José Martín Villalba
vilcom.martin@gmail.com

Universidad Nacional del Centro
Facultad de Ciencias Exactas
Ingeniería de Sistemas
Febrero 2013

Propuesta de Proyecto Final de la carrera Ingeniería de Sistemas

"Un Sistema Experto para Asistir en la Sincronización de Documentación Arquitectónica con su Materialización"

Dirección: Dr. Álvaro Soria
Alumno: José Martín Villalba

1. Motivación

El desarrollo de software centrado en la arquitectura está siendo ampliamente aceptado como una práctica de ingeniería de software. En este tipo de proceso es clave la conformidad entre la documentación de la arquitectura y su correspondiente implementación. Para lograr esta conformidad, los desarrolladores inicialmente refinan el diseño de la arquitectura hasta obtener una implementación concreta. Durante este refinamiento, se especifican las relaciones entre el modelo arquitectónico *cómo fue documentado* y el modelo arquitectónico *cómo fue implementado* [1, 5]. Estas relaciones, denominadas mapeos, mantienen un puente entre la documentación arquitectónica y su implementación [1,2]. Dicho mapeo es fundamental para permitir que la consistencia sea mantenida durante la evolución del sistema. Cuando los desarrolladores finalizan la implementación de la arquitectura, el modelo arquitectónico *cómo fue documentado* suele ser consistente con el modelo arquitectónico *cómo fue implementado*.

Sin embargo, la evolución natural del sistema habitualmente altera esta conformidad generando diferencias entre la documentación y su implementación. Esto se debe, en parte, a que una vez que la arquitectura es finalizada, todos los esfuerzos se centran en la implementación, causando una paulatina desactualización de la documentación de diseño. Por ejemplo, las tareas de reparación de fallas o la implementación de nuevos requerimientos - entre otras tareas - pueden causar cambios en el código que debieran ser reflejados en la arquitectura. En estos casos, las relaciones de componentes y responsabilidades (en la documentación del nivel arquitectónico) con clases y métodos (en la implementación real) no siempre se mantienen, y los desarrolladores deben restaurar manualmente la consistencia. Este fenómeno es conocido como corrimiento arquitectura-implementación [4] o erosión arquitectónica, y si no es manejado apropiadamente, puede afectar negativamente a los beneficios del desarrollo centrado en la arquitectura.

Con el objetivo de enfrentar esta problemática, la primera solución es utilizar un enfoque manual basado en code reviews [11, 12]. Estas revisiones requieren tener a disposición algún tipo de mapeo o asociación entre elementos arquitectónicos (componentes, módulos, responsabilidades, etc.) y elementos de implementación (paquetes, clases, métodos, etc.). Basándose en estos mapeos, la persona encargada de realizar la revisión analiza el código fuente en busca de diferencias arquitectónicas, ya sea manualmente o mediante la utilización de una herramienta de soporte (como por ejemplo *ConQAT* [13], *SAVE* [17] o Sonargraph-Architect [3]). Estas diferencias pueden involucrar aspectos estructurales o aspectos comportamentales del diseño. Asimismo, a la hora de realizar el análisis se debe tener en cuenta que los mapeos también se pueden erosionar con el tiempo [8]. Una vez detectada una diferencia

arquitectónica, el Arquitecto debe determinar las acciones a llevar a cabo para restablecer la consistencia.

Si bien la solución mencionada es válida, realizar este tipo de análisis sobre aplicaciones de gran tamaño puede consumir mucho tiempo y es una actividad propensa a errores, ya que puede haber inconsistencias que no son detectadas. Por este motivo, es que se considera vital la utilización de herramientas que permitan mantener la correspondencia entre la documentación arquitectónica y su implementación. La conservación de este tipo de correspondencia ha sido un tópico activo de investigación, con enfoques que a menudo emplean visualización [15], ingeniería reversa [6, 17], lenguajes de descripción arquitectónica [16, 18, 19] y técnicas de chequeo de consistencia [3, 13, 14, 17]. En algunos casos se emplean enfoques basados en ingeniería reversa para generar una versión actual de la documentación arquitectónica y luego se compara la versión generada con la versión original para chequear la consistencia entre la documentación arquitectónica y el código fuente asociado.

