Introducción a la Arquitectura de Software

Contenidos:

Introducción a la Arquitectura de Software

Versión 1.0 – Marzo de 2004

Carlos Billy Reynoso
UNIVERSIDAD DE BUENOS AIRES

Introducción

Este documento constituye una introducción sumaria a la Arquitectura de Software, con el propósito puntual de brindar una visión de conjunto lo más estructurada posible para luego establecer el papel de esta disciplina emergente en relación con la estrategia arquitectónica de Microsoft, sus herramientas y sus patrones de diseño. Hay múltiples razones para desarrollar esta presentación. Por empezar, no hay todavía textos en lengua castellana que brinden aproximaciones actualizadas a la Arquitectura de Software (en adelante, AS). El proceso editorial es hoy mucho más lento que el flujo de los acontecimientos y el cambio tecnológico; casi toda la producción en papel manifiesta atraso respecto de los elementos de juicio que urge considerar, tanto en el plano conceptual como en el tecnológico. Pero aún operando en binario y en banda ancha sobre la red de redes, el flujo de información de la industria rara vez se cruza con los intercambios del circuito académico, lo que ocasiona que la empresa y la academia terminen definiendo prioridades distintas, diagnosticando la situación de maneras discrepantes, otorgando diferentes valores a los criterios y usando las mismas nomenclaturas sin compartir sus significados. Como lo ha dicho Jan Bosch, un arquitecto práctico: “Existe una considerable diferencia entre la percepción académica de la AS y la práctica industrial. … Es interesante advertir que a veces los problemas que la industria identifica como los más importantes y difíciles, no se identifican o se consideran no-problemas en la academia” [Bos00].

Por otra parte, en la estrategia de Microsoft se ha desarrollado un marco de referencia global y genérico para el desarrollo de soluciones, Microsoft Solutions Framework, hoy en día en su tercera encarnación mayor. En MSF apenas hay mención de la AS, y en la perspectiva de otros documentos que podrían tenerla más en foco (como [Platt02]) no se la trata en términos semejantes a los que son comunes en la academia, que es, después de todo, donde se originan las ideas que la constituyen. Vinculado de alguna manera (implícita) con los lineamientos de MSF y bajo el paraguas (explícito) de “arquitectura”, se encuentra además un buen número de aportes en materia de patrones de diseño y lineamientos para implementarlos en el Framework .NET, primordialmente en modelos orientados a servicios. En ese contexto, delimitado por un marco necesariamente general (más afín a IT Management que a arquitectura o ingeniería) y por una práctica sumamente concreta, las cuestiones teóricas y las metodologías específicas basadas en arquitectura han quedado sin elaborar.

Existe entonces espacio y oportunidad para comenzar a articular esas referencias pendientes, de una manera que contribuya a situar esa estrategia particular (MSF+Patrones) en el marco de las tendencias actuales de teoría y práctica arquitectónica. Sin que estos documentos expresen una visión oficial, nos parece útil llenar el vacío, tender un puente, entre la investigación básica y los aportes académicos

por un lado y las visiones y requerimientos de industria por el otro. Lo que aquí ha de hacerse es otorgar contenidos, aunque sean provisionales y contestables, al concepto de "Arquitectura de Software", dado que cada vez que aparece en la documentación de industria se da por sentado lo que esa expresión significa.

El presente estudio constituye entonces el capítulo introductorio a una visión de conjunto de la AS, articulada conforme a este temario:

1. Arquitectura de Software
    - 1.1. Antecedentes históricos
    - 1.2. Definiciones y delimitación de la disciplina
    - 1.3. Conceptos fundamentales
    - 1.4. Campos de la Arquitectura de Software
    - 1.5. Modalidades y tendencias
    - 1.6. Diferencias entre Arquitectura y Diseño
    - 1.7. Repositorios
    - 1.8. Problemas pendientes en Arquitectura de Software
    - 1.9. Relevancia de la Arquitectura de Software
    - 1.10. (Referencias bibliográficas)
2. Estilos de Arquitectura
    - 2.1. Definiciones de estilo
    - 2.2. Clasificaciones de estilos arquitectónicos
    - 2.3. Inventario y Descripción de estilos arquitectónicos
    - 2.4. Estilos y patrones de arquitectura y diseño
    - 2.5. El lugar de los estilos en los marcos de referencia y en las vistas arquitectónicas
    - 2.6. Los estilos como valor contable
    - 2.7. (Referencias bibliográficas)
3. Lenguajes de descripción arquitectónica (ADLs)
    - 3.1. Introducción a los ADLs
    - 3.2. Criterios de definición de un ADL
    - 3.3. Lenguajes: Acme / Armani ADLs: Acme /Armani – ADML – Aesop – ArTek – C2 SADL – CHAM – Darwin – Jacal – LILEANNA – MetaH / AADL – Rapide – UML – UniCon – Wright – xArch / xADL
    - 3.4. Modelos computacionales y paradigmas de modelado
    - 3.5. ADLs y Patrones
    - 3.6. (Referencias bibliográficas)
4. Modelos de proceso y diseño
    - 4.1. Métodos tradicionales y de peso completo – CMM, UPM
    - 4.2. Métodos basados en arquitectura
        - 4.2.1. Métodos de análisis y diseño en el ciclo de vida – Visión general
        - 4.2.2. Arquitectura basada en escenarios (FAAM, ALMA)
        - 4.2.3. El diseño arquitectónico en el ciclo de vida: ABD
        - 4.2.4. Active Review for Intermediate Design (ARID)
        - 4.2.5. Quality Attribute Workshops (QAW) - QASAR
        - 4.2.6. Attribute-Driven Design (ADD)
        - 4.2.7. Evaluación: Architecture Tradeoff Analysis Method (ATAM)
        - 4.2.8. Métodos de evaluación de opciones arquitectónicas (SACAM)
        - 4.2.9. Derivación de tácticas arquitectónicas
        - 4.2.10. Economía de la arquitectura: Cost-Benefits Analysis Method (CBAM)
        - 4.2.11. Documentación de la Arquitectura
    - 4.3. Métodos heterodoxos y de peso ligero: Métodos Agiles, Programación Extrema – Concepción caórdica de los sistemas complejos
    - 4.4. Relación de los métodos LW con MSF 3 y con las metodologías basadas en arquitectura.
5. Herramientas arquitectónicas
    - 5.1. El lugar de UML 1.x y 2.0 en arquitectura de software – Alcances y limitaciones
    - 5.2. Herramientas de diseño y análisis asociadas a ADLs

5.3. Herramientas de Análisis, Evaluación y Visualización (SAAMTool, AET y análogas)
5.4. Herramientas de recuperación y visualización de arquitectura
5.5. Herramientas auxiliares de integración (MBI)
5.6. Servicios, plantillas y herramientas arquitectónicas en VS .NET Architect
6. Prácticas de diseño sobre arquitecturas orientadas a servicios y modelos de integración

El presente documento cubre los puntos 1.1 a 1.9 del plan de tratamiento del tema. Ya se encuentran disponibles documentos que cubren los temas 2 (estilos de arquitectura) y 3 (lenguajes de descripción arquitectónica). Los demás elementos del itinerario están en proceso de elaboración y se irán publicando a medida que estén listos.

El propósito de la totalidad del estudio consiste en establecer las bases para una discusión teórica y los fundamentos para una puesta en práctica de un modelo de diseño y desarrollo basado en arquitectura, ya que a pesar de la buena imagen de la disciplina, sus aportes sustantivos permanecen desconocidos para una mayoría de los ingenieros y metodólogos de software. Dado que estos documentos se presentan como punto de partida para la discusión de estos temas para la comunidad de arquitectos, después de la presentación de cada tópico se formula invitación para desarrollar los mismos contenidos desde otras perspectivas, aportar elementos de juicio adicionales, suministrar referencias de casos, corregir lagunas, agregar vínculos, difundir investigaciones relacionadas, compensar los sesgos, refinar el debate o enriquecer los materiales que aquí comienzan.

Breve Historia de la Arquitectura de Software

Todavía no se ha escrito una historia aceptable de la AS. Desde que Mary Shaw o David Garlan reseñaran escuetamente la prehistoria de la especialidad a principios de los 90, los mismos párrafos han sido reutilizados una y otra vez en la literatura, sin mayor exploración de las fuentes referidas en la reseña primaria y con prisa por ir al grano, que usualmente no es de carácter histórico. En este estudio se ha optado, más bien, por inspeccionar las fuentes más de cerca, con el objeto de señalar las supervivencias y las re-semantizaciones que han experimentado las ideas fundadoras en la AS contemporánea, definir con mayor claridad el contexto, entender que muchas contribuciones que pasaron por complementarias han sido en realidad antagónicas y comprender mejor por qué algunas ideas que surgieron hace cuatro décadas demoraron un cuarto de siglo en materializarse.

Esta decisión involucra algo más que el perfeccionamiento de la lectura que pueda hacerse de un conjunto de acontecimientos curiosos. Las formas divergentes en que se han interpretado dichas ideas, después de todo, permiten distinguir corrientes de pensamiento diversas, cuyas diferencias distan de ser triviales a la hora de plasmar las ideas en una metodología. Todo lo que parece ser un simple elemento de juicio, no pocas veces implica una disyuntiva. Situar las inflexiones de la breve historia de la AS en un contexto temporal, asimismo, ayudará a comprender mejor cuáles son sus contribuciones perdurables y cuáles sus manifestaciones contingentes al espíritu de los tiempos y a las modas tecnológicas que se han ido sucediendo.

Si bien la AS acostumbra remontar sus antecedentes al menos hasta la década de 1960, su historia no ha sido tan continua como la del campo más amplio en el que se inscribe, la ingeniería de software [Pfl02] [Pre01]. Después de las tempranas inspiraciones del legendario Edsger Dijkstra, de David Parnas y de Fred Brooks, la AS quedó en estado de

vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado [PW92]. Puede decirse que Perry y Wolf fundaron la disciplina, y su llamamiento fue respondido en primera instancia por los miembros de lo que podría llamarse la escuela estructuralista de Carnegie Mellon: David Garlan, Mary Shaw, Paul Clements, Robert Allen.

Se trata entonces de una práctica joven, de apenas unos doce años de trabajo constante, que en estos momentos experimenta una nueva ola creativa en el desarrollo cabal de sus técnicas en la obra de Rick Kazman, Mark Klein, Len Bass y otros metodólogos en el contexto del SEI, en la misma universidad. A comienzos del siglo XXI comienzan ya a discernirse tendencias, cuyas desavenencias mutuas todavía son leves: al menos una en el sur de California (Irvine y Los Angeles) con Nenad Medvidovic, David Rosenblum y Richard Taylor, otra en el SRI de Menlo Park con Mark Moriconi y sus colegas y otra más vinculada a las recomendaciones formales de la IEEE y los trabajos de Rich Hilliard. Hoy se percibe también un conjunto de posturas europeas que enfatizan mayormente cuestiones metodológicas vinculadas con escenarios y procuran inscribir la arquitectura de software en el ciclo de vida, comenzando por la elicitación de los requerimientos. Antes de Perry y Wolf, empero, se formularon ideas que serían fundamentales para la disciplina ulterior. Comencemos entonces por el principio, aunque siempre cabrá la posibilidad de discutir cuál puede haber sido el momento preciso en el que todo comenzó.

Cada vez que se narra la historia de la arquitectura de software (o de la ingeniería de software, según el caso), se reconoce que en un principio, hacia 1968, Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera [Dij68a]. Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes y que se superponen “como capas de cebolla”. Inventó o ayudó a precisar además docenas de conceptos: el algoritmo del camino más corto, los *stacks*, los vectores, los semáforos, los abrazos mortales. De sus ensayos arranca la tradición de hacer referencia a “niveles de abstracción” que ha sido tan común en la arquitectura subsiguiente. Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego expresarían Niklaus Wirth [Wir71] como *stepwise refinement* y DeRemer y Kron [DK76] como *programming-in-the large* (o programación en grande), ideas que poco a poco irían decantando entre los ingenieros primero y los arquitectos después.

En la conferencia de la NATO de 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló estas sorprendentes apreciaciones comentando las ideas de Dijkstra:

> Pienso que tenemos algo, aparte de la ingeniería de software: algo de lo que hemos hablado muy poco pero que deberíamos poner sobre el tapete y concentrar la atención en ello. Es la cuestión de la arquitectura de software. La arquitectura es diferente de la ingeniería. Como ejemplo de lo que quiero decir, echemos una mirada a OS/360. Partes de OS/360 están extremadamente bien codificadas.

> Partes de OS, si vamos al detalle, han utilizado técnicas que hemos acordado constituyen buena práctica de programación. La razón de que OS sea un amontonamiento amorfo de programas es que no tuvo arquitecto. Su diseño fue delegado a series de grupos de ingenieros, cada uno de los cuales inventó su propia arquitectura. Y cuando esos pedazos se clavaron todos juntos no produjeron una tersa y bella pieza de software [NATO76: 150].

Sharp continúa su alegación afirmando que con el tiempo probablemente llegue a hablarse de "la escuela de arquitectura de software de Dijkstra" y se lamenta que en la industria de su tiempo se preste tan poca o ninguna atención a la arquitectura. La frase siguiente también es extremadamente visionaria:

> Lo que sucede es que las especificaciones de software se consideran especificaciones funcionales. Sólo hablamos sobre lo que queremos que haga el programa. Es mi creencia que cualquiera que sea responsable de la implementación de una pieza de software debe especificar más que esto. Debe especificar el diseño, la forma; y dentro de ese marco de referencia, los programadores e ingenieros deben crear algo. Ningún ingeniero o programador, ninguna herramienta de programación, nos ayudará, o ayudará al negocio del software, a maquillar un diseño feo. El control, la administración, la educación y todas las cosas buenas de las que hablamos son importantes; pero la gente que implementa debe entender lo que el arquitecto tiene en mente [Idem].

Nadie volvió a hablar del asunto en esa conferencia, sin embargo. Por unos años, "arquitectura" fue una metáfora de la que se echó mano cada tanto, pero sin precisión semántica ni consistencia pragmática. En 1969 Fred Brooks Jr y Ken Iverson llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador. En 1971, C. R. Spooner tituló uno de sus ensayos "Una arquitectura de software para los 70s" [Spo71], sin que la mayor parte de la historiografía de la AS registrara ese antecedente.

