CAEPIA 2005

Vol. I

Entidades organizadoras

UNIVERSIDADE DA CORUÑA

USC

Entidades patrocinadoras

MINISTERIO DE EDUCACIÓN Y CIENCIA

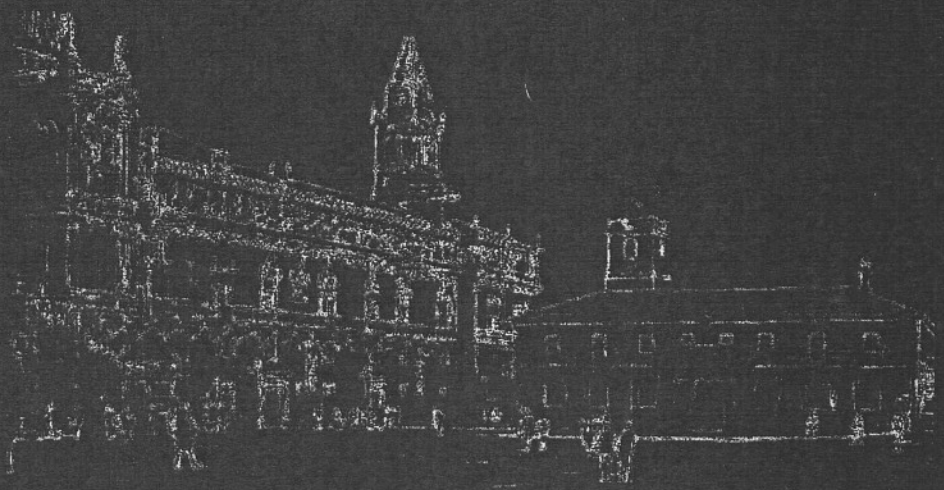
XUNTA DE GALICIA

XUNTA DE GALICIA

CONCELLO DE SANTIAGO

XI Conferencia de la Asociación Española para la Inteligencia Artificial

Santiago de Compostela
16-18 de Noviembre 2005



Actas, Volumen I

XI Conferencia de la Asociación Española para la Inteligencia Artificial

CAEPIA 2005

Santiago de Compostela, 16-18 de Noviembre de 2005

Actas, Volumen I

Entidades Organizadoras:

UNIVERSIDADE DA CORUÑA

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Entidades Colaboradoras:

- Ministerio de Educación y Ciencia.
- Consellería de Innovación e Industria (Dirección Xeral de Investigación, Desenvolvemento e Innovación), Xunta de Galicia.
- Consellería Educación e Ordenación Universitaria, Xunta de Galicia.
- Universidad de A Coruña.
- Universidad de Santiago de Compostela.

Editores:

Roque Marín
Eva Onaindia
Alberto Bugarín
José Santos

Imprime: CopyNino, Santiago de Compostela.
ISBN: 84-96474-13-5
Depósito Legal: C-2534-05

Prefacio

Hace poco más de cincuenta años, el 31 de Agosto de 1955, Marvin Minsky, John McCarthy, Nathan Rochester y Claude Shannon propusieron la celebración de una reunión de dos meses de duración, que tuvo lugar en el Dartmouth College durante el verano de 1956. En la llamada a la participación indicaban que el objetivo era discutir "la conjetura de que todos los aspectos del aprendizaje o de cualquier otra característica de la inteligencia pueden, en principio, ser descritos de modo tan preciso que se pueda construir una máquina capaz de simularlos". El tema era tan novedoso que acuñaron un nuevo término para él: Inteligencia Artificial (IA).

Está, pues, a punto de cumplirse el 50 aniversario de lo que se considera como el nacimiento histórico del campo de la IA. Ello nos brinda la oportunidad de lanzar una mirada crítica hacia el estado actual de la IA en el mundo, y en España en particular. Es tiempo también de expresar nuestro reconocimiento hacia las sucesivas generaciones de investigadores que han contribuido a consolidar y hacer avanzar este campo.

La Asociación Española para la Inteligencia Artificial (AEPIA), creada en 1983, ha contribuido a guiar y aglutinar a los investigadores españoles en IA durante más de 20 años, casi la mitad de esta andadura. AEPIA es miembro del ECCAI (European coordinating committee for Artificial Intelligence) y miembro fundador de IBERAMIA. AEPIA organiza bienalmente la Conferencia de la Asociación, cuyas ediciones anteriores se han celebrado en Madrid, Alicante, Málaga, Murcia, Gijón y Donostia.

Este año 2005, la XI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA'05) es acogida por la ciudad de Santiago de Compostela, cuya hospitalidad agradecemos sinceramente, y está organizada de modo conjunto por las Universidades de A Coruña y Santiago de Compostela.

Como en las anteriores ediciones, los objetivos de esta Conferencia son: facilitar la diseminación de nuevas ideas y experiencias, fortalecer los lazos entre los distintos grupos de investigación implicados, promover el trasvase de conocimiento entre nuevos investigadores y grupos consolidados, y ayudar a difundir los nuevos desarrollos hacia la sociedad. En CAEPIA'05, la comunidad de investigadores que trabaja en temas relacionados con la Inteligencia Artificial se reúne para presentar y discutir los últimos avances científicos y tecnológicos en este campo. Se acompaña, además, de otros eventos, como Talleres, Paneles y Tutoriales.