En muchos casos se proveen herramientas que brindan asistencia en la identificación y/o re-sincronización de diferencias estructurales [3, 6, 13, 14, 17]. Sin embargo es difícil encontrar herramientas que brinden soporte para mantener consistente la documentación arquitectónica comportamental.

En este trabajo presentamos un enfoque que ofrece un mecanismo semi-automático para brindar asistencia en la sincronización de la documentación arquitectónica tanto estructural cómo comportamental. Idealmente, los chequeos de consistencia entre arquitectura e implementación consisten de dos pasos. El primer paso es buscar diferencias estructurales con respecto a las vistas arquitectónicas. El segundo consiste en buscar diferencias comportamentales, es decir, buscar interacciones del sistema que se desvían de los escenarios de comportamientos definidos en la arquitectura.

2. Propuesta

El objetivo principal del trabajo es ofrecer una herramienta semi-automática, denominada *ASAP*, que cubra completamente el proceso de sincronización arquitectónica. De este modo, se pretende lograr que la tarea de sincronización de documentación arquitectónica sea una actividad con menor esfuerzo para los Arquitectos. El enfoque propuesto busca permitir la generación de documentación arquitectónica a partir del código fuente, y ofrecer la posibilidad de mantener esa documentación sincronizada a medida que el mismo evoluciona.

Con el fin de asistir en la sincronización de la documentación estructural de la arquitectura se propone un enfoque basado en reglas. La utilización de reglas permite incorporar conocimiento del Arquitecto en su definición. Adicionalmente, las reglas definidas pueden ser re-utilizadas en diferentes proyectos que empleen los mismos conceptos arquitectónicos o convenciones de desarrollo. Por otro lado, el enfoque plantea la utilización de logs de ejecución para la generación y sincronización de documentación comportamental. Estos logs permiten obtener información del sistema bajo análisis durante su ejecución para ser comparada con la especificación comportamental de su arquitectura. La Figura 4.1 muestra un esquema conceptual del enfoque propuesto.

José Martín Villalba
vilcom.martin@gmail.com

Universidad Nacional del Centro
Facultad de Ciencias Exactas
Ingeniería de Sistemas
Febrero 2013

