Interoperabilidad de Componentes Software mediante cálculo de canales*

Silvia N. Amaro
Departamento de Informática y Estadística,Universidad Nacional del Comahue
Buenos Aires 1400, (8300) Neuquén, Argentina
e-mail: samaro@uncoma.edu.ar

y

Ernesto Pimentel
Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Complejo Tecnológico, Campus Teatinos, (29071) Málaga, España
e-mail: ernesto@lcc.uma.es

Resumen

El desarrollo de software basado en componentes es una disciplina en continuo crecimiento dentro del campo de la ingeniería de software. Un sistema basado en componentes se describe por medio de componentes y sus interacciones. La composición de componentes es definida en términos de interfaces lógicas que describen su comportamiento observable desde el entorno. Los lenguajes de descripción de interfaces tradicionales, provistos por las plataformas orientadas a componentes del mercado sólo dan información sobre los nombres de servicios ofrecidos por los componentes. Nuestra propuesta se orienta a enriquecer esta información mediante la descripción de una abstracción del protocolo de interacción de los componentes.

Pew es un modelo de coordinación basado en cálculo de canales que posee una gran potencia expresiva para especificar software basado en componentes. La composición de conectores es muy flexible y útil respecto a protocolos de coordinación que puedan ser expresados como expresiones regulares sobre operaciones de entrada/salida. En este artículo analizamos el uso de *Pew* para la especificación del comportamiento interactivo de componentes. Presentamos un álgebra de procesos inicial basada en las primitivas de comunicación de *Pew* y algunos ejemplos que muestran su aplicación.

Palabras claves: Lenguajes de coordinación, software basado en componentes, cálculo de canales, interoperabilidad.

Abstract

Component-Based Software Development is an emerging discipline in the field of software engineering. We consider a *software component infrastructure* as a set of interacting software componentsdesigned to ensure that a software system or subsystem constructed using those componnents and interfaces will satisfy clearly defined performance specifications. Traditional Interfaz Description Languages used by available component-oriented platforms in addressing software interoperability describes the services a component offers, but says nothing about the relative order in which component methods are to be invoked, thus there is no guarantee that the components will suitably interoperate. Our work is oriented to enhance component interfaces with a description of an abstract component interaction protocol.

A composition is the combination of two or more software components yielding new component behaviour at a different level of abstraction. The characteristics of the new component behaviour are determined by the components being combined and by the way they are combined. *Pew* is a coordination model based on mobile channels which is powerful enough to specify component-based software. This paper analyse the use of *Pew* to specify the interactive behaviou of software components. we present a process algebra based in *Pew* communication primitives and some examples.

Keywords: Coordination languages, Component-based software, calculus of channels, interoperability.

* Este trabajo es financiado parcialmente por CYTED (Ciencia y Tecnología para el desarrollo), proyecto VII_J_RITOS2 (Red Iberoamericana de Tecnologías de Software para la década del 2000)

1 Introducción

Para responder a la creciente demanda de nuevas tecnologías para la construcción de aplicaciones abiertas y distribuidas, la Ingeniería de Software Basada en Componentes (ISBC) propone la creación de colecciones de componentes de software reutilizables que puedan ser adaptadas e interconectadas en forma dinámica en el desarrollo de nuevas aplicaciones. Para desarrollar sistemas de software a partir de la integración de componentes existentes es necesario que exista un mecanismo de composición que habilite su integración. Podemos encontrar componentes de distintos formatos, diseños e implementaciones. Los componentes pueden diseñarse para trabajar en conjunto o pueden obtenerse en las más diversas fuentes [16]. Todos estos factores influencian en gran medida la tarea de composición de componentes. El desarrollo de sistemas abiertos debería basarse en el uso de componentes y la tecnología debería dar soporte a esa composición.
Un componente es una entidad que puede ser utilizada y compuesta sólo por medio de su interfaz, en la que se describe la entrada/salida y comportamiento observable de las instancias del componente. La interfaz de un componente provee una abstracción que encapsula los detalles de implementación internos. Existen diversos requisitos para el proceso de construcción y uso de componentes [17]:

- La composición de objetos es un ingrediente esencial para los modelos con componentes, que son necesarios para crear las estructuras utilizadas en tiempo de ejecución. La composición puede usarse para enlazar objetos de cualquier granularidad.
- La composición de clases es necesaria cuando se adapta el comportamiento y la cooperación de las instancias. Deberá enfatizarse la separación entre el uso de componentes (caja negra) y el desarrollo de los mismos, que puede seguir un enfoque tradicional (caja blanca).
- Los componentes pueden requerir diferentes tipos de conectores, por ej., un tipo de enlace podría indicar todos los requerimientos manejados por las instancias de un componente.
- Cualquier composición debería asegurar que la aplicación resultante ha sido correctamente construida y en ese sentido, debería poder definirse qué es una composición correcta.

