Una Visión General a los Lenguajes de Descripción Arquitectónica

Informe Técnico LSI-2001-01

(23 de octubre de 2001)

OCTAVIO MARTÍN DÍAZ

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla, España
Tel/Fax: +34 95 4557139
e-mail: `octavio@lsi.us.es`

Resumen

El presente informe técnico es una breve introducción en español a los lenguajes de descripción arquitectónica, basada principalmente en los trabajos de [8] y [11], donde hacemos un repaso de las principales características que todos ellos deberían tener e intuimos hasta qué punto son tenidas en cuenta o no por las corrientes actuales. A continuación, introducimos el marco de clasificación y comparación de ADL's definido por [11] y, basándonos en él, presentamos los ADL's que hemos considerado más interesantes. Finalmente, repasamos los últimos enfoques surgidos con vistas a especificar las propiedades no funcionales a nivel arquitectónico, uno de los aspectos que se quedaron al margen cuando se definieron los primeros ADL's, enfocados principalmente hacia la funcionalidad. Al no ser una compilación original remitimos directamente al lector que desee profundizar en estos temas a las referencias bibliográficas que aparecen al final del documento.

Capítulo 1

Introducción

La obtención de un **prototipo arquitectónico** durante la fase de diseño es muy importante porque a partir de las descripciones de vistas, componentes e interconexiones existentes entre ellos podremos obtener un modelo que permita los siguientes procesamientos, ordenados de menor a mayor dificultad: **(1)** la *compilación* para obtener un esqueleto ejecutable del sistema informático que podremos ir rellenando hasta conseguir el sistema completo, **(2)** la *simulación* del modelo con vistas a la comprobación de propiedades de calidad incluidas las propiedades operativas funcionales y, finalmente, **(3)** la *ejecución* directa del modelo, por lo que obtendríamos un lenguaje de altísimo nivel que ayudaría en gran medida al desarrollo de los sistemas informáticos.

Tradicionalmente, cualquier modelo arquitectónico se ha representado mediante diagramas en los que, por regla general, la naturaleza de los componentes y sus propiedades, la semántica de las conexiones y el comportamiento del sistema en su conjunto están pobremente definidas. La ausencia de respuestas a preguntas tales como *¿qué son los componentes? ¿qué hacen y cómo se comportan? ¿dependen de otros componentes? ¿cómo son las conexiones? ¿qué significan exactamente? ¿cuáles son los mecanismos que podemos utilizar para llevarlas a cabo?* complica sobramanera cada uno de los procesamientos descritos anteriormente. Las respuestas a tales preguntas serán mucho más sencillas si utilizamos alguna forma estándar de representación formal que sirva a los propósitos fundamentales por los que queremos diseñar una arquitectura software.

Un **Lenguaje de Descripción Arquitectónica** (**ADL**[1]) es precisamente un enfoque lingüístico a la representación formal de una arquitectura [8]. Una arquitectura software (es un nivel de diseño que) incluye la descripción de elementos constituyentes de los sistemas, las interacciones entre tales elementos, los patrones que guian su composición y las restricciones sobre estos patrones [13]. Un ADL para aplicaciones software se centra en la estructura de alto nivel de la aplicación en su conjunto y no en los detalles de implementación de cualquiera de sus módulos fuentes específicos [14]. Un ADL debe modelar explícitamente los componentes, los conectores y las diversas configuraciones; más aún, para que sea realmente utilizable y útil debe facilitar un soporte de herramienta para el desarrollo arquitectónico [11]. Por tanto, un ADL es un lenguaje que proporciona características para modelar la arquitectura conceptual de un sistema software, distintiva de la implementación del sistema: los ADL's proporcionan tanto una sintaxis concreta como un marco conceptual para la caracterización de arquitecturas donde éste refleja las características del dominio para los que el ADL está enfocado y/o el estilo

[1] **A**rchitecture **D**escription **L**anguage

arquitectónico. El marco subsume la teoría semántica subyacente del ADL [6].

En los siguientes capítulos abordaremos cada uno de los aspectos más interesantes según el siguiente orden: **(1)** las caractarísticas generales de los ADL's de hoy en día, **(2)** el marco de clasificación y comparación de ADL's de [11] junto a una introducción a algunos de los ADL's más representativos y **(3)** los criterios fundamentales para la selección de un ADL.

Capítulo 2

Características Generales de los ADL's de Hoy en Día

Existe una gran variedad de ADL's, tanto enfocados hacia un dominio particular como aquellos que son de propósito general, que proceden bien de los ambientes industriales bien de los grupos de investigación académicos aunque sólamente unos cuantos de ellos son realmente productos comerciales. Sin embargo, lo más corriente y evidente es la ausencia general de una experiencia profunda en la aplicación en el mundo real de los diversos ADL's existentes. En general, podemos encontrar que casi todos los ADL's tienen la mayor parte de las siguientes características:

- Una sintáxis gráfica asociada a la sintaxis textual y, en cualquier caso, encontramos una semántica bien definida.
- Un enfoque hacia el modelado de sistemas distribuidos.
- Un escaso apoyo en la captura de la "*rationale*" del diseño y/o su historia: quizás sólo llegan a tener algunos mecanismos de notación de propósito general.
- Una gestión de los flujos de datos y de control como mecanismos de interconexión, mientras que otras clases de conexión más específicas están menos toleradas.
- Una capacidad para representar diversos niveles jerárquicos de detalle y gestionar las distintas vistas de una arquitectura.