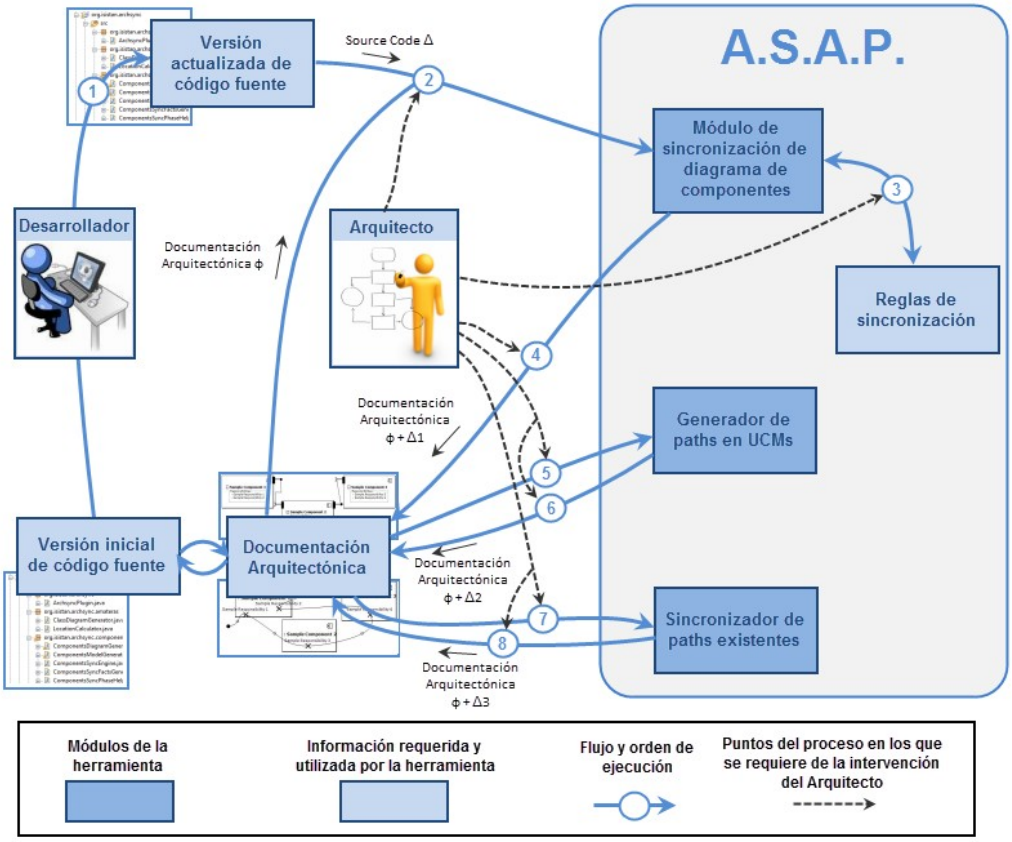

Figura : Mapa conceptual del enfoque ASAP

Cabe destacar que, para hacer uso del enfoque, una vez que la arquitectura del sistema se define, el Arquitecto deberá crear reglas de sincronización. Éstas contendrán información acerca de los elementos arquitectónicos que serán considerados relevantes en la arquitectura y también deberán indicar de qué manera esos elementos serán mapeados al código fuente.

El enfoque propuesto asume que en cierto momento, la documentación de la arquitectura es consistente con su implementación. A esta documentación arquitectónica inicial la denominaremos “Documentación Arquitectónica φ”. Dicha documentación consiste de un conjunto de diagramas de componentes [9] y de Use Case Maps (UCM) [7, 10]. Los primeros proveen información estructural de la arquitectura, mientras que los últimos exponen una vista de escenarios relativos a los componentes implicados en los diferentes flujos de ejecución junto con las responsabilidades involucradas en éstos, haciendo énfasis en las dependencias causales existentes entre las mismas.

Cuando los desarrolladores modifican el código fuente generando una nueva versión de la aplicación, éste deja de ser consistente con la documentación arquitectónica, causando que esta última requiera ser re-sincronizada (paso 1). Aquí, denominaremos “Source Code Δ” al grupo de clases que fueron agregadas, modificadas o eliminadas desde la última sincronización. En este punto, el Arquitecto puede comenzar con el proceso de sincronización (paso 2). Al hacerlo, la información

disponible de la documentación arquitectónica y las clases contenidas en el “Source Code Δ” son tomadas como entrada de la herramienta. El proceso tiene como primer paso la sincronización de la documentación arquitectónica estructural. Esta sincronización se debe llevar a cabo empleando el *Módulo de sincronización de diagrama de componentes* donde el Arquitecto debe seleccionar las reglas de sincronización a ser aplicadas (paso 3). Las reglas son especificadas por el Arquitecto previo al inicio del proceso de sincronización. Éstas contienen información acerca de los elementos arquitectónicos que serán considerados relevantes en la arquitectura y también indican de qué manera esos elementos serán mapeados al código fuente.

Una vez que el arquitecto seleccionó las reglas y las ejecutó, *ASAP* presentará al Arquitecto las sugerencias para re-sincronizar la documentación arquitectónica estructural (Diagrama de Componentes) y éste será el encargado de decidir cuáles de estas acciones serán aplicadas a la misma. Las sugerencias son afirmaciones que el Arquitecto debe aceptar o rechazar. Por ejemplo, una sugerencia puede ser: "Dado el paquete ‘*flabot.coremodel*’ se creará al componente ‘*flabot.coremodel*’”. Por cada sugerencia aceptada, *ASAP* actualizará el diagrama de componentes apropiadamente (paso 4). Una vez ejecutadas todas las reglas correspondientes y procesados los resultados, la documentación arquitectónica estructural quedará sincronizada, obteniendo lo que denominaremos “Documentación Arquitectónica φ + Δ1”. De este modo, el proceso de sincronización del diagrama de componentes es una tarea donde la herramienta propone un conjunto de transformaciones arquitectónicas y el Arquitecto es el encargado de analizar y decidir cuáles de estas transformaciones son adecuadas.

