Diseño e Implementación de un Esquema Facetado para la Publicación y Recuperación de Aspectos

Graciela Vidal y Sandra Casas

GISP - Instituto de Tecnología Aplicada
Universidad Nacional de la Patagonia Austral
Campus universitario Piloto Lero Rivera s/nro. Río Gallegos Santa Cruz
vidalgracielaunpa@gmail.com, scasas@unpa.edu.ar

Resumen. Los aspectos son requeridos por los usuarios en los repositorios de software, pero no se encuentran disponibles explícitamente. El reuso de aspectos ha sido abordado: analizando sus dependencias, utilizando la herencia para su implementación, estableciendo reglas y enfocando el reuso en aspectos que aporten calidad a las aplicaciones. Hasta el momento no se analizó el reuso de aspectos disponibles en un repositorio de software. La especificación de estos módulos es fundamental para lograr su integración en los repositorios actuales. Este trabajo expone el diseño e implementación de un proceso para la publicación y recuperación de aspectos en un repositorio, basado en un esquema facetado.

1. Introducción

El reuso de software es el proceso por el cual una organización define un conjunto de procedimientos semánticos que le permitan especificar, producir, clasificar, recuperar y adoptar artefactos software, con el objetivo de utilizarlos en sus diferentes actividades[1]. El Desarrollo de Software basado en Componentes (DSBC)[2][3] es una de las metodologías que promueve el reuso de software, para esto se requiere de un conjunto de artefactos software disponibles para ser reutilizables, lo que se denomina repositorio. Un repositorio de software es una librería en la cual se almacenan componentes software reutilizables y debe ofrecer una categorización de estos artefactos[4], además debe estar diseñado para proveer un soporte tan automatizado como sea posible que permita al usuario identificar, comparar, evaluar y recuperar los componentes.

Un artefacto de software reutilizable puede definirse como "cualquier componente desarrollado específicamente para ser utilizado en más de un contexto"[5]. Para que un artefacto sea almacenado en estos repositorios se debe proporcionar información general, información relacionada a su funcionalidad, restricciones, requerimientos y la documentación para su posterior integración y/o especialización. Una correcta especificación y categorización permitirá a los usuarios, acceder a los componentes que se requieran. Existen diversas técnicas para la especificación de componentes, se pueden citar: esquemas facetados [6], ontologías [7], frameworks[8], taxonomías[9], métodos topológicos[10]y estructurales[11], estos mecanismos han sido aplicados en el (DSBC).

El Desarrollo de Software Orientado a Aspectos (DSOA)[12] es un enfoque propuesto para encapsular Requerimientos No Funcionales (RNF)[13], también

conocidos como crosscutting concerns[14]. Los RNF (logging, sincronización, replicación, etc.) entrecruzan la funcionalidad principal de un sistema, generando el denominado código disperso y enmarañado, el cual presenta varias dificultades en su mantenimiento, reuso, etc. Algunas de las ventajas del DSOA son lograr aplicaciones más flexibles, adaptables como así también el reuso de los componentes utilizados en el sistema.

Se han explorado repositorios de software[15] que ofrecen gran variedad de artefactos software y se ha comprobado que los RNF están presentes en ellos, pero no están implementados como aspectos. Con el propósito de lograr una integración entre diferentes tipos de artefactos (componentes y aspectos), los cuales se complementan en el proceso de desarrollo de aplicaciones, resulta necesario incluir los aspectos en los repositorios.

En este trabajo se presenta un proceso para la publicación y recuperación de aspectos en un repositorio de software. El mismo consiste en extender el Framework Unificado para la Especificación de Componentes Software (FUECS)[8], con el objeto de facilitar la definición de un esquema facetado. La implementación del proceso de publicación y recuperación basado en el esquema definido, incluye una capa de indexación y búsqueda.