Ha sido nuestra pretensión consolidar y aumentar el nivel científico alcanzado en ediciones anteriores. Hemos contado, en esta undécima edición, con la participación excepcional, como conferenciantes invitados, del profesor Stephen Muggleton, director del *Computational Bioinformatics Laboratory* en el *Imperial College*, Londres; del profesor Luc Steels de la *Free University of Brussels* y director del *Sony Computer Science Laboratory* en Paris; del profesor Gerhard Brewka, del *Intelligent Systems Department - Computer Science Institute* de la Universidad de Leipzig, Alemania; de Peter Lucas, del *Institute for Computer and Information Sciences* de la Universidad de Nijmegen, Holanda. Finalmente, el profesor Vicente Botti, de la

Índice / Table of Contents

Volumen I

[4] Bratman, M.E. (1987). Intentions, Plans and Practical Reason. Harvard University Press, Cambridge, M.A.
[5] Corchado J. M. and Laza R. (2003). Constructing Deliberative Agents with Case-based Reasoning Technology, International Journal of Intelligent Systems. Vol 18, No. 12, December. pp.: 1227-1241
[6] Corchado J. M. and Lees B. (2001). A Hybrid Case-based Model for Forecasting. Applied Artificial Intelligence. Vol 15, no. 2, pp.105-127.
[7] Corchado J. M., Pavón J., Corchado E. and Castillo L. F. (2005) Development of CBR-BDI Agents: A Tourist Guide Application. 7th European Conference on Case-based Reasoning 2004. Lecture Notes in Artificial Intelligence 3155, Springer Verlag. pp. 547-559.
[8] DeLoach, S. (2001) Anlysis and Design using MaSE and AgentTool. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS).
[9] EURESCOM (2001) MESSAGE: Methodology for engineering systems of software agents. Technical report P907-TI1, EURESCOM.
[10] Glez-Bedia M. and Corchado J. M. (2002) A planning strategy based on variational calculus for deliberative agents. Computing and Information Systems Journal. Vol 10, No 1, 2002. ISBN: 1352-9404, pp. 2-14.
[11] Glez-Bedia M., Corchado J. M., Corchado E. S. and Fyfe C. (2002) Analytical Model for Constructing Deliberative Agents, Engineering Intelligent Systems, Vol 3: pp. 173-185.
[12] Iglesias, C., Garijo, M., Gonzalez J.C. and Velasco J. R. (1998) Analysis and Design using MAS-CommonKADS. Inteligent Agents IV LNAI Volume 1365 Springer Verlag.
[13] Lefevre N., Aiken J., Rutllant J., Daneri G., Lavender S. and Smyth T. (2002) Observations of pCO2 in the coastal upwelling off Chile: Sapatial and temporal extrapolation using satellite data. Journal of Geophysical research. Vol. 107, no. 0
[14] Nwana H.S., Ndumu, D. T., Lee, L. C. and Collins J. C. (1999) ZEUS: A Toolkit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence Jounal, vol 1, nº13, pp. 129-185.
[15] Odell, J., Levy R., and Nodine M. (2004) FIPA Modeling TC: Agent Class Superstructure Metamodel. FIPA meeting and interim work.
[16] Odell, J. and Huget, M. P. (2003) FIPA Modeling: Interaction Diagrams.
[17] Pokahr, A., Braubach, L. and Lamersdorf W. (2003) Jadex: Implementing a BDI-Infrastructure for JADE Agents, in: EXP - In Search of Innovation (Special Issue on JADE), Vol 3, Nr. 3, Telecom Italia Lab, Turin, Italy, September 2003, pp. 76-85.
[18] Santamaría J. and Nieto J. (2000) Los agujeros del cambio climático. World Watch no. 12. pp 62-65.
[19] Sarmiento J. L. and Dender M. (1994) Carbon biogeochemistry and climate change. Photosynthesis Research, Vol. 39, 209-234.
[20] Takahashi T., Olafsson J., Goddard J. G., Chipman D. W. and Sutherland S. C. (1993) Seasonal Variation of CO2 and nutrients in the High-latitude surface oceans: a comparative study. Global biochemical Cycles. Vol. 7, no. 4. pp 843-878.
[21] Wooldridge, M. and Jennings, N. R. (1995) Agent Theories, Architectures, and Languages: a Survey. In: Wooldridge and Jennings, editors, Intelligent Agents, Springer-Verlag, pp. 1-22.
[22] Wooldridge, M. and Jennings, N. R. and Kinny, D. (2000) The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, 3 (3). pp. 285-312.
[23] Wooldridge, M. and Jennings, N. R. (1995) Agent Theories, Architectures, and Languages: a Survey. In: Wooldridge and Jennings, editors, Intelligent Agents, Springer-Verlag, pp. 1-22.