Tradicionalmente los modelos y lenguajes de coordinación [2, 15] han evolucionado alrededor de las nociones de espacio de datos compartido, por un lado, dando origen a la familia de lenguajes orientados a datos; y control y eventos, por otro lado, dando origen a la familia de lenguajes orientados a control. Dentro del primer grupo se encuentra LINDA, como su principal exponente. Se han desarrollado varios trabajos [11] extendiendo Linda para poder utilizar su máxima potencia expresiva en la especificación de protocolos de componentes [10]. Dentro del segundo grupo de lenguajes se encuentra Manifold, que es pionero en la aplicación del modelo IWIM (Idealized Worker Idealized Manager) [3]. Se ha demostrado en [18] que es muy potente para modelar sistemas distribuidos con reconfiguración dinámica de su topología. Con el propósito de resolver los problemas de interoperabilidad [20] que pueden presentarse al trabajar con los Lenguajes de Descripción de Interfaces (LDI) provistos por las plataformas orientadas a componentes del mercado (CORBA [7], COM/DCOM [19], EJB[14], etc.), que sólo permiten describir los servicios ofrecidos por los componentes, pero nada especifican respecto a los servicios requeridos, ni el orden relativo que deben respetar, se han presentado varias propuestas para extender las interfaces con información del comportamiento concurrente de los componentes, utilizando álgebras de procesos [9, 12] y representaciones basadas en roles [13]

En este sentido una alternativa que parece prometedora dentro del campo de los modelos de coordinación es trabajar con cálculo de canales. En el cálculo de canales la interfaz consiste de un conjunto de canales móviles a través de los cuales la instancia del componente envía y recibe valores. Que un canal sea móvil significa que las identidades de sus extremos pueden ser pasadas a través de otro canal y así cambiar físicamente la posición de uno de sus extremos. Que un componente sea móvil significa que puede moverse de una posición a otra dentro del sistema distribuido en que esta inmerso. Los canales y componentes móviles permiten la reconfiguración dinámica de la topología de un sistema.

Tanto los modelos basados en canales móviles como los basados en espacio de tuplas compartido y en eventos comparten las características de comunicación anónima, y la separación de los conceptos de coordinación y computación. Sin embargo, la composición de componentes por medio de canales presenta algunas otras ventajas[4]: permite implementaciones más eficientes, provee seguridad: la comunicación punto a punto es una comunicación privada que evita interferencias accidentales o intensionales sobre la transmisión de los datos, y tiene mayor expresividad arquitectural: los canales son conexiones directas entre componentes, luego son altamente expresivos de la arquitectura del sistema, ya que en este modelo es claro ver qué entidades o componentes pueden ser afectadas por la modificación o reemplazo de algún otro componente.

Pew[1] surge como un lenguaje que permite la construcción composicional de sistemas a partir de componentes que interactúan y cooperan anónimamente a través de conectores. Los conectores se logran por la aplicación de

operaciones primitivas de *Pew* en la composición de canales. Los canales proveen el desacople temporal y espacial de las partes que intervienen en una comunicación. *Pew* es un lenguaje que respeta el modelo IWIM propuesto en [3], y como tal abstrae los detalles de implementación de los componentes encargados de la computación. Presenta un alto potencial para expresar, por medio de sus conectores y su semántica el comportamiento interactivo de componentes software, a la vez que permite la construcción de los otros modelos de coordinación referenciados anteriormente. Al integrar componentes es necesario un código de adaptación. *Pew* propone construir dicho código de manera composicional a partir de conectores primitivos.

El resto de este trabajo es organizado como sigue. La sección 2 introduce los conceptos del modelo propuesto en *Pew*, y la semántica del cálculo de canales y conectores. La sección siguiente ofrece algunos ejemplos de construcción de conectores a partir de las operaciones primitivas ofrecidas por *Pew*. En la sección 4 se propone un cálculo basado en las primitivas de *Pew*, y se presentan las reglas de transición que modelan dicho cálculo. En la sección 5 se muestra como especificar el comportamiento interactivo de componentes mediante ejemplos. Por último se dan algunas conclusiones y se indica el trabajo futuro.

2 *Pew*: Un Modelo Basado en Canales

2.1 Conceptos Generales

Pew [1] es un modelo de coordinación exógena, basado en canales, en el que a partir de un conjunto de canales provistos por el usuario, con comportamiento bien definido, se construyen de manera composicional coordinadores mas complejos llamados conectores. Cada conector impone un patrón de coordinación específico sobre las entidades que conecta. La figura 1 muestra un ejemplo.

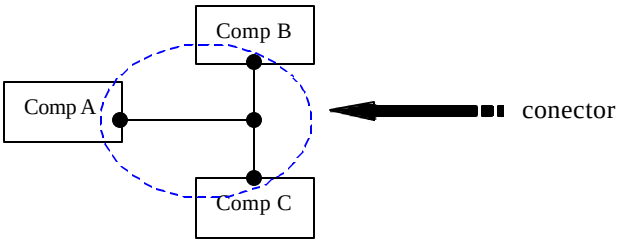


Figura 1: componentes y conectores

2.1.1 Componentes y Canales

En el contexto indicado un sistema consiste de un número de instancias de componentes que se ejecutan en 1 o más procesadores lógicos, comunicándose a través de conectores. Un componente es un tipo abstracto que describe las propiedades de sus instancias. Los conectores actúan como los componentes coordinadores del modelo IWIM y su función es coordinar las actividades de los demás componentes. La comunicación entre instancias de componentes toma lugar exclusivamente a través de los canales constituyentes de los conectores. Los conectores habilitan la comunicación entre instancias de componentes, refuerzan los patrones de coordinación exógenos, tienen semántica bien definida independiente de los componentes que coordinan, y son construidos como composición de conectores más simples. Un canal es el único medio primitivo de comunicación, y es considerado un conector atómico, permite comunicación anónima, punto a punto y de una vía. Las entidades activas de una instancia de componente se comunican con las entidades del exterior solamente a través de un conjunto de operaciones de entrada/salida que efectúan sobre el conjunto de extremos de canales conectados a él.