Sin embargo, existen muchas otras características que difieren ampliamente de un ADL a otro, entre las que encontramos:

- La capacidad para gestionar constructores de tiempo real a nivel arquitectónico.
- La capacidad para respaldar la especificación de estilos arquitectónicos particulares.
- La capacidad de personalización (casi nula en todos ellos) de las reglas de consistencia y completitud que incorporan.
- La capacidad para realizar un análisis de validez.
- La capacidad para gestionar la variabilidad o instancias diferentes de una misma arquitectura: no soportan las arquitecturas de línea de productos.

Capítulo 3

Un Marco para la Clasificación y Comparación de ADL's

[11] define un marco en el que poder clasificar y comparar los diferentes ADL's existentes de manera que se ponga en claro cuáles son sus faltas y virtudes. De esta manera, los ADL's que se definan en el futuro podrán partir de una base de información que ya comienza a estar muy contrastada. En general, los ADL's deben representar la información correspondientes a las diversas estructuras arquitectónicas que pueden aparecer en la definición de un sistema [8] donde podemos encontrarnos con varias categorías principales de información:

Información estática La capacidad para declarar un componente y dar su tipo es una característica esencial de un ADL que proporciona estructuras estáticas. Sin embargo, la definición de la semántica de un componente o conector puede ser muy difícil. El problema clásico es el momento temporal de enlace o fijación de un elemento: ¿compilación o diferido?

Información dinámica La comprensión de cómo interaccionan varios componentes en tiempo de ejecución es esencial para comprender el comportamiento de un sistema. Una estructura dinámica frecuentemente necesita de un simulador para realizar el análisis. La información proporcionada por la simulación es dependiente del tipo de análisis a realizar: en concreto hay que obtener una buena estimación de los parámetros (información sobre los recursos). Los componentes del simulador se pueden obtener de manera automática.

Calidad La satisfacción de las propiedades no funcionales, o de calidad, constituyen una nueva perspectiva de investigación: no sólo hace falta que una arquitectura software responda a sus requisitos funcionales, sino también a aquellos otros que fueron indicados por los participantes en el proyecto software y que, habitualmente, no son tenidos en cuenta hasta las últimas fases del diseño cuando, por regla general, resultan más complicadas las modificaciones a aplicar a una arquitectura para que satisfaga los requisitos que no cumple.

En la figura 3.1 podemos encontrar el esquema de este marco. Encontramos, en primer lugar, los bloques principales (y típicos) de construcción para cualquier descripción arquitectónica: **(1)** los *componentes*, **(2)** los *conectores* y **(3)** las *configuraciones arquitectónicas* que también se conocen simplemente como *configuraciones* o *topologías*. Aunque cualquier ADL sea, en principio, independiente de cualquier tipo de herramienta, el apoyo de éstas lo hará más útil

- ADL
  - Características de modelado arquitectónico
    - Componentes
      - Interfaz
      - Tipos
      - Semántica
      - Restricciones
      - Evolución
      - Propiedades no funcionales
    - Conectores
      - Interfaz
      - Tipos
      - Semántica
      - Restricciones
      - Evolución
      - Propiedades no funcionales
    - Configuraciones
      - Comprensibilidad
      - Composición
      - Refinamiento y Traza
      - Heterogeneidad
      - Escalabilidad
      - Evolución
      - Dinamismo
      - Restricciones
      - Propiedades no funcionales
  - Apoyo de Herramientas
    - Especificación activa
    - Vistas múltiples
    - Análisis
    - Refinamiento
    - Generación de la implementación
    - Dinamismo

Figura 3.1: Marco para la clasificación y comparación de ADL's [11].

y aprovechable. Por supuesto, esta lista no es estática sino que puede variar e incluir otros muchos elementos conforme la investigación avance o bien se utilicen en uno u otro dominio de información.

3.1 Características de Modelado Arquitectónico

3.1.1 Modelando los componentes

Un **componente** en una arquitectura es una unidad de procesamiento o un almacenamiento de datos, que pueden ser tan pequeños como simples procedimientos o tan grandes como aplicaciones completas. Cada componente puede requerir su propio espacio de ejecución o de datos, o puede compartirlo con otros componentes. Entre las características que se pueden utilizar para compararlos tenemos:

Interfaz La interfaz de un componente es un conjunto de puntos de interacción entre sí y el mundo exterior, especificando los servicios (mensajes, operaciones y variables) que proporciona. Un ADL debe proporcionar información sobre las necesidades de un componente respecto a otros: por tanto, un interfaz define los compromisos de procesamiento que un componente puede realizar y las restricciones sobre su utilización.

Tipos Los tipos de componentes son abstracciones que encapsulan la funcionalidad en bloques reutilizables. En general, estos pueden parametrizarse, exportarse a otras arquitecturas, definir múltiples componentes del mismo. Ayuda a la comprensibilidad y facilita el análisis de cualquier arquitectura.

Semántica La semántica de un componente es un modelo de alto nivel de su comportamiento, que se necesita para realizar análisis, hacer cumplir las restricciones y asegurar mapeos consistentes entre diferentes niveles de abstracción de una arquitectura.

Restricciones Las restricciones de un componente son *asserts* (afirmaciones) sobre el sistema (o algunas de sus partes) de manera que en caso de violación hacen al sistema inaceptable (o menos deseable) para alguno de los actores participantes.

Evolución Los componentes pueden evolucionar continuamente. Para que esto se pueda hacer de forma sistemática, los ADL's emplean técnicas tales como el subtipado de los tipos de componentes y el refinamiento de sus características.

Propiedades no funcionales Las propiedades no funcionales de un componente (por ejemplo, la seguridad, la confidencialidad, el rendimiento, la portabilidad) no pueden derivarse directamente de la especificación de su comportamiento. Su especificación (a nivel arquitectónico) es necesaria para permitir la simulación del comportamiento, realizar análisis, hacer cumplir las restricciones, mapear las implementaciones de componentes a los procesadores y ayudar en la gestión de proyectos.