El presente trabajo está organizado de la siguiente manera: en la Sección 2 se exponen las motivaciones que dieron origen a esta propuesta, en Sección 3 se presenta el proceso de publicación y recuperación de aspectos, finalmente en Sección 4 se presentan las conclusiones.

2. Motivaciones

El reuso de aspectos ha sido abordado desde diferentes puntos de vista, como por ejemplo, analizando sus dependencias, los mecanismos para implementar el reuso, los requerimientos y el contexto en el cual un aspecto es reusable y por último enfocándose en aspectos que aporten calidad a las aplicaciones. A continuación se describen brevemente algunos de estos enfoques.

En[16] se analiza el reuso basándose en las dependencias entre aspectos, proponiendo una clasificación considerando dos características de estos módulos. La primera tiene en cuenta la manera en que interactúan unos con otros y la segunda se basa en la forma en que despliegan su funcionalidad. Un aspecto es ortogonal cuando su funcionalidad es independiente de otras funcionalidades de la aplicación, un aspecto es unidireccional si depende de alguna funcionalidad del sistema, puede referirse a datos o servicios, por último un aspecto es circular si depende mutuamente con otro aspecto, en algunos casos no se considerarían aspectos uno sin el otro. La segunda clasificación propone aspectos autónomos, aquellos que se ejecutan por sí solos y aspectos activados, requieren que alguna parte del sistema los active.

La herencia es un mecanismo utilizado para la implementación del reuso en AspectJ[17], se han definido reglas y se establece el desarrollo de al menos tres aspectos para llevar a cabo el reuso[18] eliminando algunos inconvenientes que presenta. Las cuatro reglas son:

Regla 1: declaración y definición de pointcut por separado, en un aspecto abstract.

Regla 2: definición de pointcut hook, dado que AspectJ no permite la definición recursiva de pointcut, es decir no puede redefinirse un pointcut definido en un aspecto abstract. En estos casos se debe definir y declara el pointcut para luego ser asignado a otro pointcut en blanco.

Regla 3: no deben existir un pointcut para más de un advice, en AspectJ podría ser necesario, en esos casos se debe definir utilizar un pointcut hook y luego asignarlo a diferentes pointcuts.

Regla 4: los aspectos concretos no deben contener atributos, métodos o advices que puedan ser reusados.

Además de estas reglas, es necesaria la definición de al menos tres aspectos, uno para la declaración de pointcuts, un aspecto para declarar atributos, métodos y advices, por último un aspecto concreto y vacío, todos conectados por medio de la herencia.

Los aspectos son válidos bajo ciertas condiciones de contexto[19], se requiere comprobar la compatibilidad y los requerimientos que definen su contexto de uso. La utilización de pre y post condiciones de la programación de diseño por contratos son una alternativa a utilizar para la especificación de un aspecto reusable.

Por último en [20] se aborda el reuso sobre propiedades que aportan calidad a una aplicación y no sobre la funcionalidad como es habitual. Estas propiedades, son difíciles de identificar, clasificar y catalogar, por lo que no es suficiente basarse en su taxonomía[21][22].El punto de partida del artículo es enfocarse en los requerimientos no funcionales [13], ingeniería de requerimientos orientada a objetivos[23], programación orientada a aspectos[12], reuso de software[24] y gestión de calidad.

La reutilización de software es una característica deseable por todos los enfoques mencionados, sin embargo no se ha analizado el reuso de aspectos disponibles en un repositorio de software. Es por ello que se ha trabajado en la posibilidad de incluirlos como módulos independientes, implementados en lenguajes orientados a aspectos, lo que implica definir una especificación para su publicación y recuperación, proporcionando la información necesaria para su integración y reuso. En la Sección 3 se presenta un esquema facetado para la especificación de aspectos reutilizables.