Un canal es una conexión punto a punto que ofrece dos extremos: fuente y destino, para que se pueda lograr la comunicación La comunicación es anónima, el componente sólo sabe a que canales esta conectado. El flujo de datos entre los componentes conectados por el canal es de una vía[6]. (Ver Fig. 2)

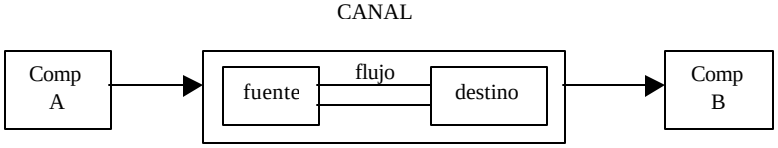


Figura 2: conexión de componentes

Cada extremo de canal puede estar conectado a un componente por vez. Cuando un extremo de canal es desconectado de una instancia de componente y conectado a otra, cambia dinámicamente la topología de conexiones del sistema.

Pew asume la disponibilidad de un conjunto de tipos de canales bien definidos, con un comportamiento dado por: el número de extremos fuente y destino, el esquema de orden, el tamaño del buffer, un filtro asociado, etc. el usuario puede definir tipos de canales adicionales indicando su comportamiento según las características mencionadas. Por ejemplo, un canal síncrono tipo *SyncDrain* es un canal que tiene dos extremos fuente y se utiliza para sincronizar la actividad en los nodos conectados a sus extremos, sin importar el valor que se escriba en el canal.

Todos los canales deben implementar las operaciones de la figura 3 con la misma semántica. El parámetro opcional *t* representa un valor de time out mayor o igual a 0. El parámetro *conds* en la operación *_wait* es una expresión booleana que combina condiciones primitivas predefinidas sobre extremos de canales (connect, empty, full, etc.). La operación *create* es la única que puede ser ejecutada directamente por una instancia de componente. Cuando se crea un canal se le asocia un patrón que regula las operaciones de entrada/salida sobre el canal. El resto de las operaciones deben ser utilizadas internamente por *Pew*.

Operaciones topológicas y de consulta

create(tipoCanal, filtro): crea un canal con el filtro especificado o * como su filtro, y retorna los identificadores de sus extremos fuente y destino. Los extremos del canal recientemente creado deben ser conectados explícitamente a una instancia de componente.

_forget(valorCanal): después de esta operación valorCanal ya no se refiere al extremo del canal que designaba

_connect([t,] valorCanal): conecta el extremo de cada canal especificado a la instancia de componente que contiene la entidad activa que ejecuta la operación.

_disconnect([t,] valorCanal): desconecta el extremo de canal especificado de la instancia de componente que contiene la entidad activa que ejecuta la operación.

_wait([t,] conds): suspende la entidad activa que ejecuta esta operación hasta que la condición especificada en el parámetro ***conds*** se haga verdadera para los extremos de canal especificados.

Operaciones de entrada/salida

_read([t,] inp[, v[, pat]]): suspende la entidad activa que ejecuta esta operación en espera del ingreso de un valor que mapee el patron indicado por ***pat***, en el extremo destino ***inp*** del canal. Es una operación no destructiva, despues de la lectura el valor se conserva en el canal.

_take([t,] inp[, v[, pat]]): suspende la entidad activa que ejecuta esta operación en espera del ingreso de un valor que mapee el patron indicado por ***pat***, en el extremo destino ***inp*** del canal. Es la versión destructiva de la operación de lectura. El valor en el extremo del canal es efectivamente consumido.

_write([t,] outp, v): suspende la entidad activa que ejecuta la operación hasta que se dan las condiciones para realizar la escritura del valor ***v*** en el extremo fuente ***outp*** del canal y la operación tiene éxito.

Figura 3: Operaciones primitivas de canales

2.1.2 Conectores

Un conector es un conjunto de extremos de canal y los canales que los conectan organizados en un grafo que tiene las siguientes características:

- Cada extremo de canal x solo puede estar conectado a un nodo N en cada momento.
- Cada nodo puede tener varios extremos de canal incidentes en él
- Cada canal interviniente en el conector es representado por un arco en el grafo

Dependiendo de los extremos de canal incidentes, cada nodo en un conector puede ser de uno de los tres tipos siguientes:

- *Nodo fuente*, cuando sólo inciden en él extremos fuente de canales.
- *Nodo destino*, cuando sólo inciden en él extremos destino de canales.
- *Nodo mixto*, cuando se da cualquier combinación de extremos de canales incidentes en él.

Sólo los nodos fuente y destino pueden estar conectados a componentes.

Las operaciones sobre nodos se organizan según lo muestra la figura 4. Las operaciones *connect* y *disconnect* sólo son aplicables con éxito a un nodo N que sea nodo fuente ó nodo destino. En caso de que haya mas de un extremo de canal incidentes en N de los que se pueda tomar un valor, las operaciones *read* y *take* eligen de manera no determinista. Las operaciones de composición de nodos se utilizan para la construcción de conectores y la reconfiguración topológica de los canales internos a un conector.