Aplicando Gestión del Conocimiento y Razonamiento Basado en Casos en el Proceso de Mantenimiento del Software

Aurora Vizcaíno, Juan P. Soto, Félix García, Francisco Ruiz, Mario Piattini

Universidad de Castilla-La Mancha, Escuela Superior de Informática, España
{Aurora.Vizcaino| Felix.Garcia | Francisco.RuizG | Mario.Piattini}@uclm.es
jpsoto@proyectos.inf-cr.uclm.es

Resumen. En toda organización es conveniente que la información y el conocimiento se procesen y almacenen de forma que estos se puedan reutilizar. En el caso del mantenimiento del software es todavía más importante realizar una buena gestión de la información y del conocimiento ya que estos provienen de distintas fuentes y etapas del ciclo de vida. Sin embargo, en la actualidad existen muy pocos trabajos enfocados en la aplicación de técnicas de gestión del conocimiento en el mantenimiento del software. En este artículo se describe cómo hemos definido los conceptos involucrados en el mantenimiento del software y cómo se han representado en una ontología, la cual posteriormente ha sido implementada en un sistema de gestión del conocimiento usando REFSENO. Todo ello con el fin de potenciar la reutilización de la información (usando técnicas de razonamiento basado en casos) y que de está forma los ingenieros de mantenimiento puedan aprovechar la experiencia y lecciones aprendidas de otros trabajadores.

Palabras Claves: Mantenimiento del software, gestión del conocimiento, ontologías, razonamiento basado en casos, REFSENO.

1. Introducción

Los procesos de desarrollo y mantenimiento del software generan diariamente gran cantidad de información [19]. Sin embargo, esta información raramente se gestiona de forma que se sepa quien la ha generado, dónde se va a almacenar y a quién le puede resultar útil, lo cual implica que con frecuencia se "reinvente la rueda" y no se aproveche el "capital intelectual" de una organización. Este hecho es aún más grave en el caso del mantenimiento, el cual es considerado el proceso más costoso (en cuanto a dinero y esfuerzo) y más largo del ciclo de vida del software [4, 15, 16]. Durante esta etapa la información no sólo proviene de los profesionales involucrados en las actividades de mantenimiento, sino también del propio producto que se mantiene, de las causas que motivaron el mantenimiento, de los clientes y usuarios, así como de los procesos, metodologías y herramientas utilizadas por la organización.

Además, la información va cambiando constantemente debido a la propia naturaleza del mantenimiento que surge, entre otras razones, de la necesidad de adaptar sistemas software a un entorno que a su vez está siempre cambiando [14]. Por la misma razón el conocimiento que un ingeniero de mantenimiento posee también evoluciona. Por ejemplo durante la fase de "desarrollo inicial" el grupo de mantenimiento adquiere conocimiento acerca del dominio de la aplicación, requisitos de usuario, roles, algoritmos, formato de los datos, capacidades y debilidades de la arquitectura del programa, etc. En la fase de "evolución del software" surgen nuevos usuarios y requisitos que propician nuevos cambios y producen nuevo conocimiento en los ingenieros [3], y de forma similar cada fase genera nueva información que es interiorizada [13] por los ingenieros, que a su vez le añaden su experiencia convirtiendo la información en conocimiento tácito. En el resto del artículo, cuando la fuente sea un ingeniero le llamaremos conocimiento y cuando sea un documento u otro artefacto le llamaremos información.

Por otra parte, cada ingeniero de mantenimiento posee un conocimiento que puede ser útil para los demás miembros del equipo. Por ejemplo, cuando una persona modifica un método que afecta a otro que a su vez va a ser modificado por otra persona, la primera debería informar a la segunda. En consecuencia, debe existir cierta colaboración y comunicación entre los ingenieros del mantenimiento y así puede ser muy conveniente saber qué actividades está desarrollando cada empleado en cada momento. Otro aspecto a tener en cuenta en el proceso de mantenimiento es la experiencia que el empleado posee, ya que el factor que más influye en la productividad del mantenimiento es la presencia de por lo menos un miembro del equipo con experiencia en la aplicación que se está manteniendo [1]. En [2] se explica que los sistemas mantenidos por programadores sin experiencia suelen tener mayor número de errores en promedio. Esto demuestra la importancia que la experiencia tiene en el proceso de mantenimiento. Por ello, es útil capturar, en la medida de lo posible, parte de la experiencia que un empleado posee y almacenarla en un sistema de forma que esta experiencia pueda ser aprovechada por nuevos empleados o por otros compañeros.

Un problema adicional que complica el proceso de mantenimiento es la escasa documentación que suele existir relacionada con el software a mantener y la obsolescencia de la documentación asociada. Este hecho se acentúa en el caso de software heredado de otras organizaciones sin documentación que describa sus características [20].

