1 UniCon: Language for Universal Connector Support

El modelo en el que se basa UniCon nos proporciona una infraestructura para comprender la definición del lenguaje:

- Ofrece idiomas de abstracción normalmente utilizados por diseñadores. Por ejemplo, se distingue entre distintos tipos de elementos y se proporciona el soporte necesario para el análisis de tipos específicos.
- Especifica las propiedades de los paquetes así como las propiedades funcionales de los componentes; por ejemplo, se discrimina entre la funcionalidad proporcionada por un *filtro* de la que proporciona un *procedimiento* o función de un programa.
- Proporciona un elemento sintáctico denominado *conector* para la información acerca de las reglas de las interacciones entre componentes, tales como protocolos, intercambio de representaciones y especificación de formatos de datos para comunicación.
- Define una función de abstracción para hacer corresponder código o construcciones sintácticas de bajo nivel a construcciones de nivel más alto.
- Está abierto para incluir construcciones sintácticas externas al lenguaje y herramientas de análisis. Soporta la recolección y entrega de información relevante para una determinada herramienta–software y devuelve los resultados de dicha herramienta.

El lenguaje arquitectónico que vamos a definir soporta configuración del sistema e independencia de entidades, por tanto, *reusabilidad*, abstracción y análisis de las diferentes propiedades del sistema: desde *funcionalidad* hasta *seguridad* y *confiabilidad*.

1.1 Componentes y Conectores

Los *componentes* de UniCon se corresponden más o menos con las unidades de compilación de los lenguajes de programación convencionales y con otros objetos a nivel de usuario y achivos.

Los *conectores* sirven para mediar en las interacciones que se establecen entre *componentes*: establecen las reglas que gobiernan la interacción entre componentes y especifican cualquier mecanismo de implementación que se necesite de forma auxiliar a los propios componentes. Resulta conceptualmente útil pensar en un *conector* como el que define un conjunto de *roles*, que determinadas entidades incluidas en los componentes han de *jugar*.

El modelo de descripción de un sistema que propone UniCon se basa entonces en dos clases de elementos diferentes y distinguibles: (a) componentes y (b) conectores. Cada uno de estos elementos posee un *tipo*, una *especificación* y una *implementación*. La especificación define las unidades de asociación entre componentes y la implementacuón puede ser primitiva o compuesta. La figura 1 sugiere la estructura general del modelo sintáctico que soporta el lenguaje de descripción arquitectónico.

Los *componentes* constituyen la ubicación de los cálculos y del estado de las entidades.

Cada *componente* posee una especificación de interfaz que define sus propiedades:

- tipo o subtipo del componente (filtro, proceso, servidor, almacén de datos, etc.),
- funcionalidad,
- garantías acerca del cumplimiento de invariantes globales

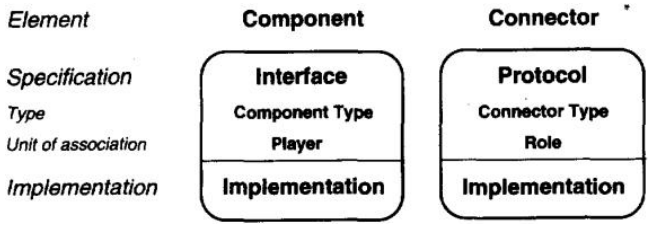

Figura 1: Estructura general de un lenguaje de descripción arquitectónico

Las entidades con nombre que aparecen en las interfaces de los componentes se denominan *players*. la interfaz de un componente ha de incluir la signatura, funcionalidad y propiedades de interacción de sus *players*.

En los conectores se ubica la definición de las relaciones entre componentes. Sirven para mediar en las interacciones entre componentes, pero no han de entenderse como *cosas en las que conectarse*, sino que más bien proporcionan las reglas para llevar a cabo las conexiones. Cada *conector* posee una especificación de protocolo que define sus propiedades:

- tipo o subtipo (llamada a procedimiento remoto (RPC), pipeline, broadcast, representación de datos compartidos, estándar para intercambio de documentos, evento),
- reglas acerca de los tipos de interfaces con los que trabaja,
- seguridades acerca de la interacción,
- compromisos acerca de la interacción, tales como ordenación relativa o rendimiento.

Las entidades con nombre que aparecen en el protocolo de un conector son los *roles* que han de satisfacerse. La interfaz incluye también reglas sobre cómo los difrentes *players* pueden adaptarse para desempeñar los *roles* junto con otras propiedades de la interacción.

