh) = Transaction{PagarSaldo, DarEstadoDev}

Para el caso (2) donde *RecibirVeh* se convierte en transacción y *DevolverVeh* no se descompone, la sincronización se establece en el punto de confirmación de la transaccioón RecibirVeh con el evento DevolverVeh, es decir, la declaración de sincronización no se modifica. Para el caso (1) donde *EntregarVeh* se convierte en una operación podrían presentarse alternativas de sincronización válidas y significativas desde el dominio de la aplicación : En qué momento se hace efectivo el aumento de vehículos a cargo del cliente (SolicitarVeh) ? Al crear el contrato o cuando el cliente haya pagado la fianza ? En este caso se seleccionó la primera alternativa.

El cuadro no. 5 presenta la especificación refinada de *AlquilarVehiculo*. En esta especificación se puede realizar una declaración completa de las precondiciones, restricciones, etc. que representan las reglas del negocio. Es decir, la especificación refinada de un tipo actividad puede establecer limitaciones de comportamiento sobre los objetos base. La especificación refinada de la actividad se realiza de manera asistida a través de un Diagrama de Secuencia similar al mostrado en el figura no. 1.

```
Class AlquilarVehiculoR reified AlquilarVehiculo
participants
        c : Cliente01 as alquilador;
        h : Vehiculo01R as objeto_alquiler;     /* referencia el tipo refinado de Vehiculo
        t : Tarifa;                             /* Tarifa es un agente secundario que surge al
constants attributes                            /* reficar Vehiculo01
        plazo_lim_alquiler : nat ;
        nro_veh_cliente : nat ;
variable attributes
events
        alta new
        SolicitarVehiculo calling with members      /* nueva declaración de sincronización sobre
                c.SolicitarVeh;                     /* los eventos concretos
                h.CrearContrato;
        RecibirVehiculo calling with members
                 h.RecibirVeh;
                c.DevolverVeh;
        CrearContrato calling with members      /* nuevos servicios que pueden ser activados
                h.CrearContrato;                        /* por el entorno
        PagarFianza calling with members
                h.PagarSaldo;
        PagarSaldo calling with members
                h.PagarSaldo ;
        CrearTarifa calling with members
                t.crear_tarifa ;
        ...
valuations
preconditions
        SolicitarVehiculo if c.totalcontratos <  nro_veh_cliente                    (c)
End Class  AlquilarVehiculoR
```

Cuadro no. 5 Especificación refinada de *AlquilarVehiculo*

4. Conclusiones

Se ha presentado una extensión al lenguaje formal Oasis para especificación de componentes abstractos que representen una funcionalidad relevante en el espacio del problema (partición

horizontal) y que a la vez permita el refinamiento de los objetos de negocio en objetos de diseño (partición vertical).

Este trabajo aumenta la expresividad de un lenguaje formal de especificación para tratar de manera homogénea aspectos que son considerados por los trabajos relacionados de manera independiente. Las extensiones propuestas habilitan el lenguaje como soporte formal para el proceso de desarrollo de componentes en modelos del dominio construidos pensando en su reutilización posterior.

La formalización permite determinar los principios y heurísticos básicos que soportan el entorno gráfico, para capturar los requisitos y refinamiento de los componentes, utilizando buena parte de la notación UML. Una actividad se corresponde con el concepto de Caso de Uso, el vocabulario que permite describir la funcionalidad se define a través del Diagramas de Clase particulares a cada comportamiento (modelo de roles) y la sincronización de acciones y responsabilidades de los objetos participantes se describe por medio de Diagramas de Secuencia [Ana99].

Aunque Denker et.al propone un marco de formalización del refinamiento uniforme desde el punto de vista estructural y de comportamiento, para efectos prácticos este trabajo retoma el aspecto de refinamiento del comportamiento y utiliza principios del área de modelización semántica para el refinamiento estructural [DP95].

Material de Referencia

[Ana99]. Anaya, R. *Un Modelo Para el Desarrollo de Componentes Reusables.* Memorias 2º Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes Software. San José de Costa Rica. Marzo de 1.999. Ed. Centro de Información Tecnológica (CIT), del Instituto Tecnológico de Costa Rica.

[AR92] Andersen, E.; Reenskaug, T. *System Design by Composing Structures of Interacting Objects.* ECOOP'92. European Conference on Object Oriented Programming. Ed. Springer-Verlag, 1.992

[BGK+97] Baumer, D.; Gryczan, G.; Knoll, R.; Lilienthal, C.; Riehle, D.; Zullighoven, H. *Framework Development for Large Systems.* Comunications of the ACM. Volumen 40 no. 10, Octubre de 1.997.