En 1975, Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar "la especificación completa y detallada de la interfaz de usuario" y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa [Bro75], empleando una nomenclatura que ya nadie aplica de ese modo. En el mismo texto, identificaba y razonaba sobre las estructuras de alto nivel y reconocía la importancia de las decisiones tomadas a ese nivel de diseño. También distinguía entre arquitectura e implementación; mientras aquella decía *qué* hacer, la implementación se ocupa de *cómo*. Aunque el concepto de AS actual y el de Brooks difieren en no escasa medida, el texto de Brooks *The mythical man-month* sigue siendo, un cuarto de siglo más tarde, el más leído en ingeniería de software. Se ha señalado que Dijkstra y Brooks, el primero partidario de un formalismo matemático y el segundo de considerar las variables humanas, constituyen dos personalidades opuestas, que se sitúan en los orígenes de las metodologías fuertes y de las heterodoxias ágiles, respectivamente [Tra02]; pero eso será otra historia.

Una novedad importante en la década de 1970 fue el advenimiento del diseño estructurado y de los primeros modelos explícitos de desarrollo de software. Estos modelos comenzaron a basarse en una estrategia más orgánica, evolutiva, cíclica, dejando atrás las metáforas del desarrollo en cascada que se inspiraban más bien en la línea de

montaje de la ingeniería del hardware y la manufactura. Surgieron entonces las primeras investigaciones académicas en materia de diseño de sistemas complejos o "intensivos". Poco a poco el diseño se fue independizando de la implementación, y se forjaron herramientas, técnicas y lenguajes de modelado específicos.

En la misma época, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada. Parnas desarrolló temas tales como módulos con ocultamiento de información [Par72], estructuras de software [Par74] y familias de programas [Par76], enfatizando siempre la búsqueda de calidad del software, medible en términos de economías en los procesos de desarrollo y mantenimiento. Aunque Dijkstra, con sus frases lapidarias y memorables, se ha convertido en la figura legendaria por excelencia de los mitos de origen de la AS, Parnas ha sido sin duda el introductor de algunas de sus nociones más esenciales y permanentes.

En 1972, Parnas publicó un ensayo en el que discutía la forma en que la modularidad en el diseño de sistemas podía mejorar la flexibilidad y el control conceptual del sistema, acortando los tiempos de desarrollo [Par72]. Introdujo entonces el concepto de ocultamiento de información (*information hiding*), uno de los principios de diseño fundamentales en diseño de software aún en la actualidad. La herencia de este concepto en la ingeniería y la arquitectura ulterior es inmensa, y se confunde estrechamente con la idea de abstracción. En la segunda de las descomposiciones que propone Parnas comienza a utilizarse el ocultamiento de información como criterio. "Los módulos ya no se corresponden con las etapas de procesamiento. … Cada módulo en la segunda descomposición se caracteriza por su conocimiento de una decisión de diseño oculta para todos los otros módulos. Su interfaz o definición se escoge para que revele tan poco como sea posible sobre su forma interna de trabajo" [Par72]. Cada módulo deviene entonces una caja negra para los demás módulos del sistema, los cuales podrán acceder a aquél a través de interfaces bien definidas, en gran medida invariables. Es fácil reconocer en este principio ideas ya presentadas por Dijkstra en su implementación del THE-Multiprogramming System [Dij68a]. Pero la significación del artículo de 1972 radica en la idea de Parnas de basar la técnica de modularización en decisiones de diseño, mientras que los "niveles de abstracción" de Dijkstra involucraban más bien (igual que su famosa invectiva en contra del Go-to) técnicas de programación.

El concepto de ocultamiento se fue mezclando con encapsulamiento y abstracción, tras algunos avatares de avance y retroceso. Los arquitectos más escrupulosos distinguen entre encapsulamiento y ocultamiento, considerando a aquél como una capacidad de los lenguajes de programación y a éste como un principio más general de diseño. De hecho, Parnas no hablaba en términos de programación orientada a objeto, sino de módulos y sub-rutinas, porque el momento de los objetos no había llegado todavía.

El pensamiento de Parnas sobre familias de programas, en particular, anticipa ideas que luego habrían de desarrollarse a propósito de los estilos de arquitectura:

> Una familia de programas es un conjunto de programas (no todos los cuales han sido construidos o lo serán alguna vez) a los cuales es provechoso o útil considerar como grupo. Esto evita el uso de conceptos ambiguos tales como "similitud funcional" que surgen a veces cuando se describen dominios. Por

> ejemplo, los ambientes de ingeniería de software y los juegos de video no se consideran usualmente en el mismo dominio, aunque podrían considerarse miembros de la misma familia de programas en una discusión sobre herramientas que ayuden a construir interfaces gráficas, que casualmente ambos utilizan.

Una familia de programas puede enumerarse en principio mediante la especificación del árbol de decisión que se atraviesa para llegar a cada miembro de la familia. Las hojas del árbol representan sistemas ejecutables. El concepto soporta la noción de derivar un miembro de la familia a partir de uno ya existente. El procedimiento consiste en seguir hacia atrás el árbol hasta que se alcanza un nodo (punto de decisión) genealógicamente común a ambos, y luego proceder hacia abajo hasta llegar al miembro deseado. El concepto también soporta la noción de derivar varios miembros de la familia de un punto de decisión común, aclarando la semejanza y las diferencias entre ellos.

Las decisiones iniciales son las que más probablemente permanecerán constantes entre los miembros; las decisiones más tardías (cerca de las hojas) en un árbol prudentemente estructurado deberían representar decisiones susceptibles de cambiarse trivialmente, tales como los valores de tiempo de compilación o las constantes de tiempo de carga. La significación del concepto de familia de programas para la AS es que ella corresponde a las decisiones cerca del tope del árbol de decisión de Parnas. Es importante considerar que el árbol de Parnas es *top-down* no sólo porque se construye y recorre de lo general a lo particular, sino porque sus raíces se encuentran hacia arriba (o a la izquierda si el modelo es horizontal). No menos esencial es tener en cuenta que el árbol se ofreció como alternativa a métodos de descomposición basados en diagramas de flujo, por los que la AS no ha mostrado nunca demasiada propensión.

Decía Parnas que las decisiones tempranas de desarrollo serían las que probablemente permanecerían invariantes en el desarrollo ulterior de una solución. Esas “decisiones tempranas” constituyen de hecho lo que hoy se llamarían decisiones arquitectónicas. Como escriben Clements y Northrop [CN96] en todo el desenvolvimiento ulterior de la disciplina permanecería en primer plano esta misma idea: la estructura es primordial (*structure matters*), y la elección de la estructura correcta ha de ser crítica para el éxito del desarrollo de una solución. Ni duda cabe que “la elección de la estructura correcta” sintetiza, como ninguna otra expresión, el programa y la razón de ser de la AS.

En la década de 1980, los métodos de desarrollo estructurado demostraron no escalar suficientemente y fueron dejando el lugar a un nuevo paradigma, el de la programación orientada a objetos. En teoría, parecía posible modelar el dominio del problema y el de la solución en un lenguaje de implementación. La investigación que llevó a lo que después sería el diseño orientado a objetos puede remontarse incluso a la década de 1960 con Simula, un lenguaje de programación de simulaciones, el primero que proporcionaba tipos de datos abstractos y clases, y después fue refinada con el advenimiento de Smalltalk. Paralelamente, hacia fines de la década de 1980 y comienzos de la siguiente, la expresión *arquitectura de software* comienza a aparecer en la literatura para hacer referencia a la configuración morfológica de una aplicación.

Mientras se considera que la ingeniería de software se fundó en 1968, cuando F.L. Bauer usó ese sintagma por primera vez en la conferencia de la OTAN de Garmisch, Alemania, la AS, como disciplina bien delimitada, es mucho más nueva de lo que generalmente se sospecha. El primer texto que vuelve a reivindicar las abstracciones de alto nivel,

reclamando un espacio para esa reflexión y augurando que el uso de esas abstracciones en el proceso de desarrollo pueden resultar en "un nivel de arquitectura de software en el diseño" es uno de Mary Shaw [Shaw84] seguido por otro lllamado "Larger scale systems require higher level abstractions" [Shaw89]. Se hablaba entonces de un nivel de abstracción en el conjunto; todavía no estaban en su lugar los elementos de juicio que permitieran reclamar la necesidad de una disciplina y una profesión particulares.

El primer estudio en que aparece la expresión "arquitectura de software" en el sentido en que hoy lo conocemos es sin duda el de Perry y Wolf [PW92]; ocurrió tan tarde como en 1992, aunque el trabajo se fue gestando desde 1989. En él, los autores proponen concebir la AS por analogía con la arquitectura de edificios, una analogía de la que luego algunos abusaron [WWI99], otros encontraron útil y para unos pocos ha devenido inaceptable [BR01]. El artículo comienza diciendo exactamente:

> El propósito de este *paper* es construir el fundamento para la arquitectura de software. Primero desarrollaremos una intuición para la arquitectura de software recurriendo a diversas disciplinas arquitectónicas bien definidas. Sobre la base de esa intuición, presentamos un modelo para la arquitectura de software que consiste en tres componentes: elementos, forma y razón (*rationale*). Los elementos son elementos ya sea de procesamiento, datos o conexión. La forma se define en términos de las propiedades de, y las relaciones entre, los elementos, es decir, restricciones operadas sobre ellos. La razón proporciona una base subyacente para la arquitectura en términos de las restricciones del sistema, que lo más frecuente es que se deriven de los requerimientos del sistema. Discutimos los componentes del modelo en el contexto tanto de la arquitectura como de los estilos arquitectónicos ….

La declaración, como puede verse, no tiene una palabra de desperdicio: cada idea cuenta, cada intuición ha permanecido viva desde entonces. Los autores prosiguen reseñando el progreso de las técnicas de diseño en la década de 1970, en la que los investigadores pusieron en claro que el diseño es una actividad separada de la implementación y que requiere notaciones, técnicas y herramientas especiales. Los resultados de esta investigación comienzan ahora (decían en 1992) a penetrar el mercado en la forma de herramientas de ingeniería asistida por computadoras, CASE. Pero uno de los resultados del uso de estas herramientas ha sido que se produjo la absorción de las herramientas de diseño por los lenguajes de implementación. Esta integración ha tendido a esfumar, si es que no a confundir, la diferencia entre diseño e implementación. En la década de 1980 se perfeccionaron las técnicas descriptivas y las notaciones formales, permitiéndonos razonar mejor sobre los sistemas de software. Para la caracterización de lo que sucederá en la década siguiente ellos formulan esta otra frase que ha quedado inscripta en la historia mayor de la especialidad:

> La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término "arquitectura" en contraste con "diseño", para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de los arquitectos de software) y de estilo. … Es tiempo de re-examinar el papel de la arquitectura de software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas.

Considerada como disciplina por mérito propio, la AS ha de ser, para ellos, beneficiosa como marco de referencia para satisfacer requerimientos, una base esencial para la estimación de costos y administración del proceso y para el análisis de las dependencias y la consistencia del sistema.

Dando cumplimiento a las profecías de Perry y Wolf, la década de 1990 fue sin duda la de la consolidación y diseminación de la AS en una escala sin precedentes. Las contribuciones más importantes surgieron en torno del instituto de ingeniería de la información de la Universidad Carnegie Mellon (CMU SEI), antes que de cualesquiera organismos de industria. En la misma década, demasiado pródiga en acontecimientos, surge también la programación basada en componentes, que en su momento de mayor impacto impulsó a algunos arquitectos mayores, como Paul Clements [Cle96b], a afirmar que la AS promovía un modelo que debía ser más de integración de componentes pre-programados que de programación.

Un segundo gran tema de la época fue el surgimiento de los patrones, cristalizada en dos textos fundamentales, el de la Banda de los Cuatro en 1995 [Gof95] y la serie *POSA* desde 1996 [BMR+96]. El primero de ellos promueve una expansión de la programación orientada a objetos, mientras que el segundo desenvuelve un marco ligeramente más ligado a la AS. Este movimiento no ha hecho más que expandirse desde entonces. El originador de la idea de patrones fue Christopher Alexander, quien incidentalmente fue arquitecto *de edificios*; Alexander desarrolló en diversos estudios de la década de 1970 temas de análisis del sentido de los planos, las formas, la edificación y la construcción, en procura de un modelo constructivo y humano de arquitectura, elaborada de forma que tenga en cuenta las necesidades de los habitantes [Ale77]. El arquitecto (y puede copiarse aquí lo que decía Fred Brooks) debe ser un agente del usuario.

A lo largo de una cadena de intermediarios y pensadores originales, las ideas llegaron por fin a la informática diez años más tarde. Si bien la idea de arquitectura implícita en el trabajo actual con patrones está más cerca de la implementación y el código, y aunque la reutilización de patrones guarda estrecha relación con la tradición del diseño concreto orientado a objetos [Lar03], algunos arquitectos emanados de la escuela de Carnegie Mellon formalizaron un acercamiento con esa estrategia [Shaw96] [MKM+96] [MKM+97]. Tanto en los patrones como en la arquitectura, la idea dominante es la de re-utilización. A impulsos de otra idea mayor de la época (la crisis del software), la bibliografía sobre el impacto económico de la re-utilización alcanza hoy magnitudes de cuatro dígitos.

Como quiera que sea, la AS de este período realizó su trabajo de homogeneización de la terminología, desarrolló la tipificación de los estilos arquitectónicos y elaboró lenguajes de descripción de arquitectura (ADLs), temas que en este estudio se tratan en documentos separados. También se consolidó la concepción de las vistas arquitectónicas, reconocidas en todos y cada uno de los *frameworks* generalizadores que se han propuesto (4+1, TOGAF, RM/ODP, IEEE), como luego se verá.

Uno de los acontecimientos arquitectónicos más importantes del año 2000 fue la hoy célebre tesis de Roy Fielding que presentó el modelo REST, el cual establece definitivamente el tema de las tecnologías de Internet y los modelos orientados a servicios y recursos en el centro de las preocupaciones de la disciplina [Fie00]. En el mismo año se publica la versión definitiva de la recomendación IEEE Std 1471, que

procura homogeneizar y ordenar la nomenclatura de descripción arquitectónica y homologa los estilos como un modelo fundamental de representación conceptual.

En el siglo XXI, la AS aparece dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. Todo lo que se ha hecho en ingeniería debe formularse de nuevo, integrando la AS en el conjunto. La producción de estas nuevas metodologías ha sido masiva, y una vez más tiene como epicentro el trabajo del Software Engineering Institute en Carnegie Mellon. La aparición de las metodologías basadas en arquitectura, junto con la popularización de los métodos ágiles en general y Extreme Programming en particular, han causado un reordenamiento del campo de los métodos, hasta entonces dominados por las estrategias de diseño "de peso pesado". Después de la AS y de las tácticas radicales, las metodologías nunca volverán a ser las mismas.