Operaciones topológicas y de consulta

forget(N): produce la aplicación de la operación _forget a cada extremo de canal coincidente en el nodo N.

connect([t,] N): conecta el extremo de cada canal coincidente en N a la instancia de componente que contiene la entidad activa que ejecuta la operación.

disconnect([t,] N): desconecta el extremo de canal coincidente en N de la instancia de componente que contiene la entidad activa que ejecuta la operación.

wait([t,] conds): suspende la entidad activa que ejecuta esta operación hasta que la condición especificada en el parámetro ***conds*** se haga verdadera.

Operaciones de entrada/salida

read([t,] N[, v[, pat]]): suspende la entidad activa que ejecuta esta operación en espera del ingreso de un valor que mapee el patron indicado por ***pat***, en algun extremo de canal incidente en N. Es una operación no destructiva, despues de la lectura el valor se conserva en el canal.

take([t,] N[, v[, pat]]): suspende la entidad activa que ejecuta esta operación en espera del ingreso de un valor que mapee el patron indicado por ***pat***, en algun extremo de canal incidente en N. Es la versión destructiva de la operación de lectura. El valor en el extremo del canal es efectivamente consumido.

write([t,] N, v): suspende la entidad activa que ejecuta la operación hasta que se dan las condiciones para realizar la escritura del valor ***v*** en cada extremo de canal conectado a N, en forma atómica.

Operaciones de composición y abstracción

Join (N1, N2): sólo tiene éxito si la instancia de componente que efectuó la operación tiene una conexión a alguno de los nodos indicados. El resultado de la operación es una mezcla destructiva y la generación de un nuevo nodo N que reunirá los extremos de canal coincidentes previamente en N1 y N2. La instancia de componente permanece conectada al nuevo nodo N sólo si estabá conectada a ambos nodos N1 y N2, y N no es un nodo mixto.

Split (N[, quoin]): genera un nuevo nodo N´ y divide los extremos de canal incidentes en N entre N y N´. La separación de los extremos de canales entre los nodos se realiza según lo especificado por el parámetro quoin.

Hide(N): es un mecanismo de abstracción. Se oculta el nodo N de manera que la topología de los canales coincidentes en N no pueda ser modificada. Como resultado de la aplicación de la operación sobre el nodo N, N no puede ser utilizado instancia de componente alguna.

Figura 4: Operaciones sobre nodos

2.2 Semantica de *Pew*

Pew espera que cada tipo de canal pueda proveer una implementación razonable de las operaciones primitivas. El comportamiento genérico de un canal c, con extremo fuente x_c y extremo destino y_c es definido a través de tres funciones:

- *Filter(c)*: filtro asociado al canal c en el momento de su creación
- *Offers (y_c, p)*: conjunto de pares <y_c, d> que indican las posibilidades de operaciones de lectura sobre yc, respetando p. Offers(x_c, p) = Ø
- *Takes (x_c, d)*: es una función booleana que indica si el item d respeta el filtro del canal c y el canal está en condiciones de aceptar una operación de escritura. *Takes(y_c, d)*= false, para todo dato d.

Intuitivamente todo fluye desde nodos fuente, a través de canales y nodos mixtos hacia nodos destino. Este comportamiento se logra a partir de la composición de canales en conectores y la semántica operacional de las operaciones de entrada/salida. En algunos casos, sin embargo, dependiendo del tipo de canal, algunos ítems pueden perderse.

Semántica de la operación write

Una operación *write(t, N, d)* con un time-out 0<=t<=∞, permanece pendiente hasta que:

- Su time-out t expira, y la operación falla
- *Accepts(N, d)*[1] = true y el conjunto de operaciones de escritura sobre cada extremo fuente de canal coindicente en N, ejecutadas atómicamente tienen éxito.

La operación *write* se efectua sólo para aquellos extremos de canal x_c para los cuales *takes(x_c, d)* = true.

Semántica de la operación take

Una operación *take(t, N, v, p)* con un time-out 0<=t<=∞, permanece pendiente hasta que:

- Su time-out t expira, y la operación falla
- ∃ d ∈ *offers(N)*[2] y existe un extremo de canal destino y_c, seleccionado no-determinísticamente para el cual d ∈ *offers(y_c)* y la operación *_take(0, y_c, v, p)* es ejecutada.

[1] La función *accepts (N , d)* se refiere a la posibilidad de éxito en la escritura del item de dato *d* en cada extremo fuente de canal x_c, coincidente en el nodo *N*, en el cual d respete el *filter(c)*

[2] La función *offers (N)* se refiere al conjunto de valores que se encuentran disponibles en los extremos de canales destino de N, o que se encuentran pendientes en un write si es un nodo fuente.