[BFP95] Bellinzona, R.; Fugini, M.; Pernici, B. *Reusing Specifications in OO Applications.* IEEE Software, Marzo de 1.995.

[BW94] Briggs, T.; Werth, J. *A Specification Languaje for Object Oriented Analysis and Design.* Lectures Notes in Computer Science. ECOOP94.

[Den95] Denker, G. *Transactions in Object-Oriented Specification.* http://www.cs.tu-bs.de/idb/publications/.

[DE95a] Denker, G.; Ehrich, H. *Action Reification in Object-Oriented Specification.* http://www.cs.tu-bs.de/idb/publications/.

[Den96] Denker, G. *Reification - Changing Viewpoint but Preserving Truth.* http://www.cs.tu-bs.de/idb/publications/.

[DP95] De Antonellis, V. ; Pernici, B. *Reusing Specifications Through Refinements Levels.* Data and Konwledge Engineering. Volumen 15, 1.995.

[Eck98b] Eckstein, S. Modules for Object Oriented Specification Languages: A Bipartite Approach. http://www.cs.tu-bs.de/idb/publications/.

[ED93] Ehrich, H.; Denker, G. *Constructing Systems as Object Communities*. http://www.cs.tu-bs.de/idb/publications/.

[FF95] Frakes, W. ; Fox, C. *Sixteen Questions About Software Reuse*, ACM Comunications. Volumen 38 no. 6, Junio de 1.995

[FIRE] Feature Integration in Requirements Engineering. http://www.cs.bham.ac.uk/~mcp/fireworks/.

[Fow97] Fowler, M. *UML Distilled. Applying the Standars Object Modeling* Languaje. Addison Wesley, 1.997

[Gog86] Goguen, Joseph A. *Reusing and Interconnecting Software Components*. En Tutorial Software Reusability. The Computer Society of the IEEE, 1.986

[HHG90] Helm, R.; Holland, R.; Gangopadhyyay, D. *Contract : Specifying Behavioral Composition in Object-Oriented Systems*. Memorias ECOOP'90. ACM Press

[Hol92] Holland, I. *Specifying reusable components using Contracs*. Memorias ECOOP'92. Springer-Verlag

[HWD96] Huhn, M.; Wehrheim, H.; Denker, G. *Action Refinement in System Specification : Comparing a Process Algebraic and an Object-Oriented Aproach* . http://www.cs.tu-bs.de/idb/publications/.

[Jac93] Jacobson, Ivar, et.al. Object Oriented Software Enginieering : A Use Case Driven Aproach... Wesley Publishing, 1993

[JWH+95] Jungclaus, R.; Wieringa, R.; Hartel, P.; Saake, G.; Hartmann, T. *Combining TROLL with the Object Modeling Technique*. http://www.cs.tu-bs.de/idb/publications/.

[Kru92] Krueger, C. W.: *Software Reuse*, ACM Computing Surveys. Volumen 24 no. 2, Junio de 1.992

[LM96] Liu, L.; Meersman, R. *The Building Blocks for Specifying Communication Behavior of Complex Objects : An Activity-Driven Approach*. ACM Transaction on Dtabase Systems, Vol 21, No. 2, 1.996

[LRS+98] Letelier, P. ; Ramos, I.; Sánchez, P.; Pastor, O. *OASIS Versión 3.0 : Un Enfoque formal para el modelado conceptual Orientado a Objetos*. Departamento de Sistemas Informáticos y Computación, Facultad de Informática, Universidad Politécnica de Valencia, 1.998

[MC94] Moreira, A.; Clark, R. *Combining Object-Oriented Analysis and Formal Description Techniques*.. Lectures Notes in Computer Science. ECOOP94.

[MKB97] Marshall, B.; Kennedy, J.; Barclay, P.. *Bclass : A construct and method for modelling co-operative object behavior*. Information and Software Technology. Vol 39, 1.997

[NT95] Nierstrasz, O.; Tsichritzis, D. *Object Oriented Software Composition*, Prentice Hall, 1995

[PIP+97] Pastor, O.; Insfrán, E.; Pelechano, V.; Romero, J.; Merseguer J.. *OO-Method : An OO Software Production Environment Combining Conventional and Formal Method*. Advanced Information System Engineering. CAISE ' 97. Eds. Antoni Olivé, Joan Antoni Pastor. Barcelona España, 1.997

[Pri93] Prieto-Díaz, Ruben. *Status Report : Software Reusability*. IEEE Software. Mayo 1.993.

[RPC+93] Ramos, I.; Pastor, O.; Cuevas, J.; Devesa, J. *Objects as Observable Processes*. Actas del 3rd. Workshop on the Deductive Approach to Information System Design, Cataluña, 1.993

[Rum91] Rumbaugh, James et. al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991