La semblanza que se ha trazado no es más que una visión selectiva de las etapas recorridas por la AS. Los lineamientos de ese proceso podrían dibujarse de maneras distintas, ya sea enfatizando los hallazgos formales, las intuiciones dominantes de cada período o las diferencias que median entre la abstracción cualitativa de la arquitectura y las cuantificaciones que han sido la norma en ingeniería de software.

Definiciones

No es novedad que ninguna definición de la AS es respaldada unánimemente por la totalidad de los arquitectos. El número de definiciones circulantes alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas, como la colección que se encuentra en el SEI (http://www.sei.cmu.edu/architecture/definitions.html), a la que cada quien puede agregar la suya. En general, las definiciones entremezclan despreocupadamente (1) el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño (su lugar en el ciclo de vida), (2) la configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción y (3) la caracterización de la disciplina que se ocupa de uno de esos dos asuntos, o de ambos.

Una definición reconocida es la de Clements [Cle96a]: La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

En una definición semejante, hay que aclararlo, la idea de "componente" no es la de la correspondiente tecnología de desarrollo (COM, CORBA Component Model, EJB), sino la de elemento propio de un estilo. Un componente es una cosa, una entidad, a la que los arquitectos prefieren llamar "componente" antes que "objeto", por razones que se verán en otros documentos de esta serie pero que ya es obvio imaginar cuáles han de ser.

A despecho de la abundancia de definiciones del campo de la AS, existe en general acuerdo de que ella se refiere a la estructura a grandes rasgos del sistema, estructura

consistente en componentes y relaciones entre ellos [BCK98]. Estas cuestiones estructurales se vinculan con el diseño, pues la AS es después de todo una forma de diseño de software que se manifiesta tempranamente en el proceso de creación de un sistema; pero este diseño ocurre a un nivel más abstracto que el de los algoritmos y las estructuras de datos. En el que muchos consideran un ensayo seminal de la disciplina, Mary Shaw y David Garlan sugieren que dichas cuestiones estructurales incluyen organización a grandes rasgos y estructura global de control; protocolos para la comunicación, la sincronización y el acceso a datos; la asignación de funcionalidad a elementos del diseño; la distribución física; la composición de los elementos de diseño; escalabilidad y performance; y selección entre alternativas de diseño [GS94].

En una definición tal vez demasiado amplia, David Garlan [Gar00] establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño. La definición "oficial" de AS se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que reza así:

> La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

Aunque las literaturas de ambos campos rara vez convergen, entendemos que es productivo contrastar esa definición con la de *ingeniería* de software, conforme al estándar IEEE 610.12.1990:

> La aplicación de una estrategia sistemática, disciplinada y cuantificable al desarrollo, aplicación y mantenimiento del software; esto es, la aplicación de la ingeniería al software.

Obsérvese entonces que la noción clave de la arquitectura es la organización (un concepto cualitativo o estructural), mientras que la ingeniería tiene fundamentalmente que ver con una sistematicidad susceptible de cuantificarse.

Antes el número y variedad de definiciones existentes de AS, Mary Shaw y David Garlan [SG95] proporcionaron una sistematización iluminadora, explicando las diferencias entre definiciones en función de distintas clases de modelos. Destilando las definiciones y los puntos de vista implícitos o explícitos, los autores clasifican los modelos de esta forma:

1) **Modelos estructurales**: Sostienen que la AS está compuesta por componentes, conexiones entre ellos y (usualmente) otros aspectos tales como configuración, estilo, restricciones, semántica, análisis, propiedades, racionalizaciones, requerimientos, necesidades de los participantes. El trabajo en esta área está caracterizada por el desarrollo de lenguajes de descripción arquitectónica (ADLs).
2) **Modelos de framework**: Son similares a la vista estructural, pero su énfasis primario radica en la (usualmente una sola) estructura coherente del sistema completo, en vez de concentrarse en su composición. Los modelos de framework a menudo se refieren a dominios o clases de problemas específicos. El trabajo que ejemplifica esta variante incluye arquitecturas de software específicas de dominios, como CORBA, o modelos basados en CORBA, o repositorios de componentes específicos, como PRISM.

3) **Modelos dinámicos**: Enfatizan la cualidad conductual de los sistemas. “Dinámico” puede referirse a los cambios en la configuración del sistema, o a la dinámica involucrada en el progreso de la computación, tales como valores cambiantes de datos.
4) **Modelos de proceso**: Se concentran en la construcción de la arquitectura, y en los pasos o procesos involucrados en esa construcción. En esta perspectiva, la arquitectura es el resultado de seguir un argumento (*script*) de proceso. Esta vista se ejemplifica con el actual trabajo sobre programación de procesos para derivar arquitecturas.
5) **Modelos funcionales**: Una minoría considera la arquitectura como un conjunto de componentes funcionales, organizados en capas que proporcionan servicios hacia arriba. Es tal vez útil pensar en esta visión como un framework particular.

Ninguna de estas vistas excluye a las otras, ni representa un conflicto fundamental sobre lo que es o debe ser la AS. Por el contrario, representan un espectro en la comunidad de investigación sobre distintos énfasis que pueden aplicarse a la arquitectura: sobre sus partes constituyentes, su totalidad, la forma en que se comporta una vez construida, o el proceso de su construcción. Tomadas en su conjunto, destacan más bien un consenso.

Independientemente de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción. Revisamos las diversas definiciones del concepto de abstracción en un apartado específico más adelante en este estudio.

Es casi seguro que la percepción de la AS que prevalece entre quienes no han tenido contacto con ella, así como los estereotipos dominantes sobre su naturaleza, o los nombres que se escogerían como sus personalidades más importantes, difieren sustancialmente de lo que es el caso en el interior de la especialidad. Este sería acaso un ejercicio digno de llevarse a cabo alguna vez.

Conceptos fundamentales

Más allá de que hoy existan numerosos conceptos en el plano detallado de las técnicas y metodologías, la AS se articula alrededor de unos pocos conceptos y principios esenciales y unas pocas herramientas características.

Estilos

En el texto fundacional de la AS, Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales.

Sucintamente descriptos, los estilos conjugan elementos (o “componentes”, como se los llama aquí), conectores, configuraciones y restricciones. Al estipular los conectores como elemento de juicio de primera clase, el concepto de estilo, incidentalmente, se sitúa en un orden de discurso y de método que el modelado orientado a objetos en general y UML en

particular no cubren satisfactoriamente. La descripción de un estilo se puede formular en lenguaje natural o en diagramas, pero lo mejor es hacerlo en un lenguaje de descripción arquitectónica o en lenguajes formales de especificación.

A diferencia de los patrones de diseños, que son centenares, los estilos se ordenan en seis o siete clases fundamentales y unos veinte ejemplares, como máximo. Es digno de señalarse el empeño por subsumir todas las formas existentes de aplicaciones en un conjunto de dimensiones tan modestas. Las arquitecturas complejas o compuestas resultan del agregado o la composición de estilos más básicos. Algunos estilos típicos son las arquitecturas basadas en flujo de datos, las *peer-to-peer*, las de invocación implícita, las jerárquicas, las centradas en datos o las de intérprete-máquina virtual.

Hemos tratado el tema de las definiciones, los catálogos de estilos, las propiedades de cada uno, el lugar de los estilos en la AS y su relación con los patrones de diseño y con la estrategia arquitectónica de Microsoft en un documento separado, en torno del cual se podrán discutir las cuestiones relacionadas con ellos.

Lenguajes de descripción arquitectónica

Los lenguajes de descripción de arquitecturas, o ADLs, ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML. Los ADL difiere sustancialmente de UML, que al menos en su versión 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Los ADLs son bien conocidos en los estudios universitarios de AS, pero muy pocos arquitectos de industria parecen conocerlos y son menos aún quienes los utilizan como instrumento en el diseño arquitectónico de sus proyectos. Hay unos veinte ADLs de primera magnitud y tal vez unos cuarenta o cincuenta propuestos en ponencias que no han resistido el paso del tiempo o que no han encontrado su camino en el mercado. Se han analizado esos lenguajes descriptivos en un documento aparte, en el cual se invita asimismo a su discusión.

Frameworks y Vistas

En esta sección se examinarán primero las propuestas sobre organización de vistas desarrolladas en el contexto de los *frameworks* más influyentes. En segundo lugar, se analizará algo más formalmente la historia y el significado del concepto de vistas en sí, ya que en la mayor parte de la literatura no académica la significación precisa del concepto y su función técnica se suelen dar por sentadas o se definen a la ligera y de formas cambiantes.

Existen unos cuantos organismos de estándares (ISO, CEN, IEEE, OMG) que han codificado diferentes aspectos de la AS, cuando no la AS en su totalidad, con el objetivo

de homogeneizar la terminología, los modelos y los procedimientos. Los emergentes del trabajo de sus comités son especificaciones y recomendaciones de variada naturaleza, como RM-ODP, RUP, RDS, MDA, MOF, MEMO, XMI o IEEE 1471-2000. Casi cualquier combinación de tres o cuatro letras del alfabeto corresponde al acrónimo de algún estándar a tener en cuenta. La AS hace mención eventual de esas doctrinas y los académicos ocupan sillas preferenciales en los organismos, pero su tratamiento exhaustivo en la literatura de investigación decididamente no califica como uno de los grandes temas prioritarios. Las recomendaciones de los marcos se tratan con sumo respeto pero también con alguna reticencia, tal vez porque se estima que los organismos privilegiarían más un acuerdo regido por una necesidad de equidistancia que una adecuada fundamentación formal, porque cualquier versión del estándar que se mencione en un *paper* habrá caducado cuando se lo publique, o por alguna otra razón que dejamos abierta para que se discuta.

Hay una excepción, sin embargo. Tanto los marcos arquitectónicos como las metodologías de modelado de los organismos acostumbran ordenar las diferentes perspectivas de una arquitectura en términos de vistas (*views*). La mayoría de los frameworks y estrategias reconoce entre tres y seis vistas, que son las que se incluyen en el cuadro. Una vista es, para definirla sucintamente, un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado [Hil99].

Zachman (Niveles)	TOGAF (Arquitecturas)	4+1 (Vistas)	[BRJ99] (Vistas)	POSA (Vistas)	Microsoft (Vistas)
Scope	Negocios	Lógica	Diseño	Lógica	Lógica
Empresa	Datos	Proceso	Proceso	Proceso	Conceptual
Sistema lógico	Aplicación	Física	Implementación	Física	Física
Tecnología	Tecnología	Desarrollo	Despliegue	Desarrollo	
Representación		Casos de uso	Casos de uso		
Funcionamiento					

Tabla 1 - Vistas en los marcos de referencia

- El marco de referencia para la arquitectura empresarial de John Zachman [Zac87] identifica 36 vistas en la arquitectura (“celdas”) basadas en seis niveles (*scope*, empresa, sistema lógico, tecnología, representación detallada y funcionamiento empresarial) y seis aspectos (datos, función, red, gente, tiempo, motivación). En el uso corriente de AS se ha estimado que este modelo es excesivamente rígido y sobre-articulado. Parecería existir cierto consenso sobre su excesiva ambición y su posible obsolescencia. Los manuales recientes de ingeniería de software (por ejemplo [Pre02] [Pfl02]) suelen omitir toda referencia a este marco, que se estima perteneciente a una esfera de *Information Management* y estrategia empresarial, antes que inscripto en el campo de la arquitectura. El framework ha estado también bajo nutrido fuego crítico [Cib98] [Keen91]. No obstante, hay que reconocer que tres de las vistas propuestas por Zachman tan tempranamente como en 1982 (conceptual, lógica y física) se corresponden con el modelo de vistas de los marcos de referencia posteriores.
- El Modelo de Referencia para Procesamiento Distribuido Abierto (RM-ODP) es un estándar de ISO y de ITU (ex-CCITT) que define un marco para la especificación arquitectónica de grandes sistemas distribuidos. Define, entre otras cosas, cinco

puntos de vista (*viewpoints*) para un sistema y su entorno: empresa, información, computación, ingeniería y tecnología. Los cinco puntos de vista *no* corresponden a etapas de proceso de desarrollo o refinamiento. De los cuatro estándares básicos que componen el modelo, los dos primeros se refieren a la motivación general del mismo y a sus fundamentos conceptuales y analíticos; el tercero (ISO/IEC 10746-3; UTI-T X.903) a la arquitectura, definiendo los puntos de vistas referidos; y el cuarto (ISO/IEC 10746-4; UTI-T X.904) a la formalización de la semántica arquitectónica. RM-ODP se supone neutral en relación con la metodología, las formas de modelado y la tecnología a implementarse, pero recomienda el uso de lenguajes formales de especificación como LOTOS, ESTELLE, SDL o Z. Se supone que un modelo pensado con similares propósitos, como CORBA, debería ser 100% conforme con la referencia, pero en verdad no es así; CORBA, por ejemplo, no proporciona soporte para el *viewpoint* empresarial, su modelo computacional revela desviaciones significativas del ISO/IEC o el UTI-T correspondiente, su modelo de *binding* es más restringido (carece de *multicast* o *plug and play*) y aunque hay RFPs que documentan estas divergencias, hasta donde puede saberse permanecen aún sin resolver; en los *viewpoints* de ingeniería y tecnología las discrepancias son menores, pero son muchísimas, y en ninguno hay concordancia en la nomenclatura [DIR99]. Los modelos mayores de industria como COM o J2EE son todavía más divergentes y las verificaciones de conformidad son un trámite extremadamente complejo. RM-ODP, además, no especifica nada acerca de conectores entre sistemas, integración de tecnologías *legacy* o soporte de sistemas multi-paradigmas [Bez98]. Hoy en día la interoperabilidad se garantiza más a través de tecnologías comunes de formato y protocolo ligadas a XML, SOAP, BPEL4WS o WS-I (o mediante MDA, o programando un *wrapper*) que por conformidad con esta clase de recomendaciones en todos los puntos de un proceso distribuido.

