Un Cálculo de Patrones Arquitectónicos

Alejandro Sánchez* asanchez@unsl.edu.ar
Luis Soares Barbosa† lsb@di.uminho.pt
Daniel Riesco* driesco@unsl.edu.ar

CONTEXTO

La presente línea de investigación se encuentra enmarcada en el proyecto "Ingeniería de Software, Conceptos, Métodos y Herramientas en un contexto de Evolución de la Ingeniería del Software" acreditado por incentivos, de la Universidad Nacional de San Luis. La línea se desarrolla a partir de la colaboración surgida por el proyecto LerNet del programa ALFA con la Universidad de Minho (Portugal).

RESUMEN

En esta línea de investigación proponemos desarrollar un framework semántico y un cálculo que permita describir, transformar y razonar sobre patrones arquitectónicos de software. El trabajo apunta a responder a las necesidades que emergen a partir del éxito de la orientación a servicios, en donde se necesita tratar con arquitecturas dinámicamente reconfigurables y auto-adaptables. Se basa en un enfoque desarrollado en la Universidad de Minho para el cálculo de componentes y se aplican avances del áreas de modelos y lenguajes de coordinación.

Palabras clave: Patrones Arquitectónicos de Software, Arquitectura de Software, Cálculo, Coalgebra

1 INTRODUCCION

La continua evolución hacia sistemas de computación extremadamente grandes, altamente dinámicos y heterogéneos requiere enfoques innovadores para dominar su complejidad. Los servicios no solo están transformando la Web, de estar centrada en documentos a ser una infraestructura viva, sino que también desafían nuestro entendimiento de como desarrollar aplicaciones, e incluso, de la naturaleza misma del software (visto como un servicio a ser contratado en lugar de un producto a adquirir). El impacto de este cambio, tanto en la economía mundial como en la vida cotidiana, esta tan solo comenzando a apreciarse.

Complejos sistemas de software se construyen conectando servicios que interactúan intercambiando datos, llevando a cabo computaciones, y modificando su ambiente. Los servicios son entidades dinámicas, que se ejecutan en diferentes plataformas, a menudo pertenecen a distintas organizaciones, interactúan a través de interfaces públicas, y típicamente permanecen mínimamente acoplados, si no completamente ignorantes unos de los otros.

El correcto diseño de tales sistemas es difícil por que su complejidad va más allá del alcance actual de los métodos formales. Dificultades adicionales emergen con servicios provistos por terceros, frecuentemente subespecificados o que no cumplen con sus especificaciones. Es más, siendo el cambio la norma en lugar de la excepción, la reconfiguración dinámica y la auto-adaptabilidad, es decir, la capacidad de un sistema de ajustarse a sí mismo durante su ejecución en respuesta a su percepción del contexto, se volverán un importante tema en el futuro cercano.

1.1 Antecedentes

El problema se puede enmarcar en dos áreas de investigación dentro de la Ingeniería del Software: *arquitecturas de software*, y *lenguajes y modelos de coordinación*.

La primera emerge como un disciplina [26, 19, 2, 18, 12] en la Ingeniería de Software, de la necesidad de considerar de manera explicita en el desarrollo de sistemas, crecientemente grandes y complejos, los efectos, problemas y oportunidades de la estructura, organización y comportamiento emergente de un sistema completo. En una definición amplia, la arquitectura de un sistema describe su organización fundamental, la cual ilumina las decisiones de diseño de alto nivel:

- ¿cómo esta compuesto y de que partes interactuantes?
- ¿cuales son las interacciones y los patrones de comunicación presentes?
- ¿cuales son las propiedades claves de partes en las que el sistema completo descansa y/o impone?