3. Publicación y Recuperación de Aspectos

Los repositorios de software son fundamentales para el DSBC, en ellos se almacenan los artefactos que se encuentran disponibles para ser recuperados y reusados[25]. Para esto, un repositorio de software debe reunir ciertas características:

- Almacenar activos software reusables
- Proporcionar una interface que permita la publicación, búsqueda y recuperación de activos
- Gestionar su permanente actualización y mantenimiento
- Clasificar los activos software
- Ofrecer la navegación de los activos disponibles
- Aplicar técnicas de búsquedas y recuperación automatizadas[26]

Además los repositorios deben estar respaldados por una definición, especificación y contextualización de sus artefactos, entre ellos los componentes, clases y aspectos.

Un aspecto es la abstracción central del DSOA, el cual se compone de dos partes modulares: pointcuts y advices, además de atributos y métodos. Los advices son similares a los métodos, debido a que son fragmentos de código que serán incorporados al dominio en algún momento (after, before, around); los pointcuts son las expresiones que establecen ante qué eventos y condiciones estos fragmentos se ejecutarán, por ejemplo la llamada a un determinado método[12]. En Ejemplo 1 se presenta el código del aspecto StateManager implementado en AspectJ, en el cual se definen los pointcuts addVariable y removeVariable que capturan las llamadas y ejecuciones respectivamente, de los métodos detallados y están asociados a los advices before y after.

```
public aspect StateManager
{
 pointcut addVariable(): call (* …set*(..));
 pointcut removeVariable():execution (*
someComplexMethod(..));
 before() : addVariable()
   { …. }
 after(): removeVariable():
   { …. }
}
```

Ejemplo 1. Implementación en AspectJ del aspecto StateManager

3.1 Definición del esquema facetado

En [8] se presenta el FUECS, cuyo objetivo es proveer la información necesaria para facilitar el desarrollo, descubrimiento y configuración de componentes. Conformado por un conjunto de campos específicos agrupados en formularios denominados páginas, a las cuales se las identifica con un color. La página blanca contiene información general y comercial (nombre,autor,versión,precio,etc.), la página amarilla provee la categorización (dominio,tecnología,etc.), la página azul describe la funcionalidad (procesos,tareas,etc.), en la verde se definen las interfaces especificadas y la página gris aporta información extra sobre la calidad del componente.

En Figura 1, se presenta el FUECS extendido para ser aplicado a aspectos con la incorporación de una página de color rojo denominada Tejido, en la misma se incluyen conceptos dinámicos y estáticos. Los conceptos dinámicos permiten la definición de pointcut, advices y orden de ejecución cuando corresponda. En conceptos estáticos se definen las introducciones y herencia. En Figura 2 se muestra la especificación del aspecto StateManager utilizando el esquema facetado definido en XML.