- C4ISR Architecture Framework es el marco de referencia arquitectónico promovido por el Departamento de Defensa de Estados Unidos (DoD). Algunos de los otros marcos listados en esta sección se inspiran en él, como es el caso de TOGAF. En la versión 2 de C4I, completada en diciembre de 1997, la definición de arquitectura reconocida es exactamente la misma que después se promulgaría como canónica en IEEE 1471. Las vistas arquitectónicas homologadas son la Operacional (que identifica relaciones y necesidades de información), la de Sistemas (que vincula capacidades y características a requerimientos operacionales) y la Técnica (que prescribe estándares y convenciones).
- El marco de referencia arquitectónico de The Open Group (TOGAF) reconoce cuatro componentes principales, uno de los cuales es un *framework* de alto nivel que a su vez define cuatro vistas: Arquitectura de Negocios, Arquitectura de Datos/Información, Arquitectura de Aplicación y Arquitectura Tecnológica. The Open Group propone un modelo de descripción arquitectónica, Architecture Description Method (ADM) que se supone independiente de las técnicas de modelado, aunque en la versión 7 se propone Metis como herramienta.
- En 1995 Philippe Kruchten propuso su célebre modelo “4+1”, vinculado al Rational Unified Process (RUP), que define cuatro vistas diferentes de la arquitectura de software: (1) La vista lógica, que comprende las abstracciones fundamentales del

sistema a partir del dominio de problemas. (2) La vista de proceso: el conjunto de procesos de ejecución independiente a partir de las abstracciones anteriores. (3) La vista física: un mapeado del software sobre el hardware. (4) La vista de desarrollo: la organización estática de módulos en el entorno de desarrollo. El quinto elemento considera todos los anteriores en el contexto de casos de uso [Kru95]. Lo que académicamente se define como AS concierne a las dos primeras vistas. El modelo 4+1 se percibe hoy como un intento se reformular una arquitectura estructural y descriptiva en términos de objeto y de UML. Con todo, las cuatro vistas de Kruchten forman parte del repertorio estándar de los practicantes de la disciplina.

- En su introducción a UML (1.3), Grady Booch, James Rumbaugh e Ivar Jacobson han formulado un esquema de cinco vistas interrelacionadas que conforman la arquitectura de software, caracterizada en términos parecidos a los que uno esperaría encontrar en el discurso de la vertiente estructuralista. En esta perpectiva, la arquitectura de software (a la que se dedican muy pocas páginas) es un conjunto de decisiones significativas sobre (1) la organización de un sistema de software; (2) la selección de elementos estructurales y sus interfaces a través de los cuales se constituye el sistema; (3) su comportamiento, según resulta de las colaboraciones entre esos elementos; (4) la composición de esos elementos estructurales y de comportamiento en subsistemas progresivamente mayores; (5) el estilo arquitectónico que guía esta organización: los elementos estáticos y dinámicos y sus interfaces, sus colaboraciones y su composición. Los autores proporcionan luego un esquema de cinco vistas posibles de la arquitectura de un sistema: (1) La vista de casos de uso, como la perciben los usuarios, analistas y encargados de las pruebas; (2) la vista de diseño que comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución; (3) la vista de procesos que conforman los hilos y procesos que forman los mecanismos de sincronización y concurrencia; (4) la vista de implementación que incluye los componentes y archivos sobre el sistema físico; (5) la vista de despliegue que comprende los nodos que forma la topología de hardware sobre la que se ejecuta el sistema [BRJ99: 26-27]. Aunque las vistas no están expresadas en los mismos términos estructuralistas que campean en su caracterización de la arquitectura, y aunque la relación entre vistas y decisiones arquitectónicas es de simple yuxtaposición informal de ideas antes que de integración rigurosa, es natural inferir que las vistas que más claramente se vinculan con la semántica arquitectónica son la de diseño y la de proceso.
- En los albores de la moderna práctica de los patrones, Buschmann y otros presentan listas discrepantes de vistas en su texto popularmente conocido como *POSA* [BMR+96]. En la primera se las llama “arquitecturas”, y son: (1) Arquitectura conceptual: componentes, conectores; (2) Arquitectura de módulos: subsistemas, módulos, exportaciones, importaciones; (3) Arquitectura de código: archivos, directorios, bibliotecas, inclusiones; (4) Arquitectura de ejecución: tareas, hilos, procesos. La segunda lista de vistas, por su parte, incluye: (1) Vista lógica: el modelo de objetos del diseño, o un modelo correspondiente tal como un diagrama de relación; (2) Vista de proceso: aspectos de concurrencia y sincronización; (3) Vista física: el mapeo del software en el hardware y sus aspectos distribuidos; (4) Vista de desarrollo: la organización estática del software en su entorno de desarrollo. Esta

segunda lista coincide con el modelo 4+1 de Kruchten, pero sin tanto énfasis en el quinto elemento.

- Bass, Clements y Kazman presentan en 1998 una taxonomía de nueve vistas, decididamente sesgadas hacia el diseño concreto y la implementación: (1) Estructura de módulo; las unidades son asignaciones de tareas. (2) Estructura lógica o conceptual. Las unidades son abstracciones de los requerimientos funcionales del sistema. (3) Estructura de procesos o de coordinación. Las unidades son procesos o *threads.* (4) Estructura física. (5) Estructura de uso. Las unidades son procedimientos o módulos, vinculados por relaciones de presunción-de-presencia-correcta. (6) Estructura de llamados. Las unidades son usualmente (sub)procedimientos, vinculados por invocaciones o llamados. (7) Flujo de datos. Las unidades son programas o módulos, la relación es de envío de datos. (8) Flujo de control; las unidades son programas, módulos o estados del sistema. (9) Estructura de clases. Las unidades son objetos [BCK98]. Esta taxonomía de vista no coincide con ninguna otra.
- La recomendación IEEE Std 1471-2000 procura establecer una base común para la descripción de arquitecturas de software, e implementa para ello tres términos básicos, que son arquitectura, vista y punto de vista. La *arquitectura* se define como la organización fundamental de un sistema, encarnada en sus componentes, las relaciones entre ellos y con su entorno, y los principios que gobiernan su diseño y evolución. Los elementos que resultan definitorios en la utilidad, costo y riego de un sistema son en ocasiones físicos y otras veces lógicos. En otros casos más, son principios permanentes o patrones que generan estructuras organizacionales duraderas. Términos como *vista* o *punto de vista* son también centrales. En la recomendación se los utiliza en un sentido ligeramente distinto al del uso común. Aunque reflejan el uso establecido en los estándares y en la investigación de ingeniería, el propósito del estándar es introducir un grado de formalización homogeneizando informalmente la nomenclatura. En dicha nomenclatura, un punto de vista (*viewpoint*) define un patrón o plantilla (*template*) para representar un conjunto de incumbencias (*concerns*) relativo a una arquitectura, mientras que una vista (*view*) es la representación concreta de un sistema en particular desde una perspectiva unitaria. Un punto de vista permite la formalización de grupos de modelos. Una vista también se compone de modelos, aunque posee también atributos adicionales. Los modelos proporcionan la descripción específica, o contenido, de una arquitectura. Por ejemplo, una vista estructural consistiría de un conjunto de modelos de la estructura del sistema. Los elementos de tales modelos incluirían componentes identificables y sus interfaces, así como interconexiones entre los componentes. La concordancia entre la recomendación de IEEE y el concepto de estilo se establece con claridad en términos del llamado “punto de vista estructural”. Otros puntos de vista reconocidos en la recomendación son el conductual y el de interconexión física. El punto de vista estructural ha sido motivado (afirman los redactores del estándar) por el trabajo en lenguajes de descripción arquitectónica (ADLs). El punto de vista estructural, dicen, se ha desarrollado en el campo de la AS desde 1994 y es hoy de amplio uso.
- La estrategia de arquitectura de Microsoft define, en consonancia con las conceptualizaciones más generalizadas, cuatro vistas, ocasionalmente llamadas también

arquitecturas: Negocios, Aplicación, Información y Tecnología [Platt02]. La vista que aquí interesa es la de la aplicación, que incluye, entre otras cosas: (1) Descripciones de servicios automatizados que dan soporte a los procesos de negocios; (2) descripciones de las interacciones e interdependencias (interfaces) de los sistemas aplicativos de la organización, y (3) planes para el desarrollo de nuevas aplicaciones y la revisión de las antiguas, basados en los objetivos de la empresa y la evolución de las plataformas tecnológicas. Cada arquitectura, a su vez, se articula en vistas también familiares desde los días de OMT que son (1) la Vista Conceptual, cercana a la semántica de negocios y a la percepción de los usuarios no técnicos; (2) la Vista Lógica, que define los componentes funcionales y su relación en el interior de un sistema, en base a la cual los arquitectos construyen modelos de aplicación que representan la perspectiva lógica de la arquitectura de una aplicación; (3) la Vista Física, que es la menos abstracta y que ilustra los componentes específicos de una implementación y sus relaciones.

Ahora que se han examinado cuáles son las vistas que proponen los diferentes frameworks, habría que precisar mejor el significado específicamente arquitectónico de las vistas. Como expresa Rich Hilliard [Hil99], a quien seguimos en este tratamiento, aunque expresiones tales como *múltiples vistas* son algo así como el Santo Grial de la ingeniería de software, de requerimientos y de sistemas, su estatus en la AS es bastante más oscuro. Las razones para ello son múltiples. En primer lugar, no existe una fundamentación coherente para su uso en la disciplina. En segundo término, muchos estudiosos las consideran problemáticas, porque la existencia de múltiples vistas introduce problemas de integración y consistencia entre las diferentes vistas. Sin embargo, los arquitectos practicantes las usan de todas maneras, porque simplifican la visualización de sistemas complejos.

La idea de contemplar un sistema complejo desde múltiples puntos de vista no es nativa de la AS contemporánea, ni es una invención de Kruchten, sino que se origina por lo menos en la década de 1970, en el trabajo de Douglas Ross sobre análisis estructurado [Ross77]. La motivación para introducir múltiples vistas radica en la separación de incumbencias (*separations of concerns*). Las vistas se introdujeron como una herramienta conceptual para poder manejar la complejidad de lo que ya por aquel entonces se llamaban artefactos, tales como especificaciones de requerimientos o modelos de diseño. En las contribuciones más tempranas, las múltiples vistas de un modelo se basaban en perspectivas fijas, puntos de vistas o *viewpoints*; casi siempre los puntos de vista eran dos, el funcional y el de datos, ninguno de los cuales aparece en arquitectura.

En la década de 1980, sin embargo, las vistas comenzaron a multiplicarse, al punto que se realizaron *surveys* interdisciplinarios y se organizaron conferencias específicas sobre la cuestión, como *Viewpoints'96*. No hay un límite necesario para el número de vistas posibles, ni un procedimiento formal para establecer lo que una vista debe o no abstraer. El estándar IEEE 1471 no delimita el número posible de vistas, ya que se estima que no puede haber acuerdo en ello, pero señala lineamientos para su constitución y considera que un *viewpoint* es a una vista como una clase es a un objeto.

Hay una "lista corta" de vistas que se usa en los textos generales de AS y una "lista larga" que gira en torno de UML, el cual especifica nueve clases de diagramas correspondientes a ocho vistas, como se indica en el cuadro. Diferentes textos de los mismos autores, por

ejemplo [BRJ99] y [RJB00], utilizan distintas terminologías no conciliadas; vistas y puntos de vista no siempre se caracterizan como conceptos distintos y el uso de la terminología, aún en el interior de cada texto, es informal e inconsistente. Es difícil creer que esto se encuentra "unificado" de alguna manera. Cuando los promotores de UML hablan de arquitectura, a instancias de Kruchten, cambian su modelo de vistas por uno que se refiere no a puntos de perspectiva o a incumbencias de los participantes, sino a niveles de abstracción; pero aún así, como se ha visto, su definición de arquitectura difiere de la definición estándar.

Área	Vista	Diagramas	Conceptos principales
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagramas de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización
Dinámica	Vista de máquinas de estados	Diagrama de estados	Estado, evento, transición, acción
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación
		Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje
Gestión del modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo

Tabla 2 - Vistas y diagramas de UML, basado en [RJB00: 22]

Como lo subraya Hilliard, en sus textos iniciales la modalidad clásica de la AS, encarnada en Garlan y Shaw, no habla de vistas en absoluto; la AS clásica se funda en una vista singular e implícita, de carácter estructural. Muchos arquitectos de la corriente principal evitan hablar de vistas, porque cuando las ellas proliferan se hace necesario o bien elaborar lenguajes formales específicos para tratar cada una de ellas, o bien multiplicar salvajemente las extensiones del lenguaje unificado. Sin duda las vistas son una simplificación conveniente, o más bien un principio de orden; pero su abundancia y sus complicadas relaciones recíprocas generan también nuevos órdenes de complejidad.

Podría discutirse el concepto y la articulación de las diferentes vistas un largo rato, y de hecho los muchos simposios que han habido sobre el particular fueron de interés pero inconcluyentes. Se acostumbra poner las vistas que denotan "niveles de abstracción" en cuadros superpuestos, por ejemplo, como si fueran análogas a las capas del modelo OSI, pero ¿son ambas representaciones estrictamente homólogas? ¿Constituye el conjunto de las vistas un sistema? ¿Por qué cada vez que se enumeran los artefactos característicos de cada vista aparece la palabra "etcétera" o la expresión "elementos principales"?

Procesos y Metodologías

En los diferentes marcos, las vistas estáticas se corresponden con las perspectivas particulares de los diferentes participantes (*stakeholders*), mientras que las vistas

dinámicas tienen que ver con etapas del proceso, ciclo de vida o metodología, caracterizadas como requerimiento, análisis, diseño (o construcción, o modelado), implementación, integración (prueba de conformidad, *testing*, evaluación). La terminología, lo mismo que la articulación temporal del proceso o el ciclo, depende de cada marco. Llegando al territorio de la metodología, hay que decir que durante varios años la AS discurrió sin elaborarlas más que circunstancialmente, como si se estimara compatible con las prácticas establecidas en ingeniería de software, cualesquiera fuesen: RUP, RAD, RDS, ARIS, PERA, CIMOSA, GRAI, GERAM, CMM. Hoy en día la metodología dominante en la industria es tal vez el Modelo de Madurez de la Capacidad (CMM), aunque el SEI no la considera formalmente como tal.