Por todos estos motivos es necesario desarrollar técnicas y herramientas que ayuden a los ingenieros del mantenimiento a realizar sus tareas. Nuestra propuesta consiste en el desarrollo de un sistema de gestión de conocimiento capaz de procesar y almacenar la información generada durante el proceso del mantenimiento, con el fin de diseminarlo y reutilizarlo. Además, ello permitirá disminuir la probabilidad de repetir errores, incrementando la competitividad de la organización [5].

Como requisito previo a la construcción de un sistema de gestión de conocimiento era necesario modelar la información generada durante el proceso de mantenimiento del software y además estructurarla de forma que se facilitara su reutilización. Con dicho fin hemos desarrollado un modelo del proceso de mantenimiento del software en el cual los objetos, conceptos, entidades y sus relaciones son representados explícitamente. Como consecuencia, hemos desarrollado un modelo del proceso de

mantenimiento del software y su correspondiente ontología que ha sido implementada usando la metodología REFSENO (Representation Formalism for Software Engineering Ontologies). Una ventaja de REFSENO es que facilita la utilización de técnicas de razonamiento basado en casos, lo cual permite que el sistema de forma automática proponga soluciones a problemas parecidos a otros que han ocurrido anteriormente.

El resto de este artículo esta estructurado de la siguiente forma: la sección 2 presenta una descripción de la ontología y su implementación con REFSENO. La sección 3 ilustra cómo la técnica del razonamiento basado en casos fue utilizada para reutilizar la información. Finalmente se presentan las conclusiones.

2. Modelado e Implementación de una Ontología para el Mantenimiento del Software

Una ontología define de forma explicita y sin ambigüedad los conceptos de un dominio y sus relaciones [6]. Además, una ontología facilita la gestión de conocimiento [11], su compartición e integración [7].

La ontología que se describe en este artículo está basada en la ontología de Kitchenham et al´s [10], en la que se definen todos los conceptos relevantes para la clasificación de estudios empíricos en el mantenimiento del software. Dicha ontología está dividida en varias sub-ontologías:

- Sub-ontologia de productos: define los productos software que son mantenidos, su estructura y su evolución.
- Sub-ontologia de actividades: describe los distintos tipos de actividades para el mantenimiento del software
- Sub-ontología de procesos: esta subontología está dividida en dos diferentes enfoques, definiéndose una subontología para cada uno de ellos:
    1. Sub-ontología de procedimientos: define cómo aplicar métodos, técnicas y herramientas a las actividades y cómo los recursos son utilizados para poder llevar a cabo las actividades.
    2. Sub-ontología de organización del proceso: esta sub-ontología se enfoca en dar soporte a los procesos organizacionales relacionados con las actividades del mantenimiento del software, y cómo el ingeniero de mantenimiento debe organizarse, así como las obligaciones que éste contrae.
- Sub-ontología de agentes: describe las habilidades y roles necesarios para llevar a cabo una actividad, qué tipo de responsabilidades tiene cada persona, y cómo los actores que intervienen en los procesos de la organización (ingeniero de mantenimiento, cliente y usuario) se relacionan.

Otros autores también han basado sus propuestas en la ontología de Kitchenham, como Oliveira et al, que presentan una ontología para determinar el conocimiento que es necesario para los encargados del mantenimiento del software [14]. En su ontología se proponen cinco aspectos en lugar de los cuatro descritos anteriormente, sin

embargo, cuatro de ellos son similares a las sub-ontologías anteriormente descritas. Los aspectos considerados en [14] son: el conocimiento acerca del sistema que corresponde a la subontología del producto, el conocimiento relacionado con las habilidades del ingeniero de mantenimiento que corresponde a la sub-ontología de agentes; el conocimiento acerca de la actividad del mantenimiento la cual corresponde a la sub-ontología de actividades, el conocimiento acerca de la estructura de la organización que corresponde a la sub-ontología de procesos de la organización y el conocimiento relacionado con el dominio de aplicación, el cual no ha sido considerado por la ontología de Kitchenham.

Deridder presenta una ontología orientada a conceptos, es decir, se enfoca en los conceptos que son reutilizados, los cuales, por otro lado, dependen de cada tipo de organización [6]. Sin embargo, nuestro objetivo es definir una ontología con un nivel de abstracción más elevado, enfocado en la gestión de proyectos de mantenimiento del software con un punto de vista de procesos de negocio.

2.1 Ontología del Mantenimiento

Este nombre es una simplificación ya que, realmente, se refiere a la ontología de la gestión de proyectos de mantenimiento de software. Para poder gestionar un proyecto de mantenimiento de software, es necesario identificar los factores que influyen en el mantenimiento, tal y como lo hace la ontología de Kitchenham. Además, es necesario indicar un conjunto de aspectos dinámicos del dominio, descritos en términos de estados, eventos, procesos, etc. Por ello, hemos extendido la ontología de Kitchenham especificando tanto aspectos estáticos como dinámicos (que por razones de claridad se han desglosado en tres ontologías y cuatro subontologías). Los aspectos dinámicos son modelados mediante flujos de trabajo [17].