En su papel de modelo, la arquitectura actúa como una abstracción de un sistema y suprime detalles de elementos que no afectan como utilizan, son utilizados por, se relacionan o interactúan con otros elementos. Entonces, se enfoca en los elementos e interfaces estructurales por los cuales un sistema es compuesto, sus comportamientos separados y combinados especificados como colaboraciones entre dichos elementos, y finalmente la composición de estos elementos

*Universidad Nacional de San Luis, Facultad de Ciencias Físico-Matemáticas y Naturales, Departamento de Informática, Tel: +54 (0) 2652 424027 / Fax: +54 (0) 2652 430059, San Luis, Argentina

†Universidade do Minho, Escola de Engenharia, Departamento de Informática, Centro de Ciências e Tecnologias de Computação, Tel: +351 253604463 / Fax: +351 253604471, Braga, Portugal

estructurales y comportamentales en sistemas más grandes. De acuerdo con la norma ANSI/IEEE Std 1471-2000, la cual es parte de un procesos de estandarización en marcha, la arquitectura describe la organización fundamental del sistema, materializado en sus componentes, sus relaciones entre ellos y el ambiente, y los principios gobernando su diseño y evolución.

Por el otro lado, el paradigma de coordinación [21, 5], el cual promueve una estricta separación entre la computación efectiva y su control, apareció como una solución al problema de manejar la interacción entre actividades concurrentes en un sistema. Emergió de la necesidad de explotar el pleno potencial de masivos sistemas en paralelo, lo cual requiere modelos capaces de tratar, de una manera explicita, la concurrencia de la cooperación entre un gran número de componentes poco acoplados, autonomos y heterogéneos. Los modelos de coordinación [20, 25, 6] hacen una clara distinción entre estos componentes y sus interacciones, y se enfocan en el comportamiento *emergente* combinado de estos. Tradicionalmente, los lenguajes y modelos de coordinación han evolucionado alrededor de la noción de un espacio de datos compartido — una abstracción de memoria para el intercambio de datos, accesible a todos los procesos cooperando para alcanzar una meta. El primer lenguaje de coordinación que introduce tal noción fue Linda [1]; varios modelos relacionados evolucionaron más tarde sobre nociones similares [21, 13]. El modelo subyacente era *dirigido por datos*, en el sentido que los procesos podían examinar la naturaleza de los datos intercambiados y actuar de acuerdo a estos. Una familia alternativa de modelos, *dirigida por eventos* o *dirigida por control*, se ajusta mejor a sistemas cuyos componentes interactúan entre ellos pasándose y recibiendo eventos, la presencia de los cuales dispara actividad. Un modelo pionero en esta familia es MANIFOLD [8], el cual implementa el modelo IWIM [4]. Al contrario que con la familia dirigida por datos, en donde los coordinadores manejaban directamente los valores de datos, en estos modelos, los procesos se ven como cajas negras que se comunican con su ambiente a través de interfaces claramente definidas (frecuentemente llamadas puertos de entrada y salida). Durante los últimos 15 años, la aparición masiva de sistemas concurrentes y heterogéneos, y el crecimiento de la complejidad de los protocolos de interacción, ha llevado a la coordinación a un lugar central en el desarrollo de software. Este avance contribuyó a ampliar su alcance de aplicación y significó el desarrollo de un número de modelos, lenguajes y semánticas específicos.

Los modelos de coordinación y las descripciones arquitectónicas nacieron dentro de contextos, preocupaciones y dominios de aplicación diferentes. Sin embargo, su foco es similar y recientes tendencias en la industria de software enfatizan la relevancia de los principios básicos subyacentes. Recordar, por ejemplo, los desafíos vinculados al cambio desde el paradigma *programming-in-the-large* dos décadas atrás, al reciente paradigma de *programming-in-the-world*, donde no solo se tiene que dominar la complejidad de construir y desplegar grandes aplicaciones en tiempo y dentro del presupuesto, sino que también se tiene que administrar una estructura abierta de componentes autonomos, posiblemente distribuidos y altamente heterogéneos. O también considere el cambio de entender al *software como producto* a *software como servicio* [17], enfatizando su estructura abierta, dinámica, reconfigurable y evolutiva. Términos tales como *orquestación* y *coreografía*, y los intensos esfuerzos de investigación asociados (ver [15, 3, 14, 29], entre otros), acentúan la relevancia de los principales temas de investigación en arquitecturas y coordinación para la Ingeniería de Software. En cierto modo, una definición temprana de coordinación que enfatiza su meta de encontrar soluciones al problema de manejar la interacción entre programas concurrentes [5], puede ser tomada como un desafio clave a este dominio de la Ingeniería.