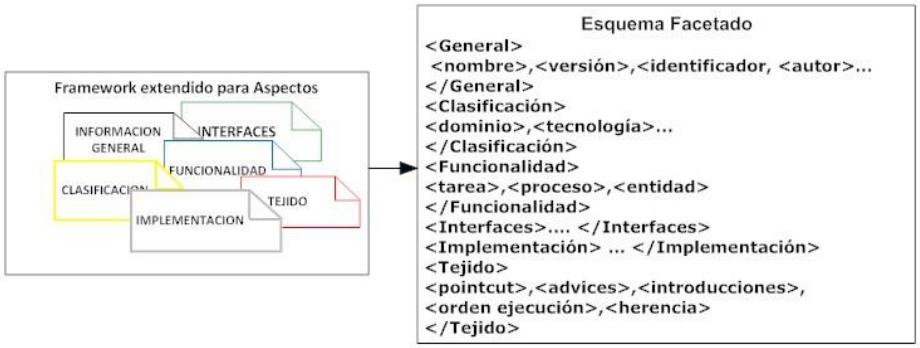

Figura 1. Framework extendido y esquema facetado para especificación de aspectos

```
<aspecto>
 <general>
      <nombre>StateManager</nombre>
      <version>1.0</version>
 </general>
<clasificacion>
      <dominio>Movilidad de Codigo</dominio>
      <reuso>lógico</reuso>
      <tecnologia>AspectJ</tecnologia>
</clasificacion>
<funcionalidad> .... </funcionalidad>
<interfaces> ... </interfaces>
<implementación> .... </implementación>
<tejido>
      <pointcut>addVariable(): call (* ...set*(..))</pointcut>
      <pointcut>removeVariable():execution (* someComplexMethod(..))</pointcut>
      <advice>before:addVariable()</advice>
      <advice>after:addVariable()</advice>
</tejido>
</aspecto>
```

Figura 2. Esquema facetado aplicado al aspecto StateManager

3.2 Proceso de Publicación y Recuperación de Artefactos Software

En Figura 3 se presenta proceso de publicación y recuperación de aspectos a partir del esquema facetado definido. En este gráfico se representan dos instancias:

- **Publicación**

a. El usuario A completa un formulario con la especificación definida en el esquema facetado y adjunta el código fuente del aspecto.

b. La especificación de cada aspecto genera un archivo XML similar al de Figura 2.

c. El algoritmo de publicación analiza cada archivo XML y luego de indexarlos, genera un documento que luego es incorporado a un índice.

- **Recuperación**

d. El usuario B completa un formulario con las características del artefacto software que necesita.

e. El algoritmo de recuperación analiza esta consulta sobre el índice de documentos.

f. Los documentos que cumplan con las condiciones y/o restricciones de la consulta son exhibidos al usuario, para luego seleccionar los aspectos a ser recuperados.

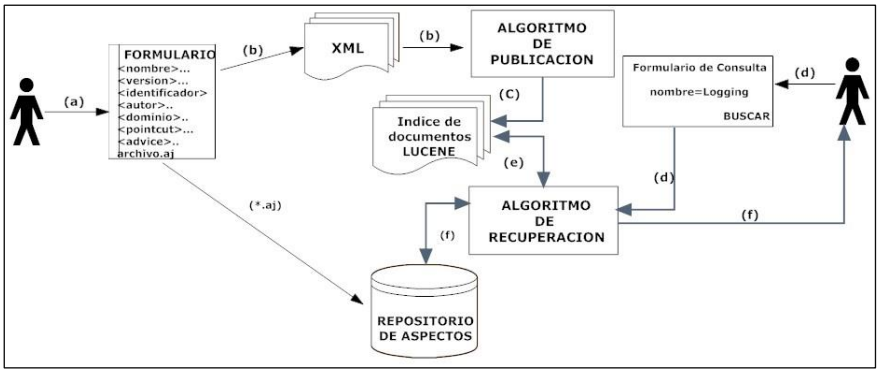

Figura 3. Proceso de publicación y recuperación

3.3 Implementación de la herramienta

Se ha desarrollado una herramienta del estilo "prueba de conceptos" para la implementación del proceso descripto en la Sección 3.2. A partir del esquema facetado definido se diseña un formulario de publicación de artefactos, cada especificación es almacenada en un archivo XML. Esto significa que cada faceta del formulario representa una etiqueta del archivo XML y la información de estos campos es el contenido de las etiquetas. Como mecanismo para trasladar estas facetas y su contenido a un medio de rápido y fácil acceso para el proceso de recuperación, se utiliza Apache Lucene API[27].

El algoritmo de publicación incluye un proceso de indexación, el cual genera un índice de documentos sobre los cuales se analizan las consultas. La estrategia elegida permite seleccionar la información que incluyen estos documentos, los desarrolladores pueden extraer información de los archivos indexados, cualquier tipo de información de texto.

Para esta herramienta se implementó un algoritmo en el cual se define que los documentos estarán conformados por dos campos "tags" y "descripción", los cuales obtienen la información de los elementos del archivo XML que contiene la especificación de cada aspecto. El campo "tags" obtiene el nombre de las etiquetas y

el campo “descripción” almacena el contenido. En el Ejemplo 2 se muestra el resultado de este proceso sobre la especificación del aspecto Sincronización.