Teniendo el diagrama de componentes sincronizado, el Arquitecto puede proceder con la sincronización de los UCMs. Para esto, *ASAP* ofrecerá asistencia en la generación de nuevos paths para nueva funcionalidad implementada, como así también en la re-sincronización de paths existentes. En los casos en los que se agrega nueva funcionalidad a la aplicación bajo análisis, es necesario agregar nuevos paths para documentar los flujos de ejecución correspondientes. Estos flujos de ejecución pueden ser documentados haciendo uso de ASAP. Empleando esta herramienta es posible generar un path a partir de un log de ejecución de forma semi-automática. Para hacerlo, el Arquitecto debe hacer uso del módulo *Generador de paths en UCMs*. Este módulo permitirá, en primer lugar, generar un log de ejecución para la funcionalidad bajo análisis (paso 5). Luego, dicho log será empleado para determinar una secuencia de activación de responsabilidades. En base a esta secuencia, el módulo presentará una lista de las responsabilidades que se podrían agregar. Finalmente, a medida que el usuario selecciona opciones y las procesa, la herramienta actualizará el UCM agregando las responsabilidades indicadas al path creado (paso 6). Una vez generados los paths correspondientes a nueva funcionalidad, se obtendrá una nueva versión de la documentación arquitectónica, a la cual denominaremos “Documentación Arquitectónica φ + Δ2”.

Para finalizar con el proceso de sincronización, se deberá hacer uso del módulo *Sincronizador de paths existentes*. La sincronización de paths existentes comienza cuando el Arquitecto chequea los paths potencialmente modificados desde la última sincronización. Aquí, se debe ejercitar el sistema para que *ASAP* pueda generar un log de ejecución para la funcionalidad asociada (paso 7). Luego, el log generado será analizado y se lo comparará con la versión actual del path. Como resultado del análisis, *ASAP* presentará una serie de scripts candidatos con indicaciones para re-sincronizar el path (paso 8). En este caso, es tarea del Arquitecto determinar cuál de los scripts es el

correcto, y aplicar los cambios manualmente. Una vez que se finaliza con la sincronización de todos los paths potencialmente cambiados, obtendremos la versión de la documentación denominada "Documentación Arquitectónica φ + Δ3", la cual estará completamente sincronizada con el código fuente.

En resumen, el enfoque presentado permitirá sincronizar documentación de diseño, en forma de Diagramas de Componentes y UCMs, con su implementación. Para hacerlo, ASAP empleará información de mapeos entre elementos arquitectónicos y código fuente, sugiriendo la actualización de las partes de la documentación que han quedado desactualizadas. Tanto la generación de mapeos, como la sincronización de documentación arquitectónica serán realizadas de forma semi-automática. ASAP, entonces, asistirá al Arquitecto en la generación de documentación arquitectónica a partir de la implementación y posteriormente, brindará asistencia durante la evolución del código fuente para mantener la documentación arquitectónica consistente, evitando de este modo que se produzca el fenómeno de erosión arquitectónica.

3. Cronograma de actividades

A continuación, se presenta el cronograma de las actividades propuestas (y su duración estimada) que conformarán el desarrollo del trabajo final. El tiempo total de trabajo se fijó en aproximadamente seis meses. Cabe destacar que algunas actividades ya fueron realizadas con antelación (Las primeras 3 actividades y un 50% del desarrollo del enfoque).