Para poder representar los aspectos estáticos, se definió una ontología llamada "Mantenimiento", la cual se divide en cuatro subontologías (tal como muestra la Figura 1): La *sub-ontología de los productos*, la *sub-ontología de las actividades*, la *sub-ontología de organización del proceso* y una cuarta llamada *sub-obtonlogía de agentes*. El número de ontologías estáticas coincide con las propuestas en [10]. No obstante, éstas han sido extendidas, añadiendo conceptos relacionados con la gestión de proyectos de mantenimiento, por ejemplo incorporando el concepto de Versión.

La parte dinámica es representada por la ontología de los *flujos de trabajo*, la cual define los siguientes tres aspectos relevantes para el mantenimiento:

- Descomposición de actividades
- Restricciones temporales entre las actividades (por ejemplo, el orden en el que deben realizarse las actividades).
- Control de las actividades y la ejecución de los procesos durante el desarrollo de los proyectos.

Además, hemos diseñado una tercera ontología llamada *Ontología de la medida* que representa aspectos tanto dinámicos como estáticos relacionados con conceptos de medición.

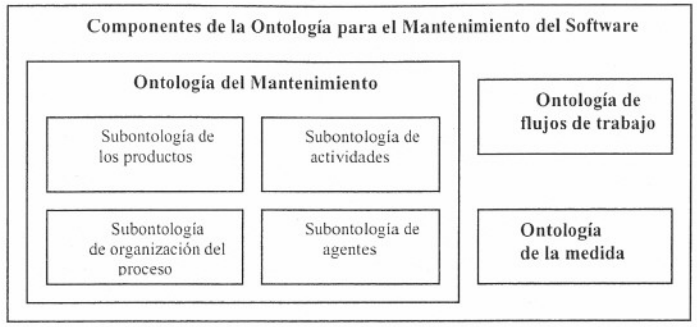


Figura 1. Estructura de la ontología para el mantenimiento del software

Por cuestiones de espacio, en este trabajo sólo se presenta un ejemplo de cómo hemos modelado e implementado la ontología. Una descripción más detallada se puede encontrar en [18]. En la figura 2 se muestra, mediante el uso de diagramas de clases en UML, el diagrama ontológico de la sub-ontología de los productos, la cual hace énfasis en el componente producto ya que representa el concepto más importante. Esta sub-ontología define los productos software que son mantenidos, su estructura interna, composición y las versiones que existen de ellos.

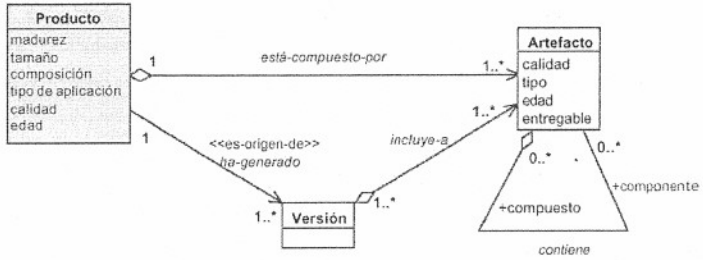


Figura 2. Diagrama de la subontología de los productos

Como se puede apreciar en la Figura 2, un producto software puede tener diferentes versiones, las cuales están formadas por un conjunto de artefactos. Por ejemplo, para un producto llamado "Ventas" pueden existir diferentes versiones y cada versión puede estar compuesta de varios artefactos. El concepto versión tiene sus propios atributos, tales como número, fecha, etc., pero por cuestiones de simplicidad, no se han representado en el diagrama. El diagrama anterior sólo muestra una visión resumida de la ontología en cuestión.

2.3 Implementación de la Ontología

Con el fin de sistematizar la implementación de la ontología decidimos usar una metodología. En la literatura se proponen diferentes metodologías, por ejemplo, en [9] se utiliza una representación basada en lógica de predicados de primer orden. Otros enfoques se basan en "frames" como Ontolingua [8], que es uno de los lenguajes ontológicos más utilizados.

Después de un estudio de las diferentes metodologías se optó por utilizar la metodología REFSENO (Representation Formalism for Software Engineering Ontologies) [21] debido a los siguientes motivos:

1. Como el mismo nombre indica, fue especialmente diseñada para el desarrollo de ontologías en la ingeniería del software.
2. REFSENO hace una distinción entre los diferentes niveles de conocimiento: conceptual y específico del contexto, en cambio los enfoques citados anteriormente representan un nivel de abstracción más alto.
3. La metodología propone diferentes técnicas para revisar la consistencia de la ontología y, además, tiene métodos para controlar la consistencia de las instancias en un nivel de implementación, característica que otras metodologías no consideran.

El primer paso para implementar la ontología, según la metodología REFSENO, es definir un glosario de conceptos que contenga la descripción y propósito de los conceptos previamente representados en el diagrama de la sub-ontología. Cada concepto corresponde a una fila de la tabla y existe un glosario de conceptos por ontología y sub-ontología representada en la Figura 1. En este artículo se muestra, a modo de ejemplo, el glosario de conceptos de la sub-ontología de productos (Tabla 1).