Desde 1998 y cada vez con mayor intensidad, sin embargo, el SEI y otros organismos comenzaron a elaborar métodos específicos de procesos de ingeniería que sistematizan el rol de la arquitectura en la totalidad del proceso, desde la elicitación de requerimientos hasta la terminación. Algunos de esos métodos son Architecture Based Design (ABD), Software Architecture Analysis Method (SAAM), Quality Attribute Workshops (QAW), Quality Attribute-Oriented Software Architecture Design Method (QASAR), Attribute-Driven Design (ADD), Architecture Tradeoff Analysis Method (ATAM), Active Review for Intermediate Design (ARID), Cost-Benefits Analysis Method (CBAM), Family-Architecture Analysis Method (FAAM), Architecture Level Modifiability Analysis (ALMA), y Software Architecture Comparison Analysis Method (SACAM). Al lado de esos métodos está surgiendo un nutrido conjunto de técnicas de documentación; métodos, técnicas y estudios de casos se analizarán en este estudio en documentos separados. Habiendo al menos una docena de métodos complejamente engranados entre sí, el lector puede perderse con facilidad en un laberinto bibliográfico donde no siempre se discierne cuáles de estos métodos incluyen a cuáles otros, cuáles son sus relaciones con técnicas consagradas de ingeniería, cuáles se encuentran vigentes y cuáles obsoletos. Un documento actual que describe a grandes rasgos la mayoría de esos métodos es [KNK03].

En general, la AS de la corriente principal todavía no se ha expedido en relación con los llamados métodos ágiles. No hay un solo método, sino una multiplicidad de posturas más o menos radicales y combativas: Extreme Programming, SCRUM, Crystal Methods Framework, Feature-Driven Development, DSDM, Lean Development, Adaptive Software Development, Agile Modeling, Pragmatic Programming [ASR+02] [CLC03]. De hecho, la comunidad de los metodólogos, que opera a una cierta distancia de las iniciativas formales de la AS, se encuentra dividida tras lo que ha sido y sigue siendo "el gran debate metodológico" entre los métodos pesados (o rigurosos) de tipo SEI/CMM por un lado y los métodos ágiles por el otro [Hig01]. Los teóricos de cada bando han sido, entre otros, Tom De Marco, Ed Yourdon y Tim Lister en la facción rigurosa y Ken Orr, Jim Highsmith, Martin Fowler y Michael Jackson del lado ágil-extremo, con Philippe Kruchten y RUP buscando establecerse en ambos terrenos.

Ambos grupos operan en el contexto de la crisis del software, que se da por sentada, alegando que es la mentalidad del bando opuesto lo que la ha ocasionado. Los pesados, mirados por los ágiles como formalistas fracasados, enfatizan el modelado, el control y la documentación escrupulosa; los ágiles, acusados por los pesados de *hackers* irresponsables que pretenden imponer sus juguetes en la empresa, no sólo desprecian los modelos (formales o informales) sino que incluso ocasionalmente ponen en tela de juicio

hasta a los propios patrones de diseño [Fow01]. Ese tema también será tratado en un estudio aparte.

En lo que respecta a la estrategia arquitectónica de Microsoft, el marco general de Microsoft Solutions Framework, versión 3, no se expide sobre metodologías específicas y da cabida a una gran cantidad de alternativas, desde las densas a las ligeras. Abundan en MSF 3 referencias a métodos rápidos y ágiles, a través de citas y conceptos de una de las figuras cardinales de Cutter Consortium, Martin Fowler. La documentación de MSF ratifica su adecuación tanto para el CMM de SEI o UPM como para los métodos ágiles en ambientes que requieren un alto grado de adaptabilidad [MS03].

Abstracción

El concepto de abstracción (que a veces se usa en el sentido del proceso de abstraer, otra para designar una entidad) ha sufrido también diversas acepciones, con un núcleo de significados común. Las diferencias en el uso del concepto de abstracción ayudan también a identificar las diversas corrientes en el seno de la AS. La definición que proporciona Grady Booch, por ejemplo, revela que el autor identifica la abstracción arquitectónica con el encapsulamiento propio de la tecnología de objetos: “Una abstracción –escribe Booch– denota las características esenciales de un objeto que lo distinguen de otras clases de objeto y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador” [Boo91].

El concepto de abstracción (que a veces se usa en el sentido del proceso de abstraer, otra para designar una entidad abstracta) ha sufrido también diversas formulaciones sintácticas, con un núcleo de significados común. En último análisis, la abstracción siempre conlleva una heurística positiva al lado de una negación. Tanto para la IEEE como para Rumbaugh, Shaw y otros autores, la abstracción consiste en extraer las propiedades esenciales, o identificar los aspectos importantes, o examinar selectivamente ciertos aspectos de un problema, posponiendo o ignorando los detalles menos sustanciales, distractivos o irrelevantes.

La idea de abstracción forma parte de lo que acaso sea la pieza conceptual más importante de la AS, el concepto de estilo; un estilo se identifica a grandes rasgos o, como se dice habitualmente, en un estilo “menos es más”. La misma idea prevalece en todos los conceptos y procedimientos que se consideran arquitectónicos. Para Len Bass, Paul Clements y Rick Kazman [BCK98], si una decisión debe posponerse hasta el momento de tratar las cosas a un bajo nivel, no se trata de una decisión de arquitectura. Clements y Northrop [CN96] sostienen que el trabajo de Garlan y Shaw sobre los estilos arquitectónicos nos enseña que aunque los programas pueden combinarse de maneras prácticamente infinitas, hay mucho que ganar si nos restringimos a un conjunto relativamente pequeño de elecciones cuando se trata de cooperación e interacción. Las ventajas incluyen mejor reutilización, mejores análisis, menor tiempo se selección y mayor interoperabilidad. “Menos es más” figura, de hecho, en el título y en los contenidos de numerosos *papers* de la profesión [MB02] [CN96]. Conceptos como el de estilo, o marcos como MSF, revelan su naturaleza arquitectónica en su abstracción y en su generalidad.

Escenarios

Esta es una noción arquitectónica importante y se encuentra en la base de muchos de los métodos de diseño y desarrollo basados en arquitectura, como ALMA, SAAM y ATAM. Hay que ser precavidos y advertir desde el comienzo que lo que habitualmente se traduce como "escenario" no es estrictamente lo que en lengua castellana se designa como tal; la traducción correcta debería ser más bien "guión" o "libreto". La traducción literal del término no hace más que aportar confusión. Desde Kruchten en adelante, se reconoce que los escenarios se dividen en dos categorías: casos de uso (secuencias de responsabilidades) y casos de cambio (modificaciones propuestas al sistema).

Los escenarios han sido básicamente técnicas que se implementan en la elicitación de los requerimientos, particularmente en relación a los operadores de sistemas. Típicamente, la técnica comienza instrumentando sesiones de *brainstorming*. También se han utilizado escenarios como método para comparar alternativas de diseño. Los escenarios describen una utilización anticipada o deseada del sistema, y típicamente se expresan en una frase; Kazman, Abowd, Bass y Clements proponen también llamarlos viñetas [KAB+96].

Según algunas definiciones, como la de Clements y Northrop [CN96], los escenarios son algo así como libretos (en el sentido teatral o cinematográfico del término) correspondientes a las distintas piezas de funcionalidad de un sistema. Se los considera útiles para analizar una vista determinada [Cle95a] o para mostrar la forma en que los elementos de múltiples vistas se relacionan entre sí [Kru95]. Pueden concebirse también como una abstracción de los requerimientos más importantes de un sistema. Los escenarios se describen mediante texto común en prosa utilizando lo que se llama un *script* y a veces se describen mediante dibujos, como por ejemplo diagramas de interacción de objeto. Se acostumbra utilizar UML (en el contexto de 4+1) no tanto como recurso de modelado que después generará alguna clase de código, sino como instrumento de dibujo más o menos informal; pero los propios manuales de UML y los expertos mundiales en casos de uso (David Anderson, Martin Fowler, Alistair Cockburn) recomiendan desarrollar los escenarios de requerimiento en texto, no en diagramas. Algunos autores estiman que se trata de una herramienta importante para relacionar vistas arquitectónicas, porque recorriendo un escenario puede mostrar las formas en que fragmentos o escenas de esas vistas se corresponden entre sí. Los métodos propios de la arquitectura basada en escenarios se analizan también en un documento aparte.

Campos de la Arquitectura de Software

La AS es hoy en día un conjunto inmenso y heterogéneo de áreas de investigación teórica y de formulación práctica, por lo que conviene aunque más no sea enumerar algunos de sus campos y sus focos. Una semblanza semejante (de la que aquí se proporciona sólo un rudimento) dudosamente debería ser estática. La AS comenzó siendo una abstracción descriptiva puntual que en los primeros años no investigó de manera sistemática ni las relaciones que la vinculaban con los requerimientos previos, ni los pasos metodológicos a dar luego para comenzar a componer el diseño. Pero esa sincronicidad estructuralista no pudo sostenerse. Por el contrario, daría la impresión que, a medida que pasa el tiempo, la AS tiende a redefinir todos y cada uno de los aspectos de la disciplina madre, la

ingeniería de software, sólo que a un mayor nivel de abstracción y agregando una nueva dimensión reflexiva en lo que concierne a la fundamentación formal del proceso.

Hay unas pocas caracterizaciones (y mucha actividad de copiado y pegado) en torno de las áreas que componen el territorio. David Garlan y Dewayne Perry, en su introducción al volumen de abril de 1995 de IEEE Transactions on Software Engineering dedicado a la AS, en el cual se delinean las áreas de investigación más promisorias, enumeran las siguientes:

- Lenguajes de descripción de arquitecturas
- Fundamentos formales de la AS (bases matemáticas, caracterizaciones formales de propiedades extra-funcionales tales como mantenibilidad, teorías de la interconexión, etcétera).
- Técnicas de análisis arquitectónicas
- Métodos de desarrollo basados en arquitectura
- Recuperación y reutilización de arquitectura
- Codificación y guía arquitectónica
- Herramientas y ambientes de diseño arquitectónico
- Estudios de casos

Fundamental en la concepción de Clements y Northrop [CN96] es el criterio de reusabilidad como uno de los aspectos que más hacen por justificar la disciplina misma. Según estos autores, el estudio actual de la AS puede ser visto como un esfuerzo ex post facto para proporcionar un almacén estructurado de este tipo de información reutilizable de diseño de alto nivel propio de una familia (en el sentido de Parnas). De tal manera, las decisiones de alto nivel inherentes a cada miembro de una familia de programas no necesitan ser re-inventadas, re-validadas y re-descriptas. Un razonamiento arquitectónico es además un argumento sobre las cuestiones estructurales de un sistema. Se diría también que el concepto de estilo es la encarnación principal del principio de reusabilidad en el plano arquitectónico.

Paul Clements [Cle96b] define cinco temas fundamentales en torno de los cuales se agrupa la disciplina:

- **Diseño o selección de la arquitectura:** Cómo crear o seleccionar una arquitectura en base de requerimientos funcionales, de performance o de calidad.
- **Representación de la arquitectura:** Cómo comunicar una arquitectura. Este problema se ha manifestado como el problema de la representación de arquitecturas utilizando recursos lingüísticos, pero el problema también incluye la selección del conjunto de información a ser comunicada.
- **Evaluación y análisis de la arquitectura:** Cómo analizar una arquitectura para predecir cualidades del sistema en que se manifiesta. Un problema semejante es cómo comparar y escoger entre diversas arquitecturas en competencia.
- **Desarrollo y evolución basados en arquitectura:** Cómo construir y mantener un sistema dada una representación de la cual se cree que es la arquitectura que resolverá el problema correspondiente.

- **Recuperación de la arquitectura:** Cómo hacer que un sistema *legacy* evolucione cuando los cambios afectan su estructura; para los sistemas de los que se carezca de documentación confiable, esto involucra primero una "arqueología arquitectónica" que extraiga su arquitectura.

Mary Shaw [Shaw01] considera que en el 2001 los campos más promisorios de la AS siguen teniendo que ver con el tratamiento sistemático de los estilos, el desarrollo de lenguajes de descripción arquitectónica, la formulación de metodologías y (ahora) el trabajo con patrones de diseño. Se requieren todavía modelos precisos que permitan razonar sobre las propiedades de una arquitectura y verificar su consistencia y completitud, así como la automatización del proceso de análisis, diseño y síntesis. Sugiere que debe aprenderse una lección a partir de la experiencia de la ingeniería de software, la cual no obstante haberse desenvuelto durante treinta años no ha logrado plasmar un conjunto de paradigmas de investigación comparable al de otras áreas de las ciencias de la computación. Estima que la AS se encuentra ya en su fase de desarrollo y extensión, pero que tanto las ideas como las herramientas distan de estar maduras.

Un campo que no figura en estas listas pero sobre el cual se está trabajando intensamente es en el de la coordinación de los ADLs que sobrevivan con UML 2.0 por un lado y con XML por el otro. Ningún lenguaje de descripción arquitectónica en el futuro próximo (excepto los que tengan un nicho técnico muy particular) será viable si no satisface esos dos requisitos.

Los ejercicios que pueden hacerse para precisar los campos de la AS son incontables. Ahora que la AS se ha abismado en el desarrollo de metodologías, hace falta, por ejemplo, establecer con más claridad en qué difieren sus elaboraciones en torno del diseño, del análisis de requerimientos o de justificación económica de las llevadas a cabo por la ingeniería de software. Asimismo, se está esperando todavía una lista sistemática y exhaustiva que describa los dominios de incumbencia de la disciplina, así como un examen del riesgo de duplicación de esfuerzos entre campos disciplinarios mal comunicados, una situación que a primera vista parecería contradictoria con el principio de reusabilidad.

Modalidades y tendencias

En la década de 1990 se establece definitivamente la AS como un dominio todavía hoy separado de manera confusa de ese marco global que es la ingeniería y de esa práctica puntual que es el diseño. Aunque no hay un discurso explícito y autoconsciente sobre escuelas de AS, ni se ha publicado un estudio reconocido y sistemático que analice las particularidades de cada una, en la actualidad se pueden distinguir a grandes rasgos unas seis corrientes. Algunas distinciones están implícitas por ejemplo en [MT00], pero la bibliografía sobre corrientes y alternativas prácticamente no existe y la que sigue habrá de ser por un tiempo una de las pocas propuestas contemporáneas sobre el particular.

Ahora bien, articular una taxonomía de estrategias no admite una solución simple y determinista. En distintos momentos de su trayectoria, algunos practicantes de la AS se mueven ocasionalmente de una táctica a otra, o evolucionan de un punto de vista más genérico a otro más particular, o realizan diferentes trabajos operando en marcos distintos. Además, con la excepción del "gran debate metodológico" entre métodos

pesados y ligeros [Hig01], las discusiones entre las distintas posturas rara vez se han manifestado como choques frontales entre ideologías irreconciliables, por lo que a menudo hay que leer entre líneas para darse cuenta que una afirmación cualquiera es una crítica a otra manera de ver las cosas, o trasunta una toma definida de posición. Fuera de la metodología, el único factor reconocible de discordia ha sido, hasta la fecha, la preminencia de UML y el diseño orientado a objetos. Todo lo demás parece ser más o menos negociable.