Actividad	Duración Estimada (semanas)
Fundamentos Teóricos *	2
Relevamiento Bibliográfico *	2
Relevamiento de otras herramientas que traten la erosión arquitectónica*	2
Desarrollo del Enfoque	18
Definición de objetivos y requerimientos	1
Implementación de la funcionalidad	17
Pruebas	2
Creación y ejecución de casos de prueba	1
Documentación de las pruebas realizadas	1
Redacción de Informe Final – En paralelo con el desarrollo y las pruebas	22
Preparación Defensa Tesis	1
TOTAL	23 semanas ~ 6 meses

Notas:

- Las actividades marcadas con asterisco (*) son realizadas en paralelo.
- Las actividades en itálica son tareas padres que contienen sub-actividades, por lo tanto su duración es una sumatoria de éstas.
- Las actividades indentadas son sub-actividades.

José Martín Villalba
vilcom.martin@gmail.com

Universidad Nacional del Centro
Facultad de Ciencias Exactas
Ingeniería de Sistemas
Febrero 2013

4. Referencias

[1] L. Bass, P. Clement and R. Kazman. *Software Architecture in Practice*. 2ed. Addison-Wesley, 2003.

[2] C. Hofmeister, R. Nord and D. Soni. *Applied Software Architecture*. Addison Wesley, 2000.

[3] Sonargraph-Architect: http://www.hello2morrow.com/products/sonargraph/architect.

[4] D. Perry and A. Wolf. *Foundations for the study of software architecture*. SIGSOFT Softw. Eng. Notes, 17(4):40–52, 1992.

[5] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford. *Documenting software architectures: Views and beyond*. Addison-Wesley, 2002.

[6] N. Medvidovic and V. Jakobac. *Using software evolution to focus architectural recovery*. Automated Software Engg., 13(2):225–256, 2006.

[7] R.J.A. Buhr. *Use Case Maps: A New Model to Bridge the Gap Between Requirements and Design*. Austin, TX., USA, October 1995. Contribution to the OOPSLA 95 Use Case Map Workshop. 2, 10.

[8] R. Koschke. *Incremental Reflexion Analysis*. Madrid, Spain. IEEE Computer Society, 2010. 14th European Conference on Software Maintenance and Reengineering, CSMR'10.

[9] J. Rumbaugh, I. Jacobson and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley Longman, 1998.

[10] R.J.A. Buhr, "*Use Case Maps as Architectural Entities for Complex Systems*," IEEE Transactions on Software Engineering, vol. 24, pp. 1131-1155, 1998.

[11] D. L. Parnas and D. M. Weiss. *Active design reviews: Principles and practices*. IEEE Computer Society, 1985. 8th International Conference on Software Engineering, ICSE'85. pp. 132-136.

[12] D. Kelly and T. Shepard. *Task-directed software inspection*. Elsevier, 2004, Journal of Systems and Software, Vol. 73, pp. 361-368.

[13] F. Deissenboeck, L. Heinemann, B. Hummel and E. Juergens. *Flexible Architecture Conformance Assessment with ConQAT*. Technische Universität München, Garching b. München, Germany, 2010.

[14] M. Abi-Antoun and J. Aldrich. *Static extraction and conformance analysis of hierarchical runtime architectural structure using annotations*. ACM, 2009. OOPSLA'09. Vol. 44, pp. 321- 340.

[15] M. Pinzger. *ArchView Analyzing Evolutionary Aspects of Complex Software Systems*. Institute of Information Systems, Vienna University of Technology, Vienna. 2005.

[16] M. Abi-Antoun, J. Aldrich, D. Garlan, B. Schmerl, N. Nahas, and T. Tseng. *Improving system dependability by enforcing architectural intent*. In Proceedings of the 2005 Workshop on Architecting Dependable Systems (WADS 2005), St. Louis, MS., USA, May 2005.

[17] J. Knodel and D. Popescu. *A Comparison of Static Architecture Compliance Checking Approaches*. Kaiserslautern, Los Angeles, 2007

[18] J. Aldrich, C. Chambers and D. Notkin. *ArchJava: Connecting Software Architecture to Implementation*. Washington. 2002.

[19] N. Medvidovic, D. S. Rosenblum and R. N. Taylor. *A Language and Environment for Architecture-Based Software Development and Evolution*. In Proceedings of the 21th International Conference on Software Engineering (ICSE '99). Los Angeles, CA. May 1999.