Especificación XML	Documento del Índice Apache Lucene
<aspecto> <general> <nombre>Sincronización</nombre> <versión>1.0<versión> <identificador>Sin1.0<identificador> </general> …. </aspecto>	**Tags** **Descripción** nombre Sincronización versión 1.0 identificador Sin1.0 … …

Ejemplo 2. Documento generado a partir de la especificación XML.

Las consultas de recuperación pueden basarse en texto libre por ejemplo “logging”, “call”, “AspectJ” o ser más sofisticadas utilizando los operadores AND, OR y los comodines ~,^.Por ejemplo “nombre=Logging AND tecnología=AspectJ” o “AspectJ OR Java”. El algoritmo de recuperación implementado toma esta información y genera la consulta, luego analiza cada documento del índice. Como resultado de la consulta se obtiene una estructura que almacena los documentos que cumplen con las condiciones de la consulta, los cuales son presentados al usuario como primer paso para la recuperación. En el Ejemplo 3 se muestra una consulta para ser analizada a partir del formulario enviado por un usuario.

Consulta de recuperación	Consulta sobre el Índice
Nombre: Logging Categoría: Seguridad	Tags:Nombre AND Descripción:Logging OR Tags:Categoría AND Descripción:Seguridad

Ejemplo 3. Consulta para la recuperación de un aspecto.

4. Conclusiones

Los RNF se encuentran disponibles en los repositorios de software actuales, lo que demuestra que son solicitados por los usuarios para ser reusados, pero no están disponibles como aspectos. El reuso de aspectos fue abordado desde diferentes puntos de vista, basándose en sus dependencias, definiendo un conjunto de reglas y utilizando la herencia como mecanismo de reuso, por mencionar algunos.

Esto significa que los aspectos son requeridos y son reusables, lo que motivo al análisis de su almacenamiento junto a otros componentes, por considerar que de esta manera, se posibilitará un mayor reuso de aspectos.

Es por ello que este trabajo propone como primer paso para la incorporación de aspectos a los repositorios de software la extensión del FUECS. Lo cual consiste en incorporar una página en la que se definen conceptos específicos de los aspectos.

El framework ha permitido definir un esquema facetado para la especificación de estos módulos. Se ha desarrollado una herramienta del estilo “prueba de conceptos” para implementar el proceso de publicación y recuperación de aspectos basándose en el esquema facetado.

Como trabajo futuro se definirá un plan de prueba y validación sobre la herramienta. El mismo consistirá en una fase de publicación y otra de recuperación de componentes y aspectos. En la primera instancia se agruparán diferentes tipos de artefactos software (componentes y aspectos) para ser publicados. La segunda etapa consistirá en la definición de un conjunto de consultas basadas en diferentes criterios para ser analizadas sobre el índice generado por la especificación de los aspectos publicados.