3 Composición de Canales en *Pew*

3.1 Ejemplo 1: Regulador de Lectura

Figura 5: ejemplo de composición de canales y conectores

El siguiente ejemplo muestra como se puede construir un conector simple a partir de canales y operaciones sobre canales. La configuración mostrada en la figura 5.a representa un conector *Regulador de Lectura* (Take-Cue Regulator) que es una de las formas mas básicas de coordinación exógena. Asumimos que los canales <a, b> y <c, d> son de tipo FIFO, y que el canal <e, f> es de tipo Sync. El número de ítems de datos que fluyen del canal <a, b> al canal <c, d> es el mismo que el número de operaciones *take* que tienen éxito en el extremo destino f del canal <e, f>. En esta situación la entidad activa de la instancia de componente que está conectada al nodo N (Node(f)=N) puede contar y regular el flujo de datos entre los canales <a, b> y <c, d> por el número de operaciones *take* efectuadas sobre N. Las instancias de componentes conectadas a Node(a) y Node(d) son ajenas al hecho de que su comunicación este siendo regulada o medida, y aún de que tal comunicación existe. En la figura 5.b se muestra la misma configuración, ahora encapsulada por efecto de la aplicación de la operación *hide* sobre el nodo común a los 3 canales. La abstracción lograda permite que la topología del conector sea inmutable, y puede ser utilizado como un "componente conector" que provee sólo los puntos de conexión de sus límites. La figura 6 muestra el código *Pew* para generar el conector de la figura 5.b. El valor de retorno de una llamada a esta función contiene las identidades de los nodos primarios de entrada y salida, y el nodo que actúa como regulador.

```
ReguladorL
        <a, b> = create (FIFO)
        <c, d> = create (FIFO)
        <e, f> = create (Sync)
        connect (b)
        connect (c)
        join (Node(b), Node(e))
        join (Node(b), Node(c))
        hide (Node(e))
        return <Node(a), Node(d), Node(f)>
```

Figura 6: código *Pew* para el conector Regulador de Lectura

2.3.2 Ejemplo 2: Conector Inhibidor

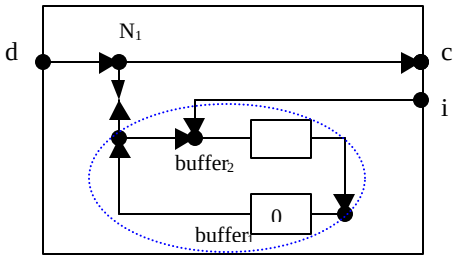


Figura 7: conector Inhibidor

El conector de la figura 7 tiene la función de permitir que los valores escritos en el extremo fuente *d* del canal fluyan libremente hacia el extremo destino *c* del otro canal, hasta que algún valor sea escrito en el extremo de canal *i*, lo que produce el corte del flujo de datos de *d* hacia *c*.

Este comportamiento se logra por las siguientes razones:

- El canal SyncDrain sincroniza la escritura en *d* con el valor inicial en el $buffer_1$ del canal del conector de serie (señalada con un círculo en la figura).
- Mientras no haya escrituras en *i*, el valor en el $buffer_1$ fluye libremente por cada escritura que haya en *d*.
- Cuando se produce una escritura en *i* se llena el $buffer_2$ y el valor en $buffer_1$ ya no puede circular por el conector de serie.
- Al bloquearse el valor en $buffer_1$, también se bloquea la operación de escritura en el nodo mixto N_1.

3. Cálculo basado en *Pew*

Representamos un sistema basado en componentes en el que intervienen los componentes C1, C2, ..., Cn con la expresión Π: C1 || C2 || ... || Cn

Asumimos una configuración inicial de conexiones a canales que definen la topología inicial de conexiones del sistema. Consideramos un conjunto *C_ID* de identificaciones de canales, con elemento característico *c*, y c_s, c_t como sus extremos fuente y destino respectivamente. Los canales considerados en este trabajo son asíncronos, sin filtro. Dado que los resultados presentados en este trabajo surgen de la fase inicial del análisis de la aplicación de *Pew* para la especificación del comportamiento interactivo de componentes, no se ha considerado la influencia de canales asíncronos especiales tipo AsyncDrain y AsyncSpout, ni de canales síncronos.

Para representar la configuración inicial seguimos lo propuesto en [5] . La interfaz observable de un componente es una tupla de la forma $\langle C, r, I(z), \varphi(r), \psi(r) \rangle$ en la cual $I(z)$ representa el invariante de bloqueo que por medio de aserciones especifica el posible comportamiento de bloqueo del componente *C*, y *z* puede ser cualquier extremo destino de canal perteneciente al conjunto C_ID. *C* denota el componente que se desea especificar, *r* representa los extremos de canal que intervienen en la configuración inicial, y $\varphi(r)$ y $\psi(r)$ denotan las pre y post condiciones del componente. Una aserción φ es definida como sigue:

$$\varphi ::= p(e_1, ..., e_n) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x(\varphi)$$

dónde p denota un predicado múltiple, aplicado sobre las expresiones e_i, y x representa una variable. Una expresión e es definida de la siguiente forma:

$$e ::= c_s \mid c_t \mid x \mid e\downarrow \mid e\downarrow_n \mid f(e_1, ..., e_n)$$

f representa un operador, y las expresiones $e\downarrow$ y $e\downarrow_n$ denotan la secuencia de valores asociados al extremo de canal e.