La división preliminar de escuelas de AS que proponemos es la siguiente:

1) **Arquitectura como etapa de ingeniería y diseño orientada a objetos**. Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado estrechamente al mundo de UML y Rational. No cabe duda que se trata de una corriente específica: Rumbaugh, Jacobson y Booch han sido llamados "Los Tres Amigos"; de lo que sí puede dudarse es que se trate de una postura *arquitectónica*. En este postura, la arquitectura se restringe a las fases iniciales y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está sistemáticamente ligada al requerimiento que viene antes o a la composición del diseño que viene después. Lo que sigue al momento arquitectónico es *business as usual*, y a cualquier configuración, topología o morfología de las piezas del sistema se la llama arquitectura. En esta escuela, si bien se reconoce el valor primordial de la abstracción (nadie después de Dijkstra osaría oponerse a ello) y del ocultamiento de información promovido por Parnas, estos conceptos tienen que ver más con el encapsulamiento en clases y objetos que con la visión de conjunto arquitectónica. Para este movimiento, la arquitectura se confunde también con el modelado y el diseño, los cuales constituyen los conceptos dominantes. En esta corriente se manifiesta predilección por un modelado denso y una profusión de diagramas, tendiente al modelo metodológico CMM o a UPM; no hay, empero, una precripción formal. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. Cuando aquí se habla de estilos, se los confunde con patrones arquitectónicos o de diseño [Lar03]. Jamás se hace referencia a los lenguajes de descripción arquitectónica, que representan uno de los *assets* reconocidos de la AS; sucede como si la disponibilidad de un lenguaje unificado de modelado los tornara superfluos. La definición de arquitectura que se promueve en esta corriente tiene que ver con aspectos formales a la hora del desarrollo; esta arquitectura es isomorfa a la estructura de las piezas de código. Una definición típica y demostrativa sería la de Grady Booch; para él, la AS es "la estructura lógica y física de un sistema, forjada por todas las decisiones estratégicas y tácticas que se aplican durante el desarrollo" [Boo91]. Otras definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten [Kru95] [BRJ99: 26-28].

2) **Arquitectura estructural**, basada en un modelo estático de estilos, ADLs y vistas. Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory

Abowd, John Ockerbloom. Se trata también de la visión de la AS dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la AS como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. En el interior del movimiento se pueden observar distintas divisiones. Hay una variante informal de "boxología" y una vertiente más formalista, representada por el grupo de Mark Moriconi en el SRI de Menlo Park. En principio se pueden reconocer tres modalidades en cuanto a la formalización; los más informales utilizan descripciones verbales o diagramas de cajas, los de talante intermedio se sirven de ADLs y los más exigentes usan lenguajes formales de especificación como CHAM y Z. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos. Mientras algunos participantes excluyen el modelo de datos de las incumbencias de la AS (Shaw, Garlan, etc), otros insisten en su relevancia (Medvidovic, Taylor). Todo estructuralismo es estático; hasta fines del siglo XX, no está muy claro el tema de la posición del modelado arquitectónico en el ciclo de vida.

3) **Estructuralismo arquitectónico radical**. Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más confrontativa con el mundo UML. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la AS y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación. Dado que pueden existir dudas sobre la materialidad de esta corriente como tal, proporcionamos referencias bibliográficas masivas que corroboran su existencia [Abd00] [AW99] [DDT99] [Dou00] [GAD+02] [GarS/f] [Gli00] [HNS99] [KroS/f] [Sch00] [StöS/f] [Tem99] [Tem01]. Pasaremos revista a los argumentos más fuertes en contra de UML emanados de este movimiento en el capítulo correspondiente del documento sobre lenguajes de descripción arquitectónica.

4) **Arquitectura basada en patrones**.[1] Si bien reconoce la importancia de un modelo emanado históricamente del diseño orientado a objetos, esta corriente surgida hacia 1996 no se encuentra tan rígidamente vinculada a UML en el modelado, ni a CMM en la metodología. El texto sobre patrones que esta variante reconoce como referencia es la serie *POSA* de Buschmann y otros [BMR+96] y secundariamente el texto de la Banda de los Cuatro [Gof95]. La diferencia entre ambos textos sagrados de la comunidad de patrones no es menor; en el primero, la expresión "Software Architecture" figura en el mismo título; el segundo se llama *Design Patterns: Elements of reusable Object-Oriented software* y su tratamiento de la arquitectura es mínimo. En esta manifestación de la AS prevalece cierta tolerancia hacia modelos de proceso tácticos, no tan macroscópicos, y eventualmente se expresa cierta simpatía por las ideas de Martin Fowler y las premisas de la programación extrema. El diseño

[1] Se han definido los patrones arquitectónicos y de diseño en el documento sobre Estilos en Arquitectura de Software [*Ref].

consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura [Shaw96] [MKM+96] [MKM+97].

5) **Arquitectura procesual**. Desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos (no todos) los arquitectos de Carnegie Mellon de la primera generación y muchos nombres nuevos de la segunda: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci, Charles Weinstock. Intenta establecer modelos de ciclo de vida y técnicas de elicitación de requerimientos, *brainstorming*, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software. Toda la documentación puede encontrarse ordenada en el SEI, pero no se mezcla jamás con la de CMM, a la que redefine de punta a punta. Otras variantes dentro de la corriente procesual caracterizan de otras maneras de etapas del proceso: extracción de arquitectura, generalización, reutilización [WL97].

6) **Arquitectura basada en escenarios**. Es la corriente más nueva. Se trata de un movimiento predominantemente europeo, con centro en Holanda. Recupera el nexo de la AS con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. Los teóricos y practicantes de esta modalidad de arquitectura se inscriben dentro del canon delineado por la arquitectura procesual, respecto de la cual el movimiento constituye una especialización. En esta corriente suele utilizarse diagramas de casos de uso UML como herramienta informal u ocasional, dado que los casos de uso son uno de los escenarios posibles. Los casos de uso *no* están orientados a objeto. Los autores vinculados con esta modalidad han sido, aparte de los codificadores de ATAM, CBAM, QASAR y demás métodos del SEI, los arquitectos holandeses de la Universidad Técnica de Eindhoven, de la Universidad Brije, de la Universidad de Groningen y de Philips Research: Mugurel Ionita, Dieter Hammer, Henk Obbink, Hans de Bruin, Hans van Vliet, Eelke Folmer, Jilles van Gurp, Jan Bosch. La profusión de holandeses es significativa; la Universidad de Eindhoven es, incidentalmente, el lugar en el que surgió lo que P. I. Sharp propopía llamar la escuela arquitectónica de Dijkstra.

En todos los simposios han habido intentos de fundación de otras variedades de AS, tales como una arquitectura adaptativa inspirada en ideas de la programación genética o en teorías de la complejidad, la auto-organización, el caos y los fractales, una arquitectura centrada en la acción que recurre a la inteligencia artificial heideggeriana o al posmodernismo, y una arquitectura epistemológicamente reflexiva que tiene a Popper o a Kuhn entre sus referentes; consideramos preferible omitirlas, porque por el momento ni su masa crítica es notoria ni su mensaje parece sustancial [DD94] [HHr+96] [ZHH+99] [Hock00]. Pero hay al menos un movimiento cismático digno de tomarse más en serio; a esta altura de los acontecimientos es muy posible que pueda hablarse también de una anti-arquitectura, que en nombre de los métodos heterodoxos se opone tanto al modelado orientado a objetos y los métodos sobredocumentados impulsados por consultoras corporativas como a la abstracción arquitectónica que viene de la academia. El Manifiesto por la Programación Agil, en efecto, valoriza:

- Los individuos y las interacciones por encima de los procesos y las herramientas
- El software que funciona por encima de la documentación exhaustiva
- La colaboración con el cliente por encima de la negociación contractual

- La respuesta al cambio por encima del seguimiento de un plan

Habrá oportunidad de desarrollar el tratamiento de estas metodologías en otros documentos de la serie.

Hay muchas formas, por cierto, de organizar el panorama de las tendencias y las escuelas. Discernir la naturaleza de cada una puede tener efectos prácticos a la hora de comprender antagonismos, concordancias, sesgos, disyuntivas, incompatibilidades, fuentes de recursos. Nadie ha analizado mejor los límites de una herramienta como UML o de los métodos ultra-formales, por ejemplo, que los que se oponen a su uso por razones teóricas precisas, por más que éstas sean interesadas. Un buen ejercicio sería aplicar vistas y perspectivas diferentes a las que han regido esta clasificación informal, y proponer en función de otros criterios o de métodos más precisos de composición y referencias cruzadas, taxonomías alternativas de las líneas de pensamiento vigentes en la AS en la actualidad.

Diferencias entre Arquitectura y Diseño

Una vez que se reconoce la diferencia, que nunca debió ser menos que obvia, entre diseño e implementación, o entre vistas conceptuales y vistas tecnológicas ¿Es la AS solamente otra palabra para designar el diseño? Como suele suceder, no hay una sola respuesta, y las que hay no son taxativas. La comunidad de AS, en particular la de extracción académica, sostiene que ésta difiere sustancialmente del mero diseño. Pero Taylor y Medvidovic [TM00], por ejemplo, señalan que la literatura actual mantiene en un estado ambiguo la relación entre ambos campos, albergando diferentes interpretaciones y posturas:

1) Una postura afirma que arquitectura y diseño son lo mismo.
2) Otra, en cambio, alega que la arquitectura se encuentra en un nivel de abstracción por encima del diseño, o es simplemente otro paso (un artefacto) en el proceso de desarrollo de software.
3) Una tercera establece que la arquitectura es algo nuevo y en alguna medida diferente del diseño (pero de qué manera y en qué medida se dejan sin especificar).

Taylor y Medvidovic estiman que la segunda interpretación es la que se encuentra más cerca de la verdad. En alguna medida, la arquitectura y el diseño sirven al mismo propósito. Sin embargo, el foco de la AS en la estructura del sistema y en las interconexiones la distingue del diseño de software tradicional, tales como el diseño orientado a objetos, que se concentra más en el modelado de abstracciones de más bajo nivel, tales como algoritmos y tipos de datos. A medida que la arquitectura de alto nivel se refina, sus conectores pueden perder prominencia, distribuyéndose a través de los elementos arquitectónicos de más bajo nivel, resultando en la transformación de la arquitectura en diseño.

En su reciente libro sobre el arte de la AS, Stephen Albin [Alb03] se pregunta en qué difiere ella de las metodologías de diseño bien conocidas como la orientación a objetos. La AS, se contesta, es una metáfora relativamente nueva en materia de diseño de software y en realidad abarca también las metodologías de diseño, así como metodologías de análisis. El arquitecto de software contemporáneo, escribe Albin, ejecuta una

combinación de roles como los de analista de sistemas, diseñador de sistemas e ingeniero de software. Pero la arquitectura es más que una recolocación de funciones. Esas funciones pueden seguir siendo ejecutadas por otros, pero ahora caen comúnmente bajo la orquestación del *chief architect*. El concepto de arquitectura intenta subsumir las actividades de análisis y diseño en un framework de diseño más amplio y más coherente. Las organizaciones se están dando cuenta que el alto costo del desarrollo de software requiere ser sometido a algún control y que muchas de las ventajas prometidas por las metodologías aún no se han materializado. Pero la arquitectura es algo más integrado que la suma del análisis por un lado y el diseño por el otro. La integración de metodologías y modelos, concluye Albin, es lo que distingue la AS de la simple yuxtaposición de técnicas de análisis y de diseño.

Para Shaw y Garlan [SG96] la AS es el primer paso en la producción de un diseño de software, en una secuencia que distingue tres pasos:

1) Arquitectura. Asocia las capacidades del sistema especificadas en el requerimiento con los componentes del sistema que habrán de implementarla. La descripción arquitectónica incluye componentes y conectores (en términos de estilos) y la definición de operadores que crean sistemas a partir de subsistemas o, en otros términos, componen estilos complejos a partir de estilos simples.
2) Diseño del código. Comprende algoritmos y estructuras de datos; los componentes son aquí primitivas del lenguaje de programación, tales como números, caracteres, punteros e hilos de control. También hay operadores primitivos.
3) Diseño ejecutable. Remite al diseño de código a un nivel de detalle todavía más bajo y trata cuestiones tales como la asignación de memoria, los formatos de datos, etcétera.

En opinión de Clements [CBK+96] el diseño basado en arquitectura representa un paradigma de desarrollo que difiere de maneras fundamentales de las alternativas conocidas actualmente. En muchos sentidos, es diferente del diseño orientado a objetos (OOD) en la misma medida en que éste difería de sus predecesores. La AS deberá nutrir una comunidad de practicantes estableciendo una cultura en la que las ideas arquitectónicas puedan florecer. Una cuestión técnica que deberá abordarse es la creación de una articulación precisa de un paradigma de diseño basado en arquitectura (o posiblemente más de uno) y las cuestiones de proceso asociadas.

En una presentación de 1997, Dewayne Perry, uno de los fundadores de la disciplina, bosquejó la diferencia entre arquitectura y diseño. La arquitectura, una vez más (todo el mundo insiste en ello) concierne a un nivel de abstracción más elevado; se ocupa de componentes y no de procedimientos; de las interacciones entre esos componentes y no de las interfaces; de las restricciones a ejercer sobre los componentes y las interacciones y no de los algoritmos, los procedimientos y los tipos. En cuanto a la composición, la de la arquitectura es de grano grueso, la del diseño es de fina composición procedural; las interacciones entre componentes en arquitectura tienen que ver con un protocolo de alto nivel (en el sentido no técnico de la palabra), mientras que las del diseño conciernen a interacciones de tipo procedural (rpc, mensajes, llamadas a rutinas) [Per97].

En los primeros años del nuevo siglo, la AS precisó la naturaleza del proceso de diseño como metodología en diversos modelos de diseño basados en arquitectura o ABD

[BBC+00]. Esta metodología considera que el diseño arquitectónico es el de más elevado nivel de abstracción, pero debe hacer frente al hecho de un requerimiento todavía difuso y al hecho de que, en ese plano, las decisiones que se tomen serán las más críticas y las más difíciles de modificar. Fundándose en el concepto de arquitectura conceptual de Hofmeister, Nord y Soni [HNS00] y en un modelos de vistas similar al 4+1 o a las vistas del modelo arquitectónico de Microsoft, el ABD describe el sistema en función de los principales elementos y las relaciones entre ellos. El proceso se basa en tres fundamentos: (1) la descomposición de la función (usando técnicas bien establecidas de acoplamiento y cohesión), (2) la realización de los requerimientos de calidad y negocios a través de los estilos arquitectónicos, y (3) las *plantillas de software*, un concepto nuevo que incluye patrones que describen la forma en que todos los elementos de un tipo interactúan con los servicios compartidos y la infraestructura. El modelo de diseño específico de ABD se describe en los documentos correspondientes a metodologías arquitectónicas.