Referencias

1. Mili, A., Yacoub, S., Addy, E. & Hafedh, M., Reuse Based Software Engineering, John Wiley & Sons, INC.(2002)
2. González R., Torres. M. Algunas Implicaciones del Desarrollo Basado en Componentes. paper. Bogotá, Colombia: Departamento de Ingeniería de Sistemas Universidad Javeriana; (2005)
3. N. A. y. J. A. C. Jonás A. Montilva C., Desarrollo de software basado en componentes IV Congreso de Automatización y Control, Maracaibo , Mérida - Venezuela, pp 9.(2003)
4. S. B. Amandeep Bakshi, Development of a software repository for the precise search and exact retrieval of the components, International Journal of Advanced Research in Computer Science and Software Engineering 3, no. 8,pp 1116-1126 (2013)
5. Kalathipparambil J. and Vasantha, A Mixed Method Approach for Efficient component retrieval from a component repository. Journal of Software Engineering and Applications, (2011)
6. Prieto-Diáz, R.: Implementing Faceted Classification for Software Reuse. In: Communications of the ACM 34 (1991)
7. Dragan G., Nima K., Milan M., Ontologies and Software Engineering (2008)
8. Overhage S. Towards a Standardized Specification Framework for Component Development, Discovery, and Configuration. Augsburg University. Dept. of Application Engineering and Business Information Systems (2003)
9. S.-S. Maninder, Goel., Identifying asset type for various reusable component storage/retrieval methods. Proceedings of National Conference on Challenges & Opportunities in Information Technology - RIMT-IET, Mandi Gobindgarh, pp. pp.38-41 (2007)
10. G. Forbes, McCartan C., Ruairi O'Donnell, Niall S. and Leon R. The integration of information retrieval techniques within a software reuse environment. Department of Information Science, Strathclyde University, 26 Richmond St., Glasgow, G1 1XH, UK (2000)
11. Klein, M. and A. Bernstein. Searching for Services on the Semantic Web Using Process Ontologies. The First Semantic Web Working Symposium. Stanford, CA, USA. (2001)
12. Kiczales G., Lamping J., Mendhekar A., et al. Aspect-Oriented Programming. In: Springer-Verlag, editor. Finland.: European Conference on Object-Oriented Programming; pp25 (1997)
13. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: A process-oriented approach. IEEE Trans. Software. Eng. 18 pp.483–497 (1992)
14. Filman, R., et al., Aspect-Oriented Software Development. Addison-Wesley (2004)
15. Vidal G., Casas S.y Marcos C. Exploración de repositorios de software y análisis de potenciales extensiones a aspectos. Aprobado por Resolución 0705/13-R-UNPA.Código ICT-UNPA-64-2013, ISSN 1852-4516 disponible en http://ict.unpa.edu.ar. (2014)

16. Kienzle J., Yu Y., Xiong J. On Composition and Reuse of Aspects. School of Computer Science McGill University. Montreal, QC H3A 2A7. Canada (2003)
17. Sitio web de AspectJ www.eclipse.org/aspectj (2014)
18. S. a. R. Hanenberg, Unland, Using and reusing aspects in AspectJ. Institute for Computer Science - University of Essen, Germany p. 6.(2001)
19. D. Wampler, The challenges of writing reusable and portable aspects in aspectJ: Lessons from contract4j. Aspect Research Associates and New Aspects of Software, pp.9 (2006)
20. Sampaio do Prado Leite J., Yu Y., Liu L., Yu E. and Mylopoulos J. Quality-Based Software Reuse. Departamento de Informática, Pontifıcia Universidad Católica do Rio de Janeiro, RJ 22453-900, Brasil. Department of Computer Science, University of Toronto, M5S 3E4 Canada . School of Software, Tsinghua University, Beijing, 100084, China. (2005)
21. Sommerville, I.: Software Engineering, 4th Ed. Addison-Wesley (1992)
22. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: ICSE, International Conference on Software Engineering, IEEE Computer Society Press pp. 592–605 (1976)
23. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements analysis. CACM 42 pp.31–37 (1999)
24. Krueger, C.: Software reuse. ACM Computer Survey 24 pp.131–183 (1992)
25. Nedhal A., Al Saiyd, Intisar A. Al Said, Takrori AHA. Semantic-Based Retrieving Model of Reuse Software Component: Computer Science Department (2010).
26. Kuljit Kaur PK, Jaspreet Bedi, and Hardeep Singh. Towards a Suitable and Systematic Approach for Component Based Software Development. World Academy of Science, Engineering and Technology (2007).
27. McCandless M.,Hatcher E., Gospodnetic O. Lucene in Action. Second Edition: Covers Apache Lucene 3.0. ISBN-10: 1933988177 ISBN-13: 978-1933988177. Manning Publications. (2010)