3.1.2 Modelando las conexiones

Los **conectores** son bloques de construcción arquitectónicos utilizados para modelar las interacciones entre los componentes y las reglas que las gobiernan. Al contrario que los componentes, los conectores no pueden corresponderse a unidades de compilación en la implementación del sistema. Podemos encontrar muchos modelos de conectores tales como protocolos cliente-servidor, tuberías, paso de mensajes, conexiones entre múltiples participantes y otras muchas. Entre las características que se pueden utilizar en este marco para compararlos tenemos:

Interface La interfaz de un conector es un conjunto de puntos de interacción entre el conector y los componentes y otros conectores. Al no ser unidades de procesamiento, el interfaz exportado estará formado por los servicios proporcionados por los componentes a los que está conectado. Estos interfaces permiten la conectividad adecuada de los componentes y sus interacciones en una arquitectura y, por tanto, el razonamiento sobre las configuraciones arquitectónicas.

Tipos Los tipos de conectores son abstracciones que encapsulan las decisiones para la comunicación, coordinación y mediación de componentes, permitiendo que estos protocolos (que pueden llegar a ser muy complejos) puedan reutilizarse en la propia arquitectura o exportarse a otras.

Semántica La semántica de un conector es un modelo de alto nivel de su comportamiento, es decir, las especificaciones de protocolos de interacción (independientes de los procesamientos). Todo ello se necesita para permitir el análisis de las interacciones entre componentes, el refinamiento consistente de arquitecturas a través de los niveles de abstracción y hacer cumplir las restricciones de interconexión y comunicación.

Restricciones Las restricciones de los conectores aseguran la observación a los protocolos de interacción intencionados, establecer interdependencias entre conectores y hacer cumplir los límites de tratamiento.

Evolución La evolución de un conector se define como la modificación de (un conjunto de) sus propiedades. Como se ha dicho anteriormente, las interacciones entre los componentes de cualquier arquitectura vienen dictadas por protocolos complejos que pueden cambiar o expandirse. Más aún, tanto los componentes como las conexiones pueden evolucionar. Los ADL's pueden facilitar esta evolución con técnicas tales como el filtrado de información incremental, subtipado y refinamiento de los conectores existentes.

Propiedades no funcionales Las propiedades no funcionales de un conector no pueden derivarse completamente a partir de la especificación de su semántica. Representan requisitos (adicionales) para la implementación correcta de los mismos. Si modelamos estas propiedades se permite la simulación de su comportamiento, el análisis de los conectores, el cumplimiento de las restricciones y la selección de conectores (de terceros) y sus mapeos a los procesadores.

3.1.3 Modelando las configuraciones

Una **configuración arquitectónica** o **topología** es un grafo conexo de componentes y conectores describiendo una estructura arquitectónica. Esta información se necesita para determinar si están conectados los componentes adecuados, sus interfaces concuerdan, los conectores permiten una comunicación adecuada y la combinación de las semánticas responden al comportamiento deseado. Conjuntamente con los modelos de componentes y conectores, la descripción de configuraciones permiten la evaluación de los aspectos de concurrencia y distribución de cualquier arquitectura (por ejemplo, posibilidad de abrazos mortales o inanición, rendimiento, fiabilidad, seguridad y otros). Estas descripciones también permiten el análisis de arquitecturas respecto al seguimiento de las heurísticas de diseño (por ejemplo, los enlaces directos de comunicación entre componentes obstaculizan la facilidad de evolución de una arquitectura) y de las restricciones impuestas por los estilos arquitectónicos (por ejemplo, los enlaces directos de comunicación entre componentes se desestiman). Entre las características que se pueden utilizar en este marco para compararlos se engloban en tres categorías generales:

- ***Cualidades de la descripción de la configuración***:

  Comprensibilidad de las especificaciones Un papel a desempeñar por cualquier arquitectura software es servir como un conducto inicial para los diferentes actores participantes en el proyecto y facilitar la comprensión de (familias de) sistemas a un gran nivel de abstracción. Por tanto, los ADL's deben modelar la información estructural (o topológica) con una sintaxis simple y comprensible. A ser posible, la estructura de un sistema debería estar clara sólo desde un punto de vista topológico, es decir, sin tener en cuenta los detalles de especificación de componentes y conectores.

  Facilidad de composición La composición jerarquizada es un mecanismo que permite la descripción arquitectónica de sistemas software con diferentes niveles de detalle:

tanto estructuras como comportamientos complejos (incluso descripciones arquitectónicas completas) pueden abstraerse hacia un único componente o conector sencillo para utilizarse a un nivel de abstracción superior.

Refinamiento y facilidad de realización de trazas Además, los ADL's deben permitir el refinamiento consistente y correcto de cualquier arquitectura en sistemas ejecutables y la traza de los cambios a través de los diferentes niveles de refinamiento arquitectónico.

Heterogeneidad Un objetivo de las arquitecturas software es facilitar el desarrollo de sistemas de gran escala, preferiblemente con componentes y conectores ya existentes con diferentes grados de granularidad y posiblemente especificados en diferentes lenguajes de modelado formal e implementados en diferentes lenguajes de programación con requisitos distintos sobre el sistema operativo y soportando diferentes protocolos de comunicación. Por lo tanto, es importante que un ADL sea *abierto*, es decir, que proporcionen ayudas para la especificación y desarrollo arquitectónico con componentes y conectores heterogéneos.

- ***Cualidades del sistema descrito***:

Heterogeneidad (Vease la explicación en el ítem anterior).