Para especificar el comportamiento interactivo de los componentes utilizamos un álgebra de procesos basado en las primitivas de comunicación de *Pew* y los operadores estándar de prefijo, composición alternativa y composición paralela. La sintaxis del álgebra es definida de la siguiente manera:

P::= 0 | A . P | P + P | P || P
A::= write(c_s, v) | take(c_t, v[, pat]) | read(c_t, v[, pat]) | wait(cond) |
connect(c) | forget(c) | τ

dónde τ representa cualquier acción interna que deba ejecutar el componente y de la cual nos abstraemos y 0 denota el proceso vacío. Los demás prefijos son las primitivas de *Pew* que nos interesa considerar.

A continuación se dan algunas de las reglas de transición que modelan la semántica operacional del álgebra presentada:

$$buf(c) \wedge write(c_s, v) \;.\; P \xrightarrow{write(c_s, v)} P \wedge buf(c) \cup \{v\}$$

$$\frac{P \xrightarrow{write(c_s, v)} P' \qquad Q \xrightarrow{take(c_t, d)} Q'}{P \| Q \xrightarrow{\alpha(write, take)} P' \| Q'}$$

La primera regla describe el comportamiento del prefijo write(c_s, v). La expresión buf(c) se refiere al contenido del buffer asociado al canal c, y la expresión buf(c) ∪ {v} representa genéricamente que el ítem v se ha agregado al buffer respetando el esquema de orden del tipo de canal. De manera equivalente se modela la semántica de los prefijos restantes. La segunda regla define la sincronización entre dos procesos que ejecutan acciones

complementarias de comunicación. El buffer conserva su estado. Las dos reglas siguientes representan la composición paralela y la composición alternativa respectivamente. Recordemos que el efecto de la operación *wait(conds)* es suspender la entidad activa que ejecuta la operación hasta que la condición especificada en el parámetro *conds* se haga verdadera.

$$\frac{P \xrightarrow{a} P' \qquad \alpha \neq wait(conds)}{P \| Q \xrightarrow{a} P' \| Q}$$

$$\frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'+Q}$$

En *Pew* los componentes se conectan a nodos fuente y destino, y un nodo representa el conjunto de extremos de canal que confluyen en él. Cada nodo al que pueda estar conectado un componente que no sea un componente conector propio de *Pew* es sólo un extremo de canal, razón por la cual por simplicidad en las operaciones consideramos directamente los extremos de canal.

5 Comportamiento Interactivo de Componentes en *Pew*

Como primer ejemplo consideramos un concentrador que combina entradas recibidas por 2 canales *e1* y *e2* y las reenvía por un canal de salida *s*. Consideramos los componentes adicionales Emisor y Receptor.
La especificación de los componentes es la siguiente:

Emisor (salida) = write (salida, item) . Emisor(salida)
Receptor (entrada) = take (entrada, d) . Receptor(entrada)
Concentrador (entrada1, entrada2, salida) =
take(entrada1, d) . write(salida, d) . concentrador(entrada1, entrada2, salida)
+
take(entrada2, d) . write(salida, d) . concentrador(entrada1, entrada2, salida)

El sistema es representado por la siguiente expresión:

Π: $\langle Concentrador, \{e1_t, e2_t, s_s\}, | e1_t \downarrow | + | e2_t \downarrow | = | s_s \uparrow |, empty(e1_k) \cup empty(e2_k) \cup empty(s_s) \rangle$
$\| \langle Emisor, \{e1_s\},, empty(e1_s) \rangle \| \langle Emisor, \{e2_s\},, empty(e2_s) \rangle \| \langle Receptor, \{s_t\},, empty(s_t) \rangle$

Las precondiciones son expresadas en términos de las condiciones primitivas sobre canales/nodos de *Pew*. No consideramos las postcondiciones dado que sólo nos interesa representar la topología inicial del sistema. La operación $| c \downarrow |$ da la longitud de la secuencia de valores escritos (ó leídos, si es un extremo destino) por él.

A continuación consideramos un ejemplo en el cual se producen cambios en la configuración del sistema de manera dinámica, sin perderse las conexiones iniciales. Se desea utilizar agentes para buscar información específica, por ejemplo precios de vuelos por internet. Los agentes consultan diferentes fuentes de información. Cada fuente de información tiene un punto de entrada donde se puede establecer los requisitos. Un agente conoce la ubicación de la fuente de información y la identidad de su punto de requerimiento. Esta información puede estar en una lista, o ser pasada como datos a través de canales, etc. El agente se mueve entre las distintas fuentes de información. El componente cliente interactúa con el agente solicitando información y recibiendo resultados. El agente por alguna acción interna recupera la identidad del canal por medio del cual debe comunicarse con la 1ra. fuente de información para obtener resultados. El proceso continúa con el agente recorriendo las distintas fuentes de información que conoce. Cada fuente de información al ser consultada devuelve sus resultados.
La especificación del sistema para un cliente, un agente y dos fuentes de información es:

Π: $\langle$Cliente,$\{e_s,, s_t\}$, $empty(e_s) \cup empty(s_t) \rangle \| \langle$Agente, $\{e_t,, s_s\}$, $empty(e_t) \cup empty(s_s) \rangle \|$
$\langle$Fuente,$\{f1_t,, i1_s\}$, $empty(f1_t) \cup empty(i1_s) \rangle \| \langle$Fuente, $\{f2_t,, i2_s\}$, $empty(f2_t) \cup empty(i2_s) \rangle$