Seguramente el lector encontrará mucho que agregar al planteamiento de las similitudes y diferencias entre arquitectura y diseño, más allá del hecho obvio de que el diseño arquitectónico se distingue del diseño del código, que viene determinado desde mucho antes y que es mucho más crítico para el éxito o el fracaso de una solución.

Repositorios

Existen unos cuantos repositorios de información arquitectónica, cuyas direcciones son más o menos permanentes. El más importante hoy en día parece ser el del Software Engineering Institute en la Universidad Carnegie Mellon de Pittsburgh, Pennsylvania (http://www.sei.cmu.edu/ata/ata_init.html). El sitio del SEI incluye abundante literatura académica y todas las especificaciones o recomendaciones metodológicas.

Entre los organismos que definen estándares, son esenciales los servicios de información de RM-ODP en http://www.dstc.edu.au/Research/Projects/ODP/ref_model.html, TOGAF en http://www.software.org/pub/architecture/togaf.asp, DoDAF (hogar del C4ISR) en http://www.software.org/pub/architecture/dodaf.asp. El estándar IEEE 1471-2000 se puede encontrar en http://www.techstreet.com/cgi-bin/detail?product_id=879737. El marco de referencia empresarial de Zachman se puede acceder desde http://www.software.org/pub/architecture/zachman.asp.

Las páginas de cabecera de la estrategia de arquitectura de Microsoft se hallan en http://msdn.microsoft.com/architecture/. Las páginas de Patterns & Practices están en http://www.microsoft.com/resources/practices/completelist.asp. La estrategia arquitectónica misma se encuentra delineada en un documento de Michael Platt en http://msdn.microsoft.com/architecture/overview/default.aspx?pull=/library/en-us/dnea/html/eaarchover.asp.

Numerosas editoriales, universidades, consorcios y centros de investigación incluyen vínculos ligados a la AS. Muy seguramente los lectores podrán suministrar otros sitios de referencia importantes.

Problemas abiertos en Arquitectura de Software

Aunque la evaluación dominante de la trayectoria de la AS es abiertamente positiva, en los primeros años del siglo comienzan a percibirse desacuerdos y frustraciones en el proyecto de la disciplina. Por empezar, está muy claro que la definición original del proyecto, formulada en círculos académicos, guarda poca relación con la imagen que se tiene de la AS en las prácticas de la industria. Aún dentro de los confines de aquel círculo, a menudo las presentaciones distan de ser contribuciones desinteresadas y representan más bien posturas teóricas particulares que se quieren imponer por vía de la argumentación. Muchas de esas posturas son más programáticas que instrumentales. Un número desmesurado de artículos y ponencias destaca asimismo la frustración que muchos sienten por no haber podido consensuar una definición de la AS universalmente aceptada.

Vale la pena revisar el inventario de aspectos negativos elaborada por Clements y Northrop [CN96]:

> **Los abogados de la distintas posturas traen sus sesgos consigo.** En efecto, mientras las definiciones propuestas para la AS coinciden en su núcleo, difieren seriamente en los bordes. Algunos autores requieren que la arquitectura incluya racionalización, otros piden más bien pasos para el proceso de construcción. Algunos exigen que se identifique la funcionalidad de los componentes, otros alegan que una simple topología es suficiente. En la lectura de los textos de AS se torna esencial entonces conocer la motivación de sus responsables.
>
> **El estudio de la AS está siguiendo a la práctica, no liderándola.** El estudio de la AS ha evolucionado de la observación de los principios de diseño y las acciones que toman los diseñadores cuando trabajan en sistemas de la vida real. La AS es un intento de abstraer los rasgos comunes inherentes al diseño de sistemas, y como tal debe dar cuenta de un amplio rango de actividades, conceptos, métodos, estrategias y resultados. Lo que en realidad sucede es que la AS se redefine constantemente en función de lo que hacen los programadores.
>
> **El estudio es sumamente nuevo.** Aunque sus raíces se prolongan en el tiempo, el campo de la AS es realmente muy nuevo, según puede juzgarse de la reciente avalancha de libros, conferencias, talleres y literatura consagrado a ella.
>
> **Las fundamentaciones han sido imprecisas.** El campo se ha destacado por la proliferación de términos inciertos, verdaderas amenazas para los no precavidos. Por ejemplo, la arquitectura definida como “la estructura global de un sistema” agrega confusión en lugar de reducirla, porque implica que un sistema posee una sola estructura.
>
> **El término se ha usado en exceso.** El significado de la palabra “arquitectura” en su relación con la ingeniería de software se ha ido diluyendo simplemente porque parece estar de moda. Es posible encontrar referencias a las siguientes “clases” de arquitectura: específica de dominio, megaprogramación, destinatario, sistemas, redes, infraestructura, aplicaciones, operaciones, técnica, framework, conceptual, de referencia, empresarial, de factoría, C4I, de manufactura, de edificio, de

> máquina-herramienta, etcétera. A menudo la palabra "arquitectura" se usa de maneras inapropiadas [CN96].

Hacia el año 2000, Taylor y Medvidovic celebraban el auge de la AS, pero señalaban que por momentos daba la impresión de haberse convertido en una moda que prometía más de lo que se podía realizar. Les preocupaba también que la AS terminara confundiéndose con los ADLs y con el diseño. Si bien el foco en la arquitectura posee un tremendo potencial, argumentan, está comenzando a percibirse una cierta reacción. Las razones son numerosas. La arquitectura es aún en gran medida una noción académica, y muy poca de la tecnología resultante ha sido transferida a la industria y cuando se lo ha hecho no se lo ha comunicado bien. La concentración en la investigación como un valor en sí mismo ha sido también mal comprendida. Mientras tales errores ganen terreno, surge el peligro de que los beneficios concretos y los resultados positivos de la investigación arquitectónica sufran las consecuencias [TMA+S/f].

Jason Baragry ha cuestionado la metáfora de la arquitectura de edificios como columna vertebral de la disciplina. Ello ha ocasionado, entre otras cosas, la falta de acuerdo en cuanto a la cantidad y calidad de vistas arquitectónicas, así como la ostensible carencia de un mecanismo de coordinación formal entre las diferentes vistas [BG01], un problema técnico que también surge, incidentalmente, en la integración de las vistas y diagramas de UML y en las recomendaciones de los organismos.

Un problema que habría que tratar sistemáticamente es la discrepancia en las definiciones de AS que prevalecen en la academia y las que son comunes en la industria, incluyendo en ella tanto a los proveedores de software de base como a los equipos de desarrollos de las empresas. Jan Bosch [Bos00] ha confeccionado una tabla de constrastes que seguramente podría enriquecerse con nuevas antinomias.

Academia	Industria
La arquitectura se define explícitamente	Prevalece una comprensión conceptual de la arquitectura. Las definiciones explícitas son mínimas, eventualmente mediante notaciones
La arquitectura consiste en componentes y conectores de primera clase	No hay conectores explícitos de primera clase (a veces hay soluciones *ad hoc* de *binding* en tiempo de ejecución)
Los lenguajes de descripción de arquitectura (ADLs) describen la arquitectura explícitamente y a veces la generan	Se utilizan lenguajes de programación
Los componentes reutilizables son entidades de caja negra	Los componentes son grandes piezas de software de estructuras interna compleja, no necesariamente encapsulados
Los componentes tienen interfaces con un solo punto de acceso	Las interfaces se proporcionan mediante entidades (clases en los componentes). Las entidades de interfaz no tienen diferencias explícitas de entidades que no son de interfaz
Se otorga prioridad a la funcionalidad y la verificación formal	La funcionalidad y los atributos de calidad (performance, robustez, tamaño, reusabilidad, mantenibilidad) tienen igual importancia

Tampoco conviene sobredimensionar el contraste, imaginando que la academia está desconectada de la realidad industrial; por el contrario, la mayor parte de las herramientas académicas se elaboraron en el contexto de proyectos industriales y de gobierno, casi

todos ellos de misión crítica y de gran envergadura. El número de problemas de consistencia detectados a tiempo (y que se hubiera manifestado sin guía arquitectónica) es colosal. La cuestión radica más bien en el desconocimiento que la industria tiene de la arquitectura académica y en su obstinación por llamar arquitectura a lo que sólo es diseño.

Todavía se está por elaborar una apreciación honesta y ordenada de las dificultades enfrentadas por la disciplina. Algunas de ellas son menos del orden de la calidad conceptual que fruto de los rigores del mercado: por ejemplo, la escasa penetración de los ADLs a despecho de las limitaciones expresivas de los lenguajes de modelado que compiten con ellos, o la escasa atención que la industria concede prestarle a los métodos verdaderamente formales por poco que su utilización exija el dominio de una notación complicada. Pero a pesar que los ADLs o los métodos formales no han logrado abrirse camino, la AS en su conjunto sí lo ha hecho, aunque la imagen que el público tenga de ella pueda en algún sentido sospecharse espuria, o estar contaminada por ideas difusas referidas a metodologías o herramientas no necesariamente arquitectónicas. Con toda seguridad, hay muchos elementos para discutir sobre este punto.

Relevancia de la Arquitectura de Software

Aunque todavía no se ha constituido un repositorio uniformizado de estudios de casos en base al cual se pueda extraer una conclusión responsable, la AS ha resultado instrumental en un número respetable de escenarios reduciendo costos, evitando errores, encontrando fallas, implementando sistemas de misión crítica. Cada uno de los documentos que describen lenguajes de descripción arquitectónica, por ejemplo, subraya su utilización exitosa en proyectos de gran envergadura requeridos por organizaciones de gobierno o por grandes empresas. Aún cuando aquí y allá se señalen dificultades ocasionales, nadie duda de la necesidad de una visión arquitectónica. Escribe Barry Boehm, especialista máximo en gestión de riesgo y bien conocido creador del COCOMO y del método de desarrollo en espiral:

> Si un proyecto no ha logrado una arquitectura del sistema, incluyendo su justificación, el proyecto no debe empezar el desarrollo en gran escala. Si se especifica la arquitectura como un elemento a entregar, se la puede usar a lo largo de los procesos de desarrollo y mantenimiento [Boe95].

Antes que transcribir listas de ideas que a veces coinciden y otras señalan características heterogéneas, sintetizaremos las virtudes de la AS articulando las opiniones convergentes de los expertos [CN96] [Gar00]:

1. Comunicación mutua. La AS representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear entendimiento mutuo, formar consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales.
2. Decisiones tempranas de diseño. La AS representa la encarnación de las decisiones de diseño más tempranas sobre un sistema, y esos vínculos tempranos tienen un peso fuera de toda proporción en su gravedad individual con respecto al desarrollo restante

del sistema, su servicio en el despliegue y su vida de mantenimiento. La arquitectura representa lo que el método SAAM una puerta de peaje: el desarrollo no puede proseguir hasta que los participantes involucrados aprueben su diseño.

3. Restricciones constructivas. Una descripción arquitectónica proporciona *blueprints* parciales para el desarrollo, indicando los componentes y las dependencias entre ellos. Por ejemplo, una vista en capas de un arquitectura documenta típicamente los límites de abstracción entre las partes, identificando las principales interfaces y estableciendo las formas en que unas partes pueden interactuar con otras.
4. Reutilización, o abstracción transferible de un sistema. La AS encarna un modelo relativamente pequeño, intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí; este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de frameworks en el que se pueden integrar componentes.
5. Evolución. La AS puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas "paredes" perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipado y reutilización.
6. Análisis. Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, incluyendo verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.
7. Administración. La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales.

Una de las demostraciones más elocuentes de la capacidad de la AS como herramienta de decisión de diseño y evaluación de estilos es el *tour de force* de Mary Shaw y David Garlan [SG96] en el que comparan una misma solución de indexación de palabras claves en cuatro estilos diferentes (datos compartidos, tubería y filtro, tipos abstractos de datos e invocación implícita). El estudio articula un modelo que permite estimar relaciones de dependencia, modularidad, refinamiento, reutilización, ventajas y desventajas de cada arquitectura antes de escribir una sola línea de código; demuestra también de qué manera los diferentes estilos definen tácticas específicas de descomposición funcional y establecen la pauta que habrá de seguirse en el desarrollo. Finalmente, Shaw y Garlan aplican diversas tablas de comparación de atributos, en un ejercicio de evaluación de decisiones estilísticas que se ha convertido en un modelo en su género [Pfl02: 279-283]. Si hubiera que singularizar un trabajo simple y elegante, representativo del estado de arte

de la teoría y la práctica de la AS, sin duda escogeríamos esta demostración cabal, a treinta años de distancia, de que Dijkstra tenía razón.

La AS se encuentra, reconocidamente, en una etapa aún formativa. Sus teóricos no se encuentran todavía en condiciones de asegurar (como lo hacían los Tres Amigos) que "con este libro como guía, usted podrá producir, dentro de un plan predecible y un presupuesto más razonable, el software de la más alta calidad posible": una clase de alegación que llevó a Aron Trauring a lamentar que no hubiese algo así como una FDA que fiscalice los mensajes publicitarios de los metodólogos. Por el contrario, los mejores entre los arquitectos considera que su disciplina es tentativa y que se encuentra en estado de flujo. Pero aunque lo que resta por hacer es formidable, con lo que se lleva hecho ya hay un enorme repertorio de ideas, experiencias e instrumentos que ayudan a pensar de qué manera, aquí y ahora, se pueden mejorar las prácticas.

Referencias bibliográficas

[Abd00] Aynur Abdurazik. "Suitability of the UML as an Architecture Description Language with applications to testing". Reporte ISE-TR-00-01, George Mason University. Febrero de 2000.

[Alb03] Stephen Albin. *The Art of Software Architecture: Design methods and techniques*. Nueva York, Wiley, 2003.

[Ale77] Christopher Alexander. *A pattern language*. Oxford University Press, 1977.

[ASR+02] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen y Juhani Warsta. "Agile Software Development Methods". *VTT Publications* 478, Universidad de Oulu, Suecia, 2002.

[AW99] David Akehurst y Gill Waters. "UML deficiencies from the perspective of automatic performance model generation". *OOSPLA'99 Workshop on Rigorous Modeling and Analysis with the UML: Challenges and Limitations*, Denver, http://www.cs.kent.ac.uk/pubs/1999/899/content.pdf, Noviembre de 1999.