Tanto la *arquitectura de software* como los *modelos de coordinación* tratan la interacción de componentes, abstrayendo los detalles de la computación y enfocándose en la naturaleza y forma de las interacciones. En consecuencia, la sincronización, comunicación, reconfiguración, inicio y fin de las computaciones son temas que les concierne.

También se debe remarcar que, a pesar del notable progreso en la representación y uso de las arquitecturas de software, la especificación del diseño arquitectónico se mantiene mayormente informal en la actualidad. Típicamente, dichos diseños se apoyan en notaciones con semántica pobre, y frecuentemente limitada a expresar solo las propiedades estructurales mas básicas. Por el otro lado, los lenguajes y modelos de coordinación recientes — tales como REO [6, 7] y ORC [22, 24] — presentan un mayor grado de formalidad, lo cual acentúa el caso para una *vista dirigida por coordinación* de la arquitectura de sistemas.

1.2 Objetivos

Ha llegado el momento de profundizar en el desarrollo de una *teoría de patrones arquitectónicos*, comprendiendo una semántica y un cálculo, construidos en base a la experiencia ganada de la investigación en coordinación. Es más, esta teoría parece fundamental a los efectos de proveer bases sólidas para el diseño orientado a servicios. En este contexto, el trabajo apunta a tratar las siguientes preguntas transversales a los fundamentos de la arquitectura de software:

- ¿Cómo especificar, transformar y razonar sobre patrones arquitectónicos, y calcular el comportamiento emergente del sistema?
- ¿Cómo aplicar tal cálculo de patrones arquitectónicos para diseñar, analizar y transformar redes evolutivas de componentes dinámicamente reconfigurables y arquitecturas auto-adaptables?

1.3 Relevancia

Estos temas, y en particular, el foco en arquitecturas dinámicas y auto-adaptables, son relevantes a un amplio rango de sistemas. Este rango va desde comercio electrónico a sistemas móviles embebidos, operados con un mínimo de supervisión humana, en el contexto en el cual

la distinción entre 'desarrollo', 'despliegue' y 'mantenimiento' tiende a ser difusa. A pesar de ser una realidad tecnológica, la reconfiguración en tiempo de ejecución es difícil de modelar, analizar y predecir. A su vez, las menos comunes arquitecturas, capaces de monitorear y adaptarse a sí mismas a fallas (por ejemplo, perdida de conexiones o fallas en servicios), a recursos variables (como disponibilidad de ancho de banda) y a cambios no predecibles en el contexto, crecerán en relevancia en el futuro próximo.

Estos son grandes desafíos de investigación para los cuales la teoría de patrones arquitectónicos a desarrollarse en este trabajo puede proveer, aunque sea de manera parcial, respuestas relevantes. En realidad, al ser un área emergente, la computación orientada a servicios requiere de claras definiciones de modelos y métodos para la especificación, orquestación y desarrollo de servicios, así como una técnica especifica para su análisis y transformación.

2 LINEAS DE INVESTIGACION y DESARROLLO

Este trabajo apunta a desarrollar una nueva y rigurosa disciplina de patrones arquitectónicos. Esta consistirá de un cálculo para tales patrones y sus transformaciones. Irá mas allá de las conocidas nociones *ad hoc* de estilos arquitectónicos (tal como se presentan en las referencias clásicas del tema), y buscará formulaciones independientes de cualquier tecnología. Describimos el enfoque a seguir a través de los siguientes puntos:

- El cálculo adoptará un punto de vista exógeno para la coordinación y se basará en la existencia de un conjunto pequeño de combinadores genéricos. Tal como en los lenguajes de coordinación modernos, el punto de vista exógeno implica un desacoplamiento estricto entre servicios, para soportar un mínimo de dependencias entre componentes. En contraste con los enfoques menos estructurados de tales lenguajes, el conjunto de combinadores genéricos permitirán el diseño de conectores de una manera estructurada. En consecuencia, preveemos dos versiones diferentes, pero relacionadas, para este cálculo:
  - Sobre lenguajes de coordinación modernos (tales como Reo y Orc): se dirige el trabajo a una amplia comunidad de usuarios y testers que estos lenguajes tienen.
  - Sobre el llamado enfoque Minho para el cálculo de componentes: se aplica este enfoque puramente composicional equipado con las nociones de bisimulación y refinamiento [23]. Diferentes trabajos siguiendo este enfoque han mostrado su aplicación para la especificación de conectores [11], la componentización de una especificación [16], el cálculo de invariantes [9], y la especificación de componentes parciales (componentes que fallan o mueren, exhibiendo un comportamiento más efímero que lo esperado) y la extensión de dichos componentes con un mecanismo de ciclos *try-again* [10].
- Proponemos *dialgebras* [27] como un modelo básico para el cálculo de patrones arquitectónicos. Los servicios y sus configuraciones aparecen naturalmente equipadas con operaciones 'constructoras' y 'destructoras' (estas últimas también llamadas observadores), las cuales requieren nociones de equivalencia semántica y refinamiento tanto estructurales (algebraicas) como comportamentales (coalgebraicas). Las dialgebras generalizan álgebras y coalgebras, pero su estudio esta todavía en su infancia con respecto a los métodos algebraicos/coalgebraicos.
- El cálculo debe extenderse para tratar con propiedades arquitectónicas no funcionales (relacionadas a calidad de servicio).
- En el nivel experimental, este trabajo apunta a caracterizar y clasificar un numero de patrones arquitectónicos, estudiando su integración, uso y relevancia. En lugar de ser postulados, su identificación y clasificación procederá de la inspección de casos reales. Para esto, extenderemos herramientas previamente desarrolladas en la Universidad de Minho [28], con el objetivo de extraer especificaciones de coordinación de código ejecutable. También aprovecharemos un enorme repositorio de datos arquitectónicos mantenidos por una empresa Holandesa, especializada en análisis de código, la cual ya ha accedido en esta forma de colaboración.

3 RESULTADOS ESPERADOS/OBTENIDOS

Se espera que el trabajo en esta línea ofrezca resultados en el nivel teórico y de aplicación. En el nivel teórico se espera obtener un framework semántico y un cálculo de patrones arquitectónicos, aplicables a arquitecturas dinámicamente reconfigurables y auto-adaptables. En el nivel de aplicación se prevee extender herramientas para soportar la extracción de datos arquitectónicos del código ejecutable de sistemas reales.

4 FORMACION DE RECURSOS HUMANOS

La línea de investigación es llevada adelante por investigadores de la Universidad de Minho y la Universidad de San Luis. La misma será el contexto en el que se desarrollarán tesis de maestría y doctorado

BIBLIOGRAFIA

[1] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, 19(8):26–34, 1986.

[2] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM TOSEM*, 6(3):213–249, 1997.

[3] L. F. Andrade and J. L. Fiadeiro. Composition contracts for service interaction. *Journal of Universal Computer Science*, 10(4):751–761, 2004.

[4] F. Arbab. The iwim model for coordination of concurrent activities. In P. Ciancarini and C. Hankin, editors, *Proc. Coordination Languages and Models, First Inter. Conf., COORDINATION '96, Cesena, Italy, April 15-17*, volume 1061, pages 34–56. Springer Lect. Notes Comp. Sci. (1061), 1996.

[5] F. Arbab. What do you mean, coordination. In *Bulletin of the Dutch Association for Theoretical Computer Science (NVTI)*, 1998.

[6] F. Arbab. Abstract behaviour types: a foundation model for components and their composition. In F. S. de Boer, M. Bonsangue, S. Graf, and W.-P. de Roever, editors, *Proc. First International Symposium on Formal Methods for Components and Objects (FMCO'02)*, pages 33–70. Springer Lect. Notes Comp. Sci. (2852), 2003.

[7] F. Arbab. Reo: a channel–based coordination model for component composition. *Mathematical Structures in Comp. Sci.*, 14(3):329–366, 2004.

[8] F. Arbab, I. Herman, and P. Spilling. An overview of manifold and its implementation. *Concurrency - Practice and Experience*, 5(1):23–70, 1993.

[9] L. S. Barbosa and J. N. Oliveira. Transposing partial components: an exercise on coalgebraic refinement. *Theor. Comp. Sci.*, 365(1-2):2–22, 2006.