Las conexiones iniciales son las que permiten la comunicación entre el cliente y el agente, y las propias de cada fuente de información. Las conexiones que permiten la comunicación entre el agente y las distintas fuentes de información son dinámicas y serán realizadas por parte del agente a medida que sea necesario. Se debe diferenciar los canales de entrada y salida de datos, identificados por su extremo destino y fuente respectivamente, debido a que en el cálculo de canales se consideran canales de un sentido.
El comportamiento del componente cliente puede ser descripto por el protocolo siguiente:

CLIENTE (salida, entrada) = write(salida, qry) . ESPERAR(entrada) (0
+
CLIENTE(salida, entrada))

ESPERAR(entrada) = PROCESAR || take(entrada, ans) .
VERIFICAR_ANS (write(salida, akn)
+
MAS_INFO(ans) . ESPERAR(entrada))

El cliente puede seguir realizando acciones internas mientras espera los resultados solicitados. Una vez recibidas las respuestas por parte del agente, puede decidir hacer una nueva consulta o terminar el proceso. El cliente permanece en espera de mas respuestas a su consulta hasta que recibe una marca de fin.
A continuación especificamos el comportamiento del componente agente, que espera el ingreso de algún pedido por su canal de entrada, y luego se conecta a las fuentes de información, una por vez, para obtener la información deseada. Al terminar con cada fuente de información libera sus puntos de requerimientos de manera que, por ejemplo, puedan ser utilizados por otro agente. Por la aplicación de la operación *write(entrada, 'FIN')* el proceso AGENTE indica al cliente que no dispone de mas información, a lo cual el proceso CLIENTE responde apropiadamente (*write(salida, akn)*).

AGENTE (salida, entrada) = take(salida, qry) . BUSCAR_INFO(entrada, qry) . take(salida, akn) .
AGENTE (salida, entrada)

BUSCAR_INFO(entrada, qry) = ELEGIR(f_s, i_k) . (write(entrada, 'FIN')
+
connect(f_s) . write(f_s, qry) . forget(f_s). connect(i_t) . take(i_t, ans) .
forget(i_t) . write(entrada, ans) . BUSCAR_INFO(entrada, qry))

El comportamiento de una fuente de información consiste sólo en esperar consultas y entregar una respuesta para cada consulta recibida.

FUENTE(f_t, i_s) = take(f_t, qry) . BUSCAR(qry, ans) . write(i_s, ans) . FUENTE(f_t, i_s)

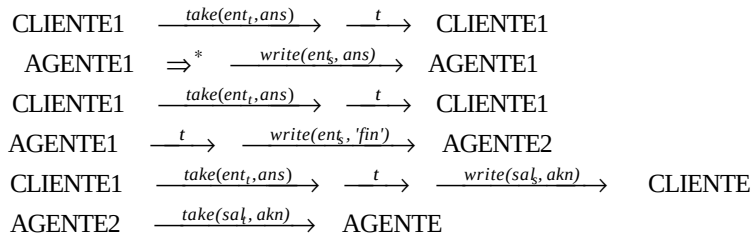
Intuitivamente la noción de compatibilidad entre componentes encierra la noción de ausencia de bloqueos en todas las ejecuciones alternativas de los procesos. Así también consideramos que dos componentes son compatibles si para cada acción posible ofrecida por uno de ellos existe una respuesta del otro y viceversa. Sobre el ejemplo podemos verificar que el proceso CLIENTE y el proceso AGENTE son compatibles. Por una primera transición aplicada sobre cada proceso obtenemos que el proceso CLIENTE queda en estado ESPERAR (representado por el proceso CLIENTE1) y el proceso AGENTE queda en estado BUSCAR_INFO (representado por AGENTE1).

$$\text{CLIENTE} \xrightarrow{write(sal_s, qry)} \text{CLIENTE1}$$
$$\text{AGENTE} \xrightarrow{take(sal_t, qry)} \xrightarrow{t} \text{AGENTE1}$$

En esta situación el AGENTE está en condiciones de conectarse con alguna fuente de información y obtener la información que le fuera solicitada por el cliente. Las acciones que compatibilizan al AGENTE con la FUENTE de información están representadas en la transición $\Rightarrow^*$ de la ecuación siguiente. El agente remite la información proporcionada por la primera fuente y se mantiene en el proceso AGENTE1.

$$\text{AGENTE1} \Rightarrow^* \xrightarrow{write(ent_s, ans)} \text{AGENTE1}$$

Este comportamiento del agente se repite mientras existan fuentes de información para consultar. Por su parte el cliente se mantiene en proceso CLIENTE1 hasta recibir la marca de 'FIN' de información, momento en el que vuelve al estado inicial del proceso CLIENTE. El proceso AGENTE2 representa la espera del agente del reconocimiento por parte del cliente del fin del flujo de información.



En las últimas dos transiciones, el componente CLIENTE elige dejar abierta la comunicación, y el AGENTE reacciona de manera acorde. Si la elección del CLIENTE hubiera sido la otra alternativa, el AGENTE está preparado para responder apropiadamente. El orden entre las transiciones de escritura y lectura por parte del CLIENTE y el AGENTE podría ser distinto sin que ello represente riesgo de bloqueo por la semántica de las operaciones write y take de *Pew* y el uso de canales asíncronos.