[BBC+00] Felix Bachmann, Len Bass, Gary Chastek, Patrick Donohoe, Fabio Peruzzi. "The Architecture Based Design Method". *Technical Report*, CMU/SEI-2000-TR-001, ESC-TR-2000-001, Enero de 2000.

[BCK98] Len Bass, Paul Clements y Rick Kazman. *Software Architecture in Practice.* Reading, Addison-Wesley, 1998.

[Bez98] Konstantin Besnozov. "Architecture of information enterprises: Problems and perspectives". http://www.beznosov.net/konstantin/doc/cis6612-paper.pdf, Abril de 1998.

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad y Michael Stal. *Pattern-oriented software architecture – A system of patterns*. John Wiley & Sons, 1996.

[Boe95] Barry Boehm. "Engineering Context (for Software Architecture)." Invited talk, *First International Workshop on Architecture for Software Systems*. Seattle, Abril de 1995.

[Boo91] Grady Booch. *Object-Oriented Design with Applications.* The Benjamin/Cummings Publishing Company, Inc, 1991.

[Bos00] Jan Bosch. *Design and use of Software Architecture*. Addison-Wesley, 2000.

[BR01] Jason Baragry y Karl Reed. "Why We Need a Different View of Software Architecture". *The Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Amsterdam, 2001.

[BRJ99] Grady Booch, James Rumbaugh e Ivar Jacobson. *El Lenguaje Unificado de Modelado*. Madrid, Addison-Wesley, 1999.

[Bro75] Frederick Brooks Jr. *The mythical man-month*. Reading, Addison-Wesley, 1975.

[Cib98] C. U. Ciborra. "Deconstructing the concept of strategic alignment". *Scandinavian Journal of Information Systems*, vol. 9, pp. 67-82, 1998.

[CLC03] David Cohen, Mikael Lindvall y Patricia Costa. "Agile Software Development. A DACS State-of-the-Art Report", *DACS Report*, The University of Maryland, College Park, 2003.

[Cle96a] Paul Clements. "A Survey of Architecture Description Languages". *Proceedings of the International Workshop on Software Specification and Design*, Alemania, 1996.

[Cle96b] Paul Clements. "Coming attractions in Software Architecture". *Technical Report*, CMU/SEI-96-TR-008, ESC-TR-96-008, Enero de 1996.

[CBK+95] Paul Clements, Len Bass, F. Kazman y Gregory Abowd. "Predicting Software Quality by Architecture-Level Evaluation," 485- 498. *Proceedings, Fifth International Conference on Software Quality*. Austin, 23 al 26 de Octubre de 1995, pp. 485-498. Austin, American Society for Quality Control, Software Division, 1995.

[CN96] Paul Clements y Linda Northrop. "Software architecture: An executive overview". *Technical Report*, CMU/SEI-96-TR-003, ESC-TR-96-003. Febrero de 1996.

[DD94] Peter Denning y Pamela Dargan. "A discipline of software architecture". *ACM Interactions*, 1(1), pp. 55-65, Enero de 1994.

[DDT99] Serge Demeyer, Stéphane Ducasse y Sander Tichelaar. "Why FAMIX and not UML?: UML Shortcomings for coping with round-trip engineering". *UML'99 Conference Proceedings*, LNCS Series, Springer Verlag, 1999.

[Dij68a] Edsger Dijkstra. "The Structure of the THE Multiprogramming system." *Communications of the ACM,* 26(1), pp. 49-52, Enero de 1983.

[Dij68b] Edsger Dijkstra. "GO-TO statement considered harmful". *ACM Communications of the ACM*, 11(3), pp. 147-148, Marzo de 1968.

[DIR99] N. Dunlop, J. Indulska y K. A. Raymond. "CORBA and RM-ODP: Parallel or divergent?", *Distributed Systems Engineering*, 6, pp. 82-91.

[DK76] Frank DeRemer y Hans Kron, "Programming-in-the-large versus programming-in-the-small". *IEEE Transaction in Software Engineering*, 2, pp. 80-86, 1976.

[Dou00] Bruce Powel Douglas. "UML – The new language for real-timed embedded systems". http://wooddes.intranet.gr/papers/Douglass.pdf, 2000.

[Fie00] Roy Thomas Fielding. "Architectural styles and the design of network-based software architectures". Tesis doctoral, University of California, Irvine, 2000.

[Fow01] Martin Fowler. "Is design dead?". *XP2000 Proceedings*, http://www.martinfowler.com/articles/designDead.html, 2001.

[GAD+02] Yann-Gaël Guéhéneuc, Hervé Albin-Amiot, Rémi Douence y Pierre Cointe. "Bridging the gap between modeling and programming languages". Reporte técnico, Object Technology International, Julio de 2002.

[Gar95] David Garlan. "Research Directions in Software Architecture. " ACM Computing Surveys 27, 2, pp. 257-261, Junio de 1995.

[Gar00] David Garlan. "Software Architecture: A Roadmap". En Anthony Finkelstein (ed.), The future of software engineering, ACM Press, 2000.

[Gars/f] David Garlan. "Software architectures". Presentación en transparencias, http://www.sti.uniurb.it/events/sfm03sa/slides/garlan-B.ppt.

[Gli00] Martin Glinz. "Problems and deficiencies of UML as a requirements specification language". En *Proceedings of the 10th International Workshop of Software Specification and Design* (IWSSD-10), San Diego, noviembre de 2000, pp. 11-22.

[GoF95] Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. *Design Patterns: Elements of reusable object-oriented software*. Reading, Addison-Wesley, 1995.

[GS94] David Garlan y Mary Shaw. "An introduction to software architecture". *CMU Software Engineering Institute Technical Report*, CMU/SEI-94-TR-21, ESC-TR-94-21, 1994.

[HHR+96] Pat Hall, Fiona Hovenden, Janet Rachel y Hugh Robinson. "Postmodern Software Development". *MCS Technical Reports*, 1996.

[Hig01] Jim Highsmith. "The great methodologies debate. Part I". *Cutter IT Journal*, 14(12), Diciembre de 2001.

[HNS99] Christine Hofmeister, Robert Nord y Dilip Soni. "Describing Software Architecture with UML". En *Proceedings of the First Working IFIP Conference on Software Architecture*, IEEE Computer Society Press, pp. 145-160, San Antonio, Febrero de 1999.

[HNS00] Christine Hofmeister, Robert Nord y Dilip Soni. *Applied software architecture*, Reading, Addison Wesley, 2000.

[Hock00] Dee Hock. *Birth of the chaordic age*. San Francisco, Berrett-Koehler.

[KAB+96] Rick Kazman, Gregory Abowd, Len Bass y Paul Clements. "Scenario-Based Analysis of Software Architecture". *IEEE Software*, 13(6), pp. 47-55, Noviembre de 1996.

[Keen91] P. G. W. Keen. "Relevance anr rigor in information systems research", en H.-E. Nissen, H. K. Klein y R. Hisrchheim (eds), *Information Systems Research: Contemporary approaches and emergent traditions*, Elsevier, 1991.

[KNK03] Rick Kazman, Robert Nord, Mark Klein. "A life-cycle view of architecture analysis and design methods". *Technical Report*, CMU/SEI-2002-TN026, Setiembre de 2003.

[Kros/f] John Krogstie. "UML, a good basis for the development of models of high quality?". Andersen Consulting Noruega & IDI, NTNU, http://dataforeningen.no/ostlandet/metoder/krogstie.pdf, sin fecha.

[Kru95] Philippe Kruchten. "The 4+1 View Model of Architecture." *IEEE Software* 12(6), pp. 42-50, Noviembre de 1995.

[Lar03] Craig Larman. *UML y Patrones*. 2ª edición, Madrid, Prentice Hall.

[MB02] Ruth Malan y Dana Bredemeyer. "Less is more with Minimalist Architecture". *IEEE IT Professional*, Setiembre-Octubre de 2002.

[McC96] Steve McConnell. "Missing in Action: Information hiding". *IEEE Software*, 13(2), Marzo de 1996.

[MKM+96] Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. "Stylized architecture, design patterns, and objects". http://citeseer.nj.nec.com/monroe96stylized.html.

[MKM+97] Robert Monroe, Andrew Kompanek, Ralph Melton y David Garlan. "Architectural Styles, design patterns, and objects". *IEEE Software*, pp. 43-52, Enero de 1997.

[MS03] Geoffrey Lory, Derick Campbell, Allison Robin, Gaile Simmons y Patricia Rytkonen. "Microsoft Solutions Framework Version 3.0 overview". http://www.microsoft.com/technet/itsolutions/techguide/msf/msfovrvw.mspx, Junio de 2003.

[NATO76] I. P. Sharp. Comentario en discusión sobre teoría y práctica de la ingeniería de software en conference de NATO Science Committee, Roma, 27 al 31 de Octubre de 1969. *Software Engineering Concepts and Techniques: Proceedings of the NATO conferences*. J. N. Bruxton y B. Randall (eds.), Petrochelli/Charter, 1976.

[Par72] David Parnas. "On the Criteria for Decomposing Systems into Modules." *Communications of the ACM* 15(12), pp. 1053-1058, Diciembre de 1972.

[Par74] David Parnas. "On a 'Buzzword': Hierarchical Structure", *Programming Methodology*, pp. 335-342. Berlin, Springer-Verlag, 1978.

[Par76] David Parnas. "On the Design and Development of Program Families." *IEEE Transactions on Software Engineering* SE-2, 1, pp. 1-9, Marzo de 1976.

[Per97] Dewayne Perry. "Software Architecture and its relevance for Software Engineering". *Coord'97*, 1997.

[Pfl02] Shari Lawrence Pfleeger. *Ingeniería de Software: Teoría y Práctica.* Madrid, Prentice-Hall.

[Platt02] Michael Platt. "Microsoft Architecture Overview: Executive summary", http://msdn.microsoft.com/architecture/default.aspx?pull=/library/en-us/dnea/html/eaarchover.asp, 2002.

[Pre02] Roger Pressman. *Ingeniería del Software: Un enfoque práctico*. Madrid, McGraw Hill, 2001.

[PW92] Dewayne E. Perry y Alexander L. Wolf. "Foundations for the study of software architecture". *ACM SIGSOFT Software Engineering Notes*, 17(4), pp. 40–52, Octubre de 1992.

[RJB00] Jamers Rumbaugh, Ivar Jacobson, Grady Booch. *El lenguaje unificado de modelado. Manual de Referencia*. Madrid, Addison-Wesley, 2000.

[Ross77] Douglas Ross. "Structured analysus (SA): A language for communicating ideas". *IEEE Transactions on Software Engineering*, SE-3(1), enero de 1977.

[Sch00] Klaus-Dieter Schewe. "UML: A modern dinosaur? – A critical analysis of the Unified Modelling Language". En H. Kangassalo, H. Jaakkola y E. Kawaguchi (eds.), *Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Bases*, Saariselk, Finlandia, 2000.

[SG96] Mary Shaw y David Garlan. *Software Architecture: Perspectives on an emerging discipline*. Upper Saddle River, Prentice Hall, 1996.

[Shaw84] Mary Shaw. "Abstraction Techniques in Modern Programming Languages". *IEEE Software*, Octubre, pp. 10-26, 1984.

[Shaw89] Mary Shaw. "Large Scale Systems Require Higher- Level Abstraction". *Proceedings of Fifth International Workshop on Software Specification and Design*, IEEE Computer Society, pp. 143-146, 1989.

[Shaw96] Mary Shaw. "Some Patterns for Software Architecture," en *Pattern Languages of Program Design, Vol. 2*, J. Vlissides, J. Coplien, y N. Kerth (eds.), Reading, Addison-Wesley, pp. 255-269, 1996.

[Shaw01] Mary Shaw. "The coming-of-age of Software Architecture research". *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, 2001.

[Spo71] C. R. Spooner. "A Software Architecture for the 70's: Part I - The General Approach." *Software - Practice and Experience,* 1 (Enero-Marzo), pp. 5-37, 1971.

[StöS/f] Harald Störrle. "Turning UML subsystems into architectural units". Reporte, Ludwig-Maximilians-Universität München, http://www.pst.informatik.uni-muenchen.de/personen/stoerrle/Veroeffentlichungen/PosPaperICSEFormat.pdf, sin fecha.

[Tem01] Theodor Tempelmeier. "Comments on Design Patterns for Embedded and Real-Time Systems". En: A. Schürr (ed.): *OMER-2 ("Object-Oriented Modeling of Embedded Realtime systems") Workshop Proceedings*. Mayo 9-12, 2001, Herrsching, Alemania. Bericht Nr. 2001-03, Universität der Bundeswehr München, Fakultät für Informatik, Mayo de 2001.

[Tem99] Theodor Tempelmeier. "UML is great for Embedded Systems – Isn't it?" En: P. Hofmann, A. Schürr (eds.): *OMER ("Object-Oriented Modeling of Embedded Realtime systems") Workshop Proceedings*. Mayo 28-29, 1999, Herrsching (Ammersee), Alemania. Bericht Nr. 1999-01, Universität der Bundeswehr München, Fakultät für Informatik, Mayo de 1999.

[TMA+S/f] Richard Taylor, Nenad Medvidovic, Kennet Anderson, James Whitehead Jr, Jason Robbins, Kari Nies, Peyman Oreizy y Deborah Dubrow. "A component- and message-based architectural style for GUI software". Reporte para el proyecto F30602-94-C-0218, Advanced Research Projects Agency, sin fecha.

[Tra02] Aron Trauring. "Software methodologies: The battle of the Gurus". *Info-Tech White Papers*, 2002.

[Wir71] Niklaus Wirth. "Program development by stepwise refinement", C*ommunications of the ACM*, 14(4), pp. 221-227, Abril de 1971.

[WL97] Sharon White y Cuauhtémoc Lemus-Olalde. "The software architecture process". http://nas.cl.uh.edu/whites/webpapers.dir/ETCE97pap.pdf, 1997.

[WWI99] WWISA. "Philosophy". Worldwide Institute of Software Architects, http://www.wwisa.org, 1999.

[Zac87] John A. Zachman, "A Framework for Information Systems Architecture", *IBM Systems Journal*, 26(3), , 1987.

[ZHH+99] Guoqiang Zhong, Babak Hodjat, Tarek Helmy y Makoto Amamiya. "Software agent evolution in adaptive agent oriented software architecture". *IWPSE'99 Proceedings*, 1999.