Concepto	Super-Concepto	Descripción	Propósito
Artefacto	Elemento	Parte diferenciada de un producto software que es creada o modificada por las actividades. Puede ser un documento (textual o gráfico), un componenete COTS, o un módulo de código. Ejemplos: documento de especificación de requisitos, plan de calidad, módulo de clase, rutina, informe de pruebas, manual de usuario. Sinónimos: elemento software, producto de trabajo, item de producto.	Definir la estructura interna y composición del software.
Producto	Concepto	Aplicación software que está siendo mantenida. Es un conglomerado de diversos artefactos. Sinónimo: software.	Mantenerlo.
Versión	Concepto	Un cambio en la línea base de un producto. Puede ser una versión completa, *upgrade*, *release* o actualización, o un simple parche en el código.	Implantar el proceso de gestión de configuración

Tabla 1. Subontología de los productos: Glosario de conceptos

El segundo paso es construir una tabla de atributos terminales para cada concepto definido en la tabla de glosarios. En la Tabla 2 se muestra la tabla del concepto Producto, las tablas de Artefacto y Versión se han omitido por razones de espacio.

Los atributos tienen un tipo determinado que bien puede ser uno de los que REFSENO propone o uno que el usuario define. En la Tabla 2 se muestran algunos de los tipos que hemos definido, por ejemplo "MaturityType", el cual define un rango de tipos de madurez que un producto software puede tener.

El valor del peso estándar (última columna de la tabla) representa el valor utilizado por las funciones de similitud (explicadas mas adelante).

REFSENO distingue tres tipos de capas: artefacto, interfaz y contexto. Los atributos pueden pertenecer a cualquiera de estas capas. Los atributos de la capa artefacto (denotados en la tabla con la palabra artefacto) describen las características de la instancia, los atributos de la capa de interfaz (denotado con I/F) caracterizan cómo una instancia en particular puede ser integrada en el sistema, y por último los atributos de la capa de contexto definen el entorno en el cual la instancia se va a usar.

Nombre	Descripción	Cardi-nalidad	Tipo	Obligatorio	Peso
Madurez (artefacto)	Etapa del ciclo de vida del producto	1	MaturityType	Si	1
Tamaño (artefacto)	Medida cualitativa del tamaño.	1	SizeMeasure	Si	1
Composición (artefacto)	Nivel de abstracción de los artefactos que lo forman.	1	CompositionType	Si	1
Tipo de aplicación (artefacto)	Tipo de aplicación. Por ejemplo: de gestión, científica, de control, etc.	1	ApplicationType	Si	1
Calidad (artefacto)	Medida cualitativa de la calidad, especialmente de la documentación.	1	MeasureQ	Si	1
Edad (artefacto)	Número de años desde que se obtuvo la primera versión.	1	Integer	Si	1
Componente (I/F)	Artefactos en los que se divide el producto.	0..*	Has-decomposition [artefacto]	No	1

Tabla 2. Subontología de los productos: Tabla de atributos

En la Tabla 2 solamente hay un atributo de la capa de interfaz el cual indica en qué artefactos se puede descomponer el producto. Obviamente, si el producto tiene una relación "Has-descomposition" con el concepto "artefacto", este componente debe tener la relación opuesta "Descomposition-of" indicando el producto del cual el artefacto forma parte.

3. Reutilización del Conocimiento

La información de un sistema de gestión del conocimiento debe ser reutilizada, en caso contrario no se obtendría todo el beneficio que se espera de dicho sistema. Para potenciar la reutilización de la información almacenada en nuestro sistema utilizamos Razonamiento Basado en Casos (CBR), técnica ampliamente utilizada con el fin de encontrar la mejor solución a un problema entre un grupo posible de soluciones [12].

Por otro lado, el hecho de que REFSENO utilice funciones de similitud ha facilitado el uso de CBR en nuestra aplicación.

A continuación se describe con un ejemplo cómo el sistema utiliza las funciones de similitud. En este caso se pretende comparar dos instancias de productos: i y q. El primer paso es calcular las funciones de similitud para cada capa, en el caso del Producto, como se aprecia en la Tabla 2, sólo hay dos capas artefacto e I/F:

$$Sim(product)(i,q) = W_{artif} * \mathbf{sim_{artif}(product)\ (i, q)} + W_{I/F} * \mathbf{sim_{I/F}(product)\ (i, q)}$$

Las funciones de similitud locales se calculan mediante la suma de las funciones de similitud de cada tipo de atributo perteneciente a esta capa. Finalmente, cada función de similitud local es normalizada resultando un valor en el rango [0,1]. Por ejemplo, en el caso de la capa artefacto del concepto producto es necesario conocer las funciones de similitud de los tipos "MaturityType", "SizeMeasure", "CompositionType", "AppplicationType", "MeasureQ" e "Integer" para poder calcular la suma de todas ellas.