Escalabilidad Las arquitecturas están enfocadas a proporcionar a los desarrolladores las abstracciones necesarias para arreglárselas con las cuestiones de complejidad y tamaño del software. Por lo tanto, los ADL's deben apoyar de manera directa la especificación y desarrollo de grandes sistemas donde haya probabilidad de que crezcan más aún.

Evolución Los nuevos sistemas software raramente proporcionan una funcionalidad sin precedentes de manera completa sino que existen muchas "*variaciones sobre el mismo tema*". Una arquitectura se desarrolla para *reflejar* y *permitir* la evolución de una familia de sistemas software. La evolución es por sí misma una de las actividades de desarrollo software más costosas, lo que la convierte en un aspecto clave del desarrollo basado en arquitecturas. Los ADL's pueden y deben aumentar esta capacidad a nivel de componentes y conectores con características tales como la adición incremental, eliminación, sustitución y reconexión en una configuración.

Dinamismo Si la evolución alude a cambios "*fuera de línea*" de una arquitectura, el dinamismo se refiere a modificaciones realizadas mientras el sistema se está ejecutando. Esta característica es muy importante para multitud de *sistemas críticos* (por ejemplo, un sistema de control de tráfico aéreo) donde la parada y reinicio de tales sistemas para realizar modificaciones puede incurrir en retrasos prolongados, incremento de costes y riesgos inaceptables. Para que un ADL tenga esta cualidad, es necesario proporcionar características específicas de cambios de modelado dinámico y técnicas para efectuarlos mientras el sistema se ejecuta.

- ***Propiedades del sistema descrito***:

Dinamismo (Vease la explicación en el ítem anterior).

Restricciones Las restricciones que representan dependencias en una configuración complementan las específicas de componentes y conectores. De hecho, muchas restricciones globales se derivan directamente de las locales. Por ejemplo, las restricciones sobre configuraciones válidas pueden expresarse como restricciones de interacción entre los componentes y conectores constituyentes que, a su vez, se expresan mediante sus interfaces y protocolos; el rendimiento de un sistema descrito mediante una configuración dependerá del rendimiento de cada elemento arquitectónico individual; la seguridad de una arquitectura es una función de la seguridad de sus partes constituyentes.

Propiedades no funcionales Ciertas propiedades no funcionales están a nivel de sistema, frente a las propiedades de componentes o conectores individuales. Las propiedades no funcionales a nivel de configuración se necesitan para seleccionar componentes y conectores adecuados, realizar análisis, hacer cumplir las restricciones, mapear los bloques arquitectónicos a los procesadores y ayudar a la gestión del proyecto.

3.2 Algunos ADL's Representivos

En las siguientes secciones repasamos las principales características de algunos de los ADL's más conocidos. Cada uno de ellos se centra en un dominio concreto de aplicación, se basa en algún estilo arquitectónico concreto o algún aspecto específico que pretende modelar.

3.2.1 ACME

El foco de atención de ACME [6] se centra en el mapeo de especificaciones arquitectónicas desde un ADL a otro, permitiendo la integración de las herramientas de soporte asociadas a distintos ADL's. Basado en componentes e independiente de la implementación, los puntos de interface de los componentes se denominan *ports* (puertos). El sistema de tipos es extensible, permitiendo incluso la parametrización mediante plantillas. Sin embargo, no soporta de manera directa la especificación semántica, aunque sí permite la utilización de modelos semánticos de otros ADL's. Las restricciones sólo se pueden plantear vía las interfaces. Respecto a la propiedad de evolución, sólo permite la claúsula *extend* para especificar un subtipado estructural. No opera directamente con propiedades no funcionales, aunque sí que se pueden utilizar listas de propiedades a modo de información adicional.

Los conectores se modelan de forma explícita con un sistema de tipos extensible que está basado en protocolos y permitiendo la parametrización vía de plantillas. Sus puntos de interfaz se denominan roles. Aunque no estaba previsto inicialmente, la especificación semántica se puede realizar utilizando modelos semánticos de otros ADL's vía las listas de propiedades. Las restricciones se describen vía las interfaces, siendo estructural para las instancias de tipos. Respecto a la evolución, la clásula *extend* que permite un subtipado estructural. No opera directamente con propiedades no funcionales, aunque sí que se pueden utilizar listas de propiedades a modo de información adicional.

La configuración se modela de forma explícita mediante *attachments* (adjuntos), siendo la composición proporcionada vía de plantillas, representaciones y mapas. Estos últimos también se utilizan en la traza y refinamiento. La comprensión es explícita mediante especificaciones textuales concisas. La heterogeneidad se consigue mediante listas de propiedades abiertas, requieriéndose mapeos explícitos entre ADL's. La escalabilidad se ayuda de configuraciones explícitas aunque obstaculizada por el número fijo de roles. La evolución también se consigue mediante configuraciones explícitas, como familias de primera clase. No se tiene en cuenta la propiedad del dinamismo. Las restricciones se describen mediante puertos que sólo pueden adjuntarse a roles y viceversa. Las propiedades no funcionales se pueden indicar mediante listas de propiedades, aunque sin operar sobre ellas.

El soporte de herramientas de ACME se limita a la gestión de múltiples vistas de manera textual, describiendo vistas arquitectónicas en términos de plantillas de alto nivel, además de constructores básicos. Por ejemplo, define unos elementos denominados “weblets” en la arquitectura ACME-Web.