6. Conclusiones y trabajo futuro

Pew es un modelo de coordinación exógena, en el cual la comunicación entre procesos puede ocurrir solamente a través de canales, que se componen para generar conectores que actúan como coordinadores. En este trabajo, aunque se presenta la fase inicial del análisis de la aplicación de *Pew* en la interoperabilidad de componentes, a través de los ejemplos estudiados se puede observar la gran potencia expresiva del lenguaje. Actualmente estamos extendiendo el cálculo de procesos presentado aquí para considerar todas las características de *Pew* en cuanto a tipos de canales y operaciones de composición y abstracción. La variedad de canales disponibles -con posibilidades de ampliación-, sumado a la facilidad de construcción de conectores permiten que los demás modelos de coordinación, mencionados en apartados anteriores – los ports, streams y eventos de Manifold, y el espacio de tuplas compartido de Linda- puedan modelarse mediante *Pew*.

Nuestro trabajo futuro estará orientado a:

- definir formalmente una noción de compatibilidad entre procesos especificados en *Pew*, que formalice la noción intuitiva presentada en esta trabajo. En este sentido resulta interesante la propuesta de considerar los conectores como código de adaptación en la coordinación de componentes, y la habilidad de reconfiguración dinámica de la topología interna de los conectores.
- analizar la influencia de los distintos tipos de conectores y canales sobre la relación de compatibilidad y la especificación de protocolos de interacción,
- analizar la influencia del cambio de conectores sobre una relación ya establecida entre componentes.
- definir un modelo de interacción para la coordinación de componentes a partir de la integración de los resultados obtenidos en nuestro trabajo con *Pew* y los desarrollados en base al modelo Linda. Nos orientaremos a lograr el desarrollo de una herramienta automática para la comprobación de compatibilidad .

Referencias

1. F. Arbab, *A Channel-based Coordination Model for Component Composition*, Reporte de CWI, Centrum voor Wiskunde en Informatica, 16th European Conference on Object-Oriented Programming, 2002.
2. F. Arbab, *What do you mean, coordination?*, Bulletin of the Dutch Association for theoretical Computer Science, NVTI, pages 11-22, 1998.
3. F. Arbad, *The IWIM Model for Coordination of Concurrent Activities*, Proceedings First International Conference on Coordination Models, Languages and Applications (Coordination'96), pp 34-56.
4. F. Arbab, F.S. de Boer, M.M. Bonsangue, J.V. Guillen Scholten, *A channel-based coordination model for components*, Reporte de CWI, Centrum voor Wiskunde en Informatica, 16th European Conference on Object-Oriented Programming, 2001.
5. F. Arbab, F.S. de Boer, M.M. Bonsangue, *A logical interfaz description language for components*, Proceedings of Coordination 2000, Vol 1906 of Lecture Notes in Computer Science, pages 249-266, 2000.
6. F. Arbab, M.M. Bonsangue, F.S. de Boer, *A coordination Language for Mobile Components*, Proceedings of SAC 2000, ACM Press, pages 166-173, 2000.
7. H. Balen, *Distributed Object Architectures with CORBA*, Managing Object Technologies Series, Cambridge University Press 2000
8. K. Bergner, R. Grosu, A. Rausch, A. Schmidt, P. Scholz, M. Broy*Focusing on Mobility*, http://www4.informatik.tu-muenchen.de

9. A. Bracciali, A. Brogi, F. Turini, *Interaction patterns*, V Jornadas Iberoamericanas de Ingeniería de Software, 2001.
10. A. Brogi, E. Pimentel, A.M. Roldán, *Interoperabilidad de Componentes Software en Linda*, IDEAS 2002
11. A. Brogi, J.M. Jacquet, *On the expressiveness of coordination models*, Coordination Languages and Models:3rd. International Conference, Vol. 1594 of Lecture Notes, pages 134-149,1999.
12. C. Canal, *Un lenguaje para la especificación y validación de arquitecturas de software*, Ph.D. thesis, Depto Lenguajes y Ciencias de la Computación, Universidad de Málaga, 2001.
13. C. Canal, L. Fuentes, J.M. Troya, and A. Vallecillo, *Extending CORBA interfaces with pi-calculus for protocol compatibility*, Proceedings of the Technology of Object-Oriented Languages and Systems - TOOLS Europe 2000, IEEE Press 2000, pp. 208-225
14. W. Emmerich, *Engineering Distributed Objects*, London University, John Wiley & Sons, Ltd. 2000.
15. D. Gelernter and N. Carriero, *Coordination Languages and their significance*, Communications of the ACM, No. 35, pp. 97-107. 1992.
16. G.T. Leavens, M. Staraman, *Foundations of component based systems*, Cambridge University Press, 2000.
17. T. D. Maijler and O. Nierstrasz, *Beyond Objects: Components*, En Cooperative Information Systems: Current Trends and Directions, M.P. Papazoglou, G. Schlageter (Ed), Academic Press, Nov. 1997
18. G. Papadopoulos and F. Arbab, *Dynamic Reconfiguration in Coordination Languages*, Advances in Computers 46, Marvin V. Zelkowitz, Academic Press, 1998, pp. 329-400
19. W. Rubin, M. Brain, *Understanding DCOM*, Prentice Hall Series on Microsoft Technologies. 1999.
20. P. Wegner, *Interoperability*, ACM Computing Surveys, Vol. 28, No. 1, March 1996.