REFSENO proporciona varios tipos de datos predefinidos y sus respectivas funciones de similitud. Por ejemplo, el tipo "Integer" tiene la siguiente función de similitud para comparar las instancias i y q:

$$Sim(i,q) = 1 - \frac{|i-q|}{(\max value - \min value)}$$

donde *minvalue* y *maxvalue* representan el menor y mayor valor correspondiente al rango de valores.

Los tipos de similitud para aquellos tipos definidos por el usuario, tales como "MaturityType", también deben ser descritos. Por ejemplo, "MaturityType" es una taxonomía formada por 4 etiquetas: inicial, evolución, servicio y retirada y su función de similitud es la siguiente:

Sim(i,q):	1	si i=q
	0.5	si i = inicial & q = evolución o viceversa
	0.25	si i = inicial & q = servicio o viceversa
	0	si i = inicial & q = retirada o viceversa
	0.5	si i = evolución & q = servicio o viceversa
	0	si i = evolución & q= retirada o viceversa

Después de calcular las funciones de similitud locales $\mathbf{sim_{artif}(product)(i, q)}$ y $\mathbf{sim_{I/F}(product)\ (i, q)}$ se deben calcular las similitud globales asignando valores a W_{artif}, and $W_{I/F}$, dependiendo de las necesidades del usuario. Por ejemplo si el sistema quiere comparar la similitud entre dos productos de acuerdo a sus propias características, el valor de W_{artif} deberá ser maximizado y $W_{I/F}$ minimizado ya que la suma de sus pesos debe ser siempre 1. De esta forma el sistema adapta los pesos según las prioridades de los usuarios.

El sistema usa principalmente las funciones de similitud para comparar productos software, y peticiones de mantenimiento. El objetivo principal de comparar productos es predecir futuras demandas de mantenimiento, ya que productos con características similares demandan muy a menudo el mismo tipo de modificaciones. Como se explica en [3], si los cambios pueden ser anticipados estos podrán realizarse de una forma más planificada y de esta manera su costo y esfuerzo disminuirán.

La finalidad de comparar las peticiones de mantenimiento es la reutilización de soluciones para problemas similares y así evitar repetir los mismos errores.

4. Conclusiones y Trabajos Futuros

El proceso de mantenimiento de software genera gran cantidad de conocimiento que debe ser procesado y gestionado con el fin de disminuir costes y esfuerzo. Sin embargo, antes de gestionarlo los diferentes tipos de información y sus relaciones deben ser especificados. Para ello hemos desarrollado una ontología que define los conceptos vinculados al mantenimiento del software y sus relaciones. Un factor a destacar de la ontología es su aspecto práctico, ya que ha sido utilizada en varios proyectos que nos han permitido detectar algunas de sus limitaciones y depurarla progresivamente. Por ejemplo, la ontología de flujos de trabajo surgió a raíz de analizar los aspectos dinámicos de un proyecto de mantenimiento.

Otra aportación importante de este artículo es la explicación de cómo llevar a cabo la implementación de la ontología puesto que aunque actualmente están surgiendo cada vez más trabajos que describen diseños ontológicos todavía muy pocos explican cómo realizar la implementación de la misma en un sistema. Por último, se ha ilustrado cómo se usan las funciones de similitud para potenciar la reutilización de la información, objetivo prioritario del sistema que se ha desarrollado para ayudar a los ingenieros del mantenimiento a realizar sus tareas.

Con el fin de mejorar la herramienta se hizo una encuesta a varios encargados del mantenimiento en la cual se les consultaba sobre qué funcionalidades añadirían al sistema. Casi todos los encargados contestaron que les gustaría que la aplicación tuviera un mecanismo que les indicará dónde encontrar la información necesaria para realizar una tarea. Actualmente se está trabajando en este objetivo y una vez terminada, la herramienta será evaluada por los propios ingenieros del mantenimiento.

Agradecimientos

Este trabajo es financiado por el proyecto MAS (TIC2003-02737-C02-02, Ministerio de Ciencia y Tecnología de España) el proyecto MESSENGER (TIC2003-07804-C05-03) y el ENIGMAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, referencia PBI-05-058).

Referencias

[1] Banker, R.D., Datar, S.M. and Kemerer, C.F., *A model to evaluate variables impacting the productivity of Software Maintenance Project.* Management Science, 1991. **37**(1) pp. 1-18.