3.2.2 Aesop

El foco de atención de Aesop [4, 5] se centra en la especificación propiamente dicha de arquitecturas. Basado en componentes e independiente de la implementación, con un sistema de tipos extensible donde los puntos de interfaz se modelan como puertos de entrada y salida. De manera opcional, se pueden utilizar diversos lenguajes para especificar la semántica. Las restricciones se describen vía los interfaces y con invariantes. La evolución se permite mediante un subtipado que preserva el comportamiento. Respecto a las propiedades no funcionales, sólo permite que se asocien textos arbitrarios a los componentes.

Los conectores se modelan de forma explícita con un sistema de tipos extensible basado en protocolos. Los puntos de interfaz se denominan roles. La especificación semántica es opcional mediante Wright. Las restricciones se describen vía los interfaces y la semántica mediante invariantes. La evolución se permite mediante un subtipado que preserva el comportamiento. Respecto a las propiedades no funcionales, sólo permite que se asocien textos arbitrarios a los componentes.

La configuración se modela de forma explícita mediante *configuraciones* propiamente dichas, siendo la composición proporcionada vía de representaciones. La comprensión es explícita con especificaciones gráficas concisas, utilizando jerarquías de tipos paralelas para la visualización. Respecto a la heterogeneidad, se permiten múltiples lenguajes para el modelado semántico, soportando el desarrollo final en C. La escalabilidad se ayuda de configuraciones explícitas, aunque obstaculizada por el número fijo de roles. La evolución también se consigue mediante configuraciones explícitas pero no soporta las arquitecturas parciales. Las restricciones se describen mediante puertos que sólo pueden adjuntarse a papeles y viceversa, y también mediante invariantes programables. No se tienen en cuenta las propiedades de facilidad de traza y refinamiento, tampoco el dinamismo ni las propiedades no-funcionales.

El soporte de herramientas de Aesop tiene una especificación activa, en el sentido de que tiene un editor sintáctico de componentes donde una visualizador de clases invoca editores externos especializados. Se permiten múltiples vistas tanto de manera textual como gráfica, con visualizaciones específicas del estilo por un lado y tipos de componentes y conectores distinguidos por sus íconos por otro. Respecto al análisis que podemos realizar, encontramos diversos *parsers* (analizadores): compiladores específicos del estilo, chequeadores de tipos, chequeadores de ciclos, chequeadores de conflictos de recursos y viabilidad de la planificación. Respecto a la generación de implementaciones, se restringe sólo a un código de pegamento para los constructores del sistema, realizado en C, para el estilo arquitectónico de tubos y filtros.

3.2.3 MetaH

El foco de atención de MetaH [3, 15] se centra en el dominio de las arquitecturas de guía, navegación y control. Excepcionalmente, los componentes con los que estamos tratando en este ADL son **procesos**, por lo que se restringe la futura implementación. Los puntos de interfaz son también puertos, distinguiendo entre síncronos y asíncronos. Tiene un conjunto predefinido de tipos. La semántica se puede especificar con ControlH para modelar algoritmos en su dominio específico. Las restricciones se describen vía los interfaces y la semántica, con diversos modos y atendiendo a las propiedades no funcionales, aunque éstos se restringen exclusivamente a aquellos que son necesarios para el análisis de planificación de tiempo real, fiabilidad y seguridad. No se tiene en cuenta la propiedad de la evolución.

Los conectores se modelan en línea y, opcionalmente, se les puede dar un nombre. Aunque no soporta un sistema de tipos, sí disponemos de tres clases generales de conectores: puertos, eventos y equivalencias. No permite la especificación de semántica, restricción, evolución y propiedades no funcionales.

La configuración se modela en línea mediante conexiones, siendo la composición soportada vía de macros. La comprensión se realiza mediante especificaciones textuales en línea con muchos detalles de los conectores y también se proporciona una notación gráfica. Respecto al refinamiento y la realización de trazas, se soporta la generación de sistemas y restricciones de implementación. Respecto a la heterogeneidad, se soporta el desarrollo en ADA, requiriéndose que todos los componentes contengan un bucle de despacho de procesos. La escalabilidad se ve obstaculizada por las configuraciones en línea y tampoco la evolución se ve favorecida por el mismo motivo, ni tampoco se soportan arquitecturas parciales. Las *aplicaciones* se restringen mediante propiedades no funcionales, entre las cuales sólo se incluyen propiedades tales como el procesador de ejecución y el período de reloj. No se tiene en cuenta la propiedad del dinamismo.

El soporte de herramientas de MetaH tiene una especificación activa, ofreciendo un editor gráfico que requiere una corrección de errores una vez que los cambios de la arquitectura son aplicados, y restringiendo la selección de propiedades de componentes vía de menús. Se permiten múltiples vistas, tanto de manera textual como gráfica, cuyos tipos de componentes se distinguen con íconos. Respecto al análisis que podemos realizar, disponemos de diversos analizadores y compiladores, algunos específicos para el análisis de planificación, fiabilidad y seguridad. Respecto al refinamiento, hay un compilador cuyos componentes primitivos se implementan en lenguajes de programación tradicionales. La generación de implementaciones sigue el enfoque DSSA, generando el compilador código ADA. Tampoco se tiene en cuenta el dinamismo en este nivel.

3.2.4 Rapide

El foco de atención de Rapide [9, 10] se centra en el modelado y simulación del comportamiento dinámico descrito mediante una arquitectura. Excepcionalmente, los componentes con los que estamos tratando en este ADL son **interfaces**, siendo la implementación independiente. Los principales puntos de interfaz se dividen en dos grupos principales: comunicaciones síncronas (*provides* y *requires*, es decir, lo que el interfaz proporciona o requiere), denotación de eventos asíncronos (*in action* y *out action*, es decir, acciones de entrada o salida) y servicios. El sistema de tipos es extensible, conteniendo un sublenguaje que soporta la parametrización. La semántica se especifica mediante *posets* (conjuntos de eventos parcialmente ordenados). Las restricciones se describen vía los interfaces y la semántica con restricciones algebraicas sobre el estado de los componentes y restricciones de patrones sobre los posets de eventos. La evolución se permite mediante la herencia en un subtipado estructural. No se tienen en cuenta las propiedades no funcionales.