[10] L. S. Barbosa, J. N. Oliveira, and A. M. Silva. Calculating invariants as coreflexive bisimulations. In J. Meseguer and G. Rosu, editors, *Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008, Urbana, IL, USA, July 28-31, 2008, Proceedings*, pages 83–99. Springer Lect. Notes Comp. Sci. (5140), 2008.

[11] M. Barbosa and L. Barbosa. Specifying software connectors. In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 53–68. Springer Lect. Notes Comp. Sci. (3407), 2004.

[12] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice (2nd ed.)*. Addison-Wesley, 2003.

[13] R. Bjornson, N. Carriero, and D. Gelernter. From weaving threads to untangling the web: A view of coordination from linda's perspective. In D. Garlan and D. L. Metayer, editors, *Proc. of Second Inter. Conf. on Coordination Languages and Models, COORDINATION '97, Berlin, Germany*, pages 1–17. Springer Lect. Notes Comp. Sci. (1282), 1997.

[14] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web services choreographies. In *Proc. First Inter. Workshop on Web Services and Formal Methods*, volume 105, pages 73–94, Pisa, Italy, 2004.

[15] N. Busi, R. Gorrieri, C. Guidi, R. Luchi, and G. Zavattaro. Choreography and orchestration: A synergic approach for systems design. In B. Benatallah, F. Casati, and P. Traverso, editors, *Proc. ICSOC 2005 Thrid Inter. Conf. on Service-Oriented Computing*, pages 228–240, 2005.

[16] A. Cruz, L. Barbosa, and J. Oliveira. From algebras to objects: Generation and composition. *Journal of Universal Computer Science*, 11(10):1580–1612, 2005.

[17] J. L. Fiadeiro. Software services: scientific challenge or industrial hype? In K. Araki and Z. Liu, editors, *Proc. First International Colloquim on Theoretical Aspects of Computing (ICTAC'04), Guiyang, China*, pages 1–13. Springer Lect. Notes Comp. Sci. (3407), 2004.

[18] D. Garlan. Formal modeling and analysis of software architecture: Components, connectors and events. In M. Bernardo and P. Inverardi, editors, *Third International Summer School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures (SFM 2003)*. Springer Lect. Notes Comp. Sci, Tutorial, (2004), Bertinoro, Italy, September 2003.

[19] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering (volume I)*. World Scientific Publishing Co., 1993.

[20] D. Gelernter and N. Carrier. Coordination languages and their significance. *Communication of the ACM*, 2(35):97–107, February 1992.

[21] D. L. M. Jean-Marc Andreoli, Chris Hankin. *Coordination Programming: Mechanisms, Models, and Semantics*. Imperial College Press, 1996.

[22] D. Kitchin, W. R. Cook, and J. Misra. A language for task orchestration and its semantic properties. In C. Baier and H. Hermanns, editors, *Proc. 17th Inter. Conf. Concurrency Theory, CONCUR 2006, Bonn, Germany, August 27-30*, pages 477–491. Springer Lect. Notes Comp. Sci. (4137), 2006.

[23] S. Meng and L. S. Barbosa. Components as coalgebras: The refinement dimension. *Theor. Comp. Sci.*, 351:276–294, 2005.

[24] J. Misra and W. R. Cook. Computation orchestration: A basis for wide-area computing. *Software and System Modeling*, 6(1):83–110, 2007.

[25] G. Papadopoulos and F. Arbab. Coordination models and languages. In *Advances in Computers — The Engineering of Large Systems*, volume 46, pages 329–400. 1998.

[26] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[27] E. Poll and J. Zwanenburg. From algebras and coalgebras to dialgebras. In H. Reichel, editor, *Coalgebraic Methods in Computer Science (CMCS'2001)*, number 44 in ENTCS. Elsevier, 2001.

[28] N. F. Rodrigues and L. S. Barbosa. Coordinspector: a tool for extracting coordination data from legacy code. In *SCAM '08: Proc. of the Eighth IEEE Inter. Working Conference on Source Code Analysis and Manipulation*. IEEE Computer Society, 2008. to appear.

[29] Q. Zongyan, Z. Xiangpeng, C. Chao, and Y. Hongli. Towards the theoretical foundation of choreography. In P. Patel-Schneider and P. Shenoy, editors, *Proceedings of the 16th Int Conf. on World Wide Web*, pages 973–982. ACM, 2007.