[2] Banker, R.D., Datar, S.M., Kemerer, C.F. and Zweig D., *Software Errors and Software Maintenance Management*. Journal of Information Technology & Management, ed. N. Pirkul, and Vargheses S.J. Vol. 3. 2002. Kluwer Academic Publichers. pp. 25-41.
[3] Bennett, K.H., Rajlich, V.T., *Software Maintenance and Evolution: a Roadmap*, in *The Future of Software Engineering. Proc. ICSE 2000*. 2000: Limerick, Ireland. pp. 75-87.
[4] Card, D.N., Glass, R.L., *Measuring Software Design Quality*. 1990, Englewood Cliffs, USA. Prentice Hall.
[5] de Looff, L.A., *Information Systems Outsourcing Decision Making: a Managerial Approach*. 1997, PA: Idea Group Publishing: Hersey.
[6] Deridder, D. *A Concept-Oriented Approach to Support Software Maintenance and Reuse activities*. In Proceedings of the 5th Joint Conference on Knowledge Based Software Engineering (JCKBSE2002). 2002 IOS Press.
[7] Falbo, R.A., Menezes, C.S., Rocha, A.R., *Using Knowledge Serves to Promote Knowledge Integration in Software Engineering Environments*. Proceedings 11th International Conference on Software Engineering and Knowledge Engineering (SEKE' 99), 1999. June pp. 170-175.
[8] Gruber, T.R., *A Translation Approach to Portable Ontology Specification*. Knowledge Adquisition, 1993. **5**(2) pp. 199-220.
[9] Hikita, T., Matsumoto, M.J., *Business Process Modelling Based on the Ontology and First-Order Logic*. Proceedings Third International Conference on Enterprise Information Systems (ICEIS' 2001), 2001. 7-10 July pp. 717-723.
[10] Kitchenham, B.A., Travassos, G.H., Mayrhauser, A., Niessink, F., Schneidewind, N.F., Singer, J., *Towards an Ontology of Software Maintenance*. Journal of Software Maintenance: Research and Practice, 1999. **11** pp. 365-389.
[11] Maedche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R., *Ontologies for Enterprise Knowledge Management*. IEEE Intelligent Systems, 2003 pp. 26-33.
[12] Niknafs, A., Shiri, M., Javidi, M. *A Case-Based Reasoning Approach in E-Tourism: Tour Itinery Planning*. In Proc. 4th International Workshop on Theory and Applications of Knowledge Management. During DEXA'03. 2003. Prague, Czech Republic. pp. 818-822.
[13] Nonaka, I.a.T., H., *The Knowledge Creation Company: How Japanese Companies Create the Dynamics of Innovation*. 1995. Oxford University Press.
[14] Oliveira, K.M., Anquetil, N., Dias, M.G., Ramal, M., Meneses, R., *Knowledge for Software Maintenance*. Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE '03), 2003. 1-3 July pp. 61-68.
[15] Pigoski, T., *Practical Software Maintenance*. John Wiley & Sons. 1997.
[16] Polo, M., Piattini, M., Ruíz, F. and Calero, C. *MANTEMA: a Complete Rigorous Methodology for Supporting Maintenance based on The ISO/IEC 12207 Standard*. In Proceedings of the Third European Conference on Software Maintenance and Reengineering. 1999. Amsterdam.
[17] Ruíz, F., García, M., Piattini, M., Polo, M., *Environment for Managing Software Maintenance Projects*. In Advances in Software Maintenance Management: TEchnologies and Solutions. 2002, USA. Idea Group Publishing. pp. 255-290.
[18] Ruíz, F., Vizcaíno, A., Piattini, M., García, F., *An Ontology for the Management of Software Maintenance Projects*. International Journal on Software Engineering and Knowledge Engineering, 2004. **14**(3). June pp. 323-346.
[19] Rus, I., Lindvall, M., *Knowledge Management in Software Engineering*. IEEE Software, 2002. **19** pp. 26-38.
[20] Sousa, K.D., Anquetil, N., Oliveira, K.M., *Learning Software Maintenance Organizations*. Proc. Advances in Learning Software Organizations (LSO) LNCS 3096, 2004. June pp. 20-21.
[21] Tautz, C.G., Von Wangenheim, *REFSENO: A Representation Formalism for software Engineering Ontologies*, in *Fraunhofer IESE-Report*. 1998.

Clasificación Temprana de Fallos en Sistemas Dinámicos Usando Razonamiento Basado en Casos

Aníbal Bregón[1], Mª Aránzazu Simón[1], Juan José Rodríguez[2], Carlos Alonso[1], Belarmino Pulido[1], Isaac Moro[1]

[1] Grupo de Sistema Inteligentes(GSI), Departamento de Informática, E.T.S.I. Informática, Universidad de Valladolid, Campus Miguel Delibes s/n, 47011 Valladolid, España
anibbre@lab.fi.uva.es, {arancha,calonso,belar,isaac}@infor.uva.es
[2] Lenguajes y Sistemas Informáticos, Universidad de Burgos, Burgos, España
jjrodriguez@ubu.es

Resumen En este trabajo se presenta un sistema para la clasificación temprana de series temporales. Estas series representan distintos modos de fallo obtenidos por medio de una planta de laboratorio. La clasificación temprana de fallos es fundamental en sistemas de supervisión o diagnosis que generan datos en línea, ya que en caso de producirse un fallo, es necesario activar la señal de alarma cuanto antes. Presentamos un marco computacional para resolver estos problemas de clasificación temprana usando Razonamiento Basado en Casos. Este artículo ilustra diferentes técnicas para la reutilización y recuperación de casos, aplicadas en distintos instantes de la ocurrencia de los fallos, evaluando y comparando los resultados.

1. Introducción

El Razonamiento Basado en Casos (CBR) es un paradigma de Inteligencia Artificial en continua expansión y que, debido a su