Los conectores se modelan en línea, siendo los conectores complejos reutilizables sólo vía unos interfaces especiales denominados “componentes de conexión”. La semántica se especifica mediante posets y conexiones condicionales. No tiene un sistema de tipado, ni se pueden especificar restricciones. Tampoco se tiene en cuenta la evolución ni las propiedades no funcionales.

La configuración se modela en línea mediante conectores, siendo la composición realizada mediante mapas que relacionan una arquitectura a un interfaz. La comprensión se realiza mediante mediante especificaciones textuales en línea con muchos detalles de los conectores, y también se proporciona una notación gráfica. Los mapas de refinamiento permiten simulaciones comparativas de arquitecturas a diferentes niveles. Respecto a la heterogeneidad se soporta el desarrollo de simulaciones ejecutables descritas en un sublenguaje ejecutable propio. La escalabilidad se ve obstaculizada por las configuraciones en línea: a pesar de ello se utiliza para proyectos de gran escala. Tampoco la evolución se ve favorecida por el mismo motivo, ni tampoco se soportan las arquitecturas parciales. El dinamismo se representa de forma restringida mediante configuraciones condicionales y generación de eventos dinámicos. Las restricciones se describen mediante mapas de refinamiento y existe un lenguaje de restricciones de posets temporales que ayuda en la definición de las propiedades no funcionales relacionadas con el eje temporal.

El soporte de herramientas de Rapide admite múltiples vistas tanto textuales como gráficas, donde la visualización del comportamiento de ejecución se describe mediante la animación de simulaciones. Respecto al análisis que podemos realizar encontramos diversos analizadores y compiladores, siendo el análisis vía filtrado de eventos y animación, y un chequeador de restricciones para asegurar la validez de los mapas. El refinamiento viene facilitado por un compilador para un sublenguaje de ejecución y herramientas para compilar y verificar mapas de patrones de eventos durante la simulación. La generación de implementaciones se realiza precisamente mediante la construcción de simuladores ejecutables escritos en dicho sublenguaje. Respecto al dinamismo, se tiene soporte de compilación y ejecución para el cambio dinámico restringido de arquitecturas (configuración condicional).

3.2.5 UniCon

El foco de atención de UniCon [12] se centra en la generación de código de pegado para la interconexión de los componentes existentes utilizando los protocolos de interacción habituales. Basado en componentes, este ADL sí que restringe la implementación posterior. Los puntos de interfaz se denominan *players* (actores). Tiene un conjunto predefinido de tipos. La semántica se especifica mediante trazas de eventos con listas de propiedades. Las restricciones se describen vía las interfaces y la semántica, atendiendo a las propiedades no funcionales necesarias para un análisis de planificación y permitiendo las restricciones sobre los actores que pueden ser provistas por los tipos de componentes. No se tiene en cuenta la propiedad de la evolución.

Los conectores se modelan de forma explícita con un conjunto predefinido de tipos. Los puntos de interfaz se denominan roles. La semántica está implícita en los tipos del conector, cuya información puede darse mediante listas de propiedades. Las restricciones se describen vía los interfaces, restringiendo el tipo de actores que pueden utilizarse con un papel dado. No se tiene en cuenta la propiedad de evolución. Sólo se tienen en cuenta aquellas propiedades no funcionales necesarias para un análisis de planificación.

La configuración se modela de manera explícita mediante conectores, siendo la composición realizada mediante componentes y conectores compuestos. La comprensión se realiza mediante especificaciones textuales y gráficas explícitas, pudiendo ser la descripción de configuraciones distribuida. Respecto al refinamiento y las trazas, se soporta la generación de sistemas y la restricción de implementaciones. Respecto a la heterogeneidad, sólo se soportan tipos de componentes y conectores predefinidos y envoltorios de componentes. La escalabilidad se ayuda mediante configuraciones explícitas y el número variable de papeles de conectores. Respecto a la evolución se soporta (en parte) las arquitecturas parciales, ayudándose de configuraciones explícitas. Las únicas restricciones consisten en que los actores sólo pueden adjuntarse a papeles y viceversa. No se tienen en cuenta las propiedades del dinamismo y las no-funcionales.

El soporte de herramientas de UniCon tiene una especificación activa, en el sentido de que su editor gráfico previene los errores durante la fase de diseño mediante la invocación de un chequeador de lenguaje. Admite múltiples vistas tanto textuales como gráficas, donde los tipos de componentes y conectores se distinguen mediante íconos. Respecto al análisis que podemos realizar encontramos diversos analizadores y compiladores, siendo el análisis de planificación el más importante de ellos. El refinamiento viene facilitado por un compilador cuyos componentes primitivos se implementan en lenguajes de programación tradicionales. La generación de implementaciones se realiza en código C.

3.2.6 Wright

El foco de atención de Wright [1, 2] se centra en el modelado y análisis (detección de abrazos mortales de manera específica) del comportamiento dinámico de sistemas concurrentes. Basado en componentes e independiente de la implementación, los puntos de interfaz se denominan *ports* (puertos) cuyas semánticas de interacción se especifican en CSP [7]. El sistema de tipos es extensible, siendo parametrizable el número de puertos y la computación. Aunque no sea el objetivo principal, la semántica se puede especificar con CSP. Las restricciones se describen vía protocolos de interacción para cada puerto en CSP y con invariantes. La evolución se permite vía diferentes parámetros de instanciación. No se tienen en cuenta las propiedades no funcionales.