Los componentes pueden ser *primitivos* o *compuestos*. Los componentes primitivos pueden ser implementados como código de un lenguaje de programación convencional, como *scripts* del shell de un sistema operativo; software desarrollado mediante una herramienta o aplicación informática, como una hoja de cálculo; u otros medios externos al lenguaje de descripción arquitectónico. Los componentes *compuestos* definen configuraciones con una notación independiente de los lenguajes de programación convencionales.

De una manera similar, los conectores pueden ser primitivos o compuestos. Pero los hay de diferentes clases: representaciones de datos compartidos, llamadas a procedimiento remoto (RPC), flujos de datos, estándares de intercambio de documentos , protocolos de red, etc. Los conectores

primitivos se pueden implementar de diferentes maneras: mecanismos internos a los lenguajes de programación (RPCs o datos compartidos); funciones del sistema operativo; como código de librería en lenguajes de programación convencionales; como datos compartidos o como entradas en tablas de enrutamiento o de tareas, etc. Los conectores *compuestos* pueden aparecer también bajo cualquiera de las formas anteriores, pero todavía no se dispone de una manera aceptada para definirlos.

La figura 2 representa la implementación heterogénea de un *pipeline*, que utiliza distintos tipos de conectores y componentes. También muestra la *implementación compuesta* de uno de los filtros.

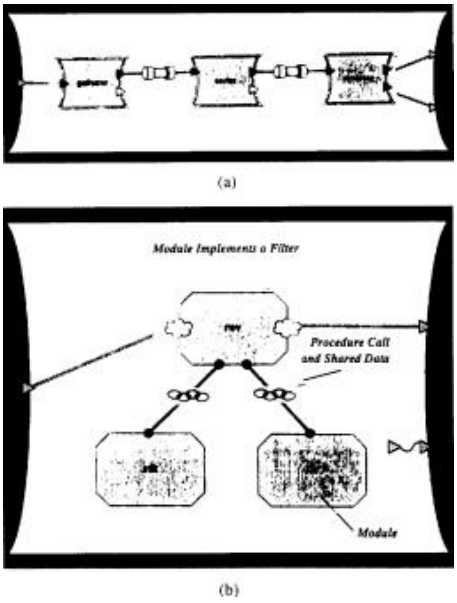

Figura 2: Implementación heterogénea de un *pipeline* utilizando *pipes* y funciones

1.2 Semántica de UniCon

UniCon hace hincapié en los distintos aspectos estructurales de una arquitecfura de software. Se trata de una notación de más alto nivel y más general que otros mecanismos actualemnte existentes para describir la composición de sistemas software. UniCon ha relizado algunas simplificaciones para poder obtener un modelo de sistema implementable con facilidad:

- Ofrece componentes compuestos pero no conectores compuestos
- Tiene abstraciones propias para modelar componentes y conectores, pero no se pueden agregar nuevas abstracciones.
- Los únicos componentes primitivos de los que se dispone en la actualidad son las *unidades de compilación* de los diferentes lenguajes, preferiblemente C.
- La definición sintáctica del lenguaje todavía no ha sido depurada y puede resultar un poco *verbosa* hasta que uno se acostumbra, sobre todo cuando se intenta utilizar para realizar conexiones entre módulos de procedmeintos y datos sin cambiar los identificadores.

Tanto los conectores como los componentes poseen los siguientes elementos básicos:

- un nombre;
- una especificación, denominada *Interface* para los componentes y*Protocol* para los conectores;
- un tipo componente o conector;
- un conjunto de asertos globales como una lista de propiedades (*Property List*);
- una colección de *unidades de asociación*: para los componentes se denominan *Players* y para los conectores *Roles*; en las listas de propiedades se suelen especificar en las listas de propiedades anteriormente aludidas;
- una implementación.

  Cada una de las definiciones anteriores puede entenderse sólo en los términos de su especificación y las propiedades de un sistema–software han de poderse derivar de la especificación de sus componentes y conectores. Cada una porporciona una plantilla que se intancia cuando se utiliza. Se utiliza la información acerca de las diferencias entre componentes, conectores, players, y roles para la comprobación de tipos en las descripciones de sistemas.

  La definición de la semántica de lenguaje UniCon se ha organizado en la relación de ejercicios de acuerdo con la siguiente estructura: descripción de los componentes (ejercicios 3–6), descripción de los conectores (ejercicio 8) y descripción de los denominados *componentes compuestos* (ejercicio 9).

Referencias

[1] Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M., Zelesnik, G. Abstractions for Software Architecture and Tools to Support Them. IEEE Transactions On Software Engineering,v.21,4, 1995.