Los conectores se modelan de forma explícita con un sistema de tipos extensible basado en protocolos y con un número parametrizable de roles y *glue* (puntos de pegado). Los puntos de interfaz se denominan roles, cuya semántica se especifica mediante CSP, al igual que la semántica de pegado de los conectores. Las restricciones se describen via los interfaces y la semántica, los protocolos de interacción para cada rol en CSP y con invariantes. La evolución se define vía las diferentes instancias de parámetros. No se tienen en cuenta las propiedades no funcionales.

La configuración se modela de manera explícita mediante *attachments* (adjuntos), siendo la composición realizada mediante computaciones y *glue* (puntos de pegado) que son expresados como arquitecturas. La comprensión se realiza mediante especificaciones textuales concisas y explícitas. Respecto a la heterogeneidad, se soportan tanto elementos de grano fino como grueso. La escalabilidad se ayuda por configuraciones explícitas y número variable de roles, se utiliza en proyectos de gran escala. Respecto a la evolución, se ajusta a las especificaciones parciales ayudada por configuraciones explícitas. El dinamismo se ve restringido a la inserción, eliminación y re-conexión de elementos. Respecto a las restricciones, los puertos sólo pueden adjuntarse a papeles y viceversa, y con invariantes programables. No se tiene en cuenta las propiedades de refinamiento ni traza, ni las no-funcionales.

El soporte de herramientas de Wright admite múltiples vistas pero sólo en un formato textual, donde el chequeador de modelos proporciona una equivalencia textual de los símbolos CSP. Respecto al análisis que podemos realizar encontramos diversos analizadores y compiladores, siendo los más importantes el chequeador de conformidad de tipos de los puertos respecto a los roles y el análisis de bloqueos de conectores individuales.

Capítulo 4

Criterios para la Selección de un ADL

4.1 El Conjunto Mínimo de Requisitos Exigibles a un ADL

En general, el conjunto mínimo de requisitos que podemos exigir a la hora de considerar un lenguaje como un ADL son:

Comunicación Un ADL debe ser adecuado para comunicar una arquitectura a todas las partes interesadas en la misma. Todas las estructuras de una arquitectura deben poder definirse utilizando el ADL, incluyendo tanto aquéllas que son estáticas como las que son dinámicas. Los diversos tipos de componentes y conectores deben estar identificados en cada una de las estructuras. El nivel de granularidad de la información se debe poder personalizar según el lector actual de la arquitectura.

Análisis y Validación Un ADL debe dar soporte a las tareas de creación de la arquitectura, refinamiento y validación. Por tanto, debe incluir reglas sobre lo que se entiende por una arquitectura completa y consistente.

Propósito General Un ADL debe proporcionar la capacidad para representar (aunque sea indirectamente) la mayoría de los estilos arquitectónicos habituales.

Abstracción Un ADL debe tener la capacidad de proporcionar estructuras del sistema que expresen información arquitectónica y que, al mismo tiempo, supriman toda aquella información sobre la implementación que no sea arquitectónica.

Derivación El ADL debe proporcionar una base para fomentar la implementación (por ejemplo, la generación rápida de prototipos). Sobre todo, debe posibilitar la agregación de información extra a la especificación ADL de manera que permita que una especificación final del sistema sea derivada del ADL.

Alternativas de Implementación Si el lenguaje puede expresar información de nivel de implementación, debe tener posibilidades para optar a más de una implementación de las estructuras de nivel arquitectónico del sistema. Esto es, debe soportar la especificación de familias de implementaciones que satisfagan todas una arquitectura común.

4.2 ¿Cómo ayudan los ADL's al desarrollo de sistemas?

En cualquier modo, un ADL con todas sus características deberá formar parte de un entorno de desarrollo integrado que, además, soporte la agregación incremental de información. Un escenario general de utilización de un ADL durante la construcción de un sistema podría ser el que sigue:

- Sistema inicialmente descrito textual o gráficamente, basado en estilos arquitectónicos y tipos de componentes. Subsistemas principales descritos en términos de la información que aceptan y producen. Una descripción del comportamiento asociada a cada componente: podría incluir clases de eventos que producen y a los que responden, información temporal e información sobre la utilización de recursos.

- Descripción de alto nivel del sistema, sea mediante casos de uso, escenarios u otra alternativa más formal.

- Realización de diversos tipos de análisis y/o simulaciones del modelo arquitectónico que nos permitan estimar los recursos necesarios y obtener el grado de satisfacción de propiedades de calidad tales como el rendimiento, la disponibilidad o la seguridad. La información obtenida nos podrá llevar a realizar cambios, por lo que sería de gran utilidad realizarlos sin necesidad de reescribir el modelo.

- Refinamiento de componentes (cuando sea necesario) para cada tipo de análisis. Visualización de información de cada componente y de los resultados del análisis sobre ellos.

- Codificación o generación de plantillas a partir de las descripciones de los componentes.

4.3 Clasificación de Atributos de Selección

Podemos evaluar el ADL candidato aplicando un conjunto de criterios, mediante los cuales podemos compararlo con otros ADL's en términos de su capacidad inherente, el soporte de herramientas y otros muchos que vamos a describir en este capítulo:

4.3.1 Atributos orientados al sistema

- Adecuación del ADL para representar un dominio particular de sistemas.

- Aptitud del ADL para la descripción de estilos arquitectónicos.

- Capacidad del ADL para la descripción de diversos tipos de sistemas (tiempo-real, distribuidos, empotrados y otros)

4.3.2 Atributos orientados al lenguaje

- Formalización de la sintáxis y semántica del ADL.

- Concepto de completitud de una descripción arquitectónica descrita por el ADL, y si permite la posibilidad de tener descripciones arquitectónicas que no sean completas.

- Definición de la consistencia de una descripción arquitectónica frente a otros niveles abstractos del sistema software. Por ejemplo: el seguimiento de requisitos de calidad.
- Los tipos de información no arquitectónica que puede representar el ADL.
- Los tipos de estructuras que son soportadas por el ADL respecto a las diferentes perspectivas del sistema. La definición de mapeos entre dichas estructuras.
- El control de presentación del ADL.
- La capacidad del ADL para definir nuevos tipos de componentes, conectores y otros elementos.
- El soporte del ADL para definir extensiones específicas de un dominio.
- La capacidad del ADL para definir nuevos elementos o abstracciones.
- El soporte del ADL para la instanciación múltiple, es decir, la capacidad para crear copias de un componente idéntico al original pero que varían de una forma específica.
- La capacidad del ADL para crear subconjuntos, es decir, la aptitud para partir la descripción arquitectónica en piezas más pequeñas que pueden ser examinadas o analizadas a parte.
- La capacidad de incluir comentarios del ADL.
- La capacidad de facilitar la modificación de las descripciones de arquitecturas software del ADL.
- La escalabilidad del ADL para la representación de grandes sistemas complejos.
- El soporte del ADL para definir diversos niveles de abstracción arbitrarios, es decir, niveles jerárquicos de detalle.
- El soporte del ADL para definir referencias cruzadas arbitrarias, es decir, la capacidad para hacer referencia a la información que esté relacionada dentro de la descripción arquitectónica.
- La capacidad del ADL para representar las variaciones en los diversos sistemas que pueden derivarse de la arquitectura.

4.3.3 Atributos orientados al proceso

- Existencia de un editor de textos o una herramienta específica al ADL que permita manipular directamente las descripciones textuales de arquitecturas.
- Existencia de un editor de gráficos o una herramienta específica al ADL que permita manipular directamente las descripciones gráficas de arquitecturas.
- La capacidad de una herramienta específica al ADL que permita importar información de otras descripciones en la arquitectura.

- La capacidad de una herramienta específica al ADL que permita chequeos sintácticos, chequeos semánticos, chequeos de completitud, chequeos de la consistencia interna y/o chequeos de la consistencia externa.
- Existencia de un navegador que permita un soporte interactivo para la navegación adireccional de la descripción.
- Existencia de un buscador que permita un soporte interactivo para la navegación direccional de la descripción.
- La capacidad del ADL para soportar un refinamiento incremental.
- La capacidad del ADL para soportar directamente el control de versiones.
- La capacidad del ADL para comparar dos representaciones arquitectónicas para comprobar si representan a la misma arquitectura.
- El soporte proporcionado por el ADL para analizar información a nivel arquitectónico con el objetivo de predecir o proyectar el grado de satisfacción de requisitos de calidades del sistema final, tales como la planificabilidad, el rendimiento, el uso de la memoria, la completitud, la correctitud, la seguridad, la interoperabilidad, la facilidad de mantenimiento, la fiabilidad, la facilidad de uso y otras muchas.
- La capacidad del ADL para soportar la construcción de un sistema software compilable o ejecutable desde el diseño. En concreto, que permita que la descripción de un sistema sea compuesta de (o integrada con) cuerpos de componentes para producir un sistema software compilable y ejecutable (particularmente para tipos de plataformas diversas y entornos tales como distribuidos heterogeneos).
- La capacidad del ADL para generar casos de prueba.
- La capacidad del ADL para generar la documentación.
- La documentación para utilizar el ADL.

Bibliografía

[1] R. Allen. *A Formal Approach to Software Architecture*. Phd. thesis, Carnegie Mellon University, Octubre 1997.

[2] R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Trans. Software Engineering and Methodology*, 6(3):213–249, July 1997.

[3] P. Binns, M. Engelhart, M. Jackson, and S. Vestal. Domain-Specific Software Architectures for Guidance, Navigation and Control. *Int'l J. Software Engineering and Knowledge Engineering*, 6(2), 1996.

[4] D. Garlan. An Introduction to the Aesop System. Technical report, Carnegie Mellon University, 1995. http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/html/aesop-overview.ps.

[5] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proc. SIGSOFT'94: Foundations of Software Engineering*, pages 175–188, December 1994.

[6] D. Garlan, D. Monroe, and D. Wile. ACME: An architectural interchange language. In *Proc. of the* XIX^{th} *Intl. Conference on Software Engineering. ICSE' 97*, Boston, 1997.

[7] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[8] P. Clements L. Bass and R. Kazman. *Software Architecture in Practice*. Addison–Wesley, 1998.

[9] D. Luckham, J. Kenney, L. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and Analysis of System Architecture using Rapide. *IEEE Transactions on Software Engineering*, 21(4):336–355, April 1995.

[10] D. Luckham and J. Vera. An Event-Based Architecture Definition Language. *IEEE Transactions on Software Engineering*, 21(9):717–734, September 1995.

[11] N. Medvidovic and R.N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, January 2000.

[12] M. Shaw, R. DeLine, D. Klein, T. Ross, D. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, April 1995.

[13] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[14] S. Vestal. A Cursory Overview and Comparison of Four Architecture Description Languages. Technical report, Honeywell Technology Center, February 1993.

[15] S. Vestal. MetaH Programmer's Manual, Version 1.09. Technical report, Honeywell Technology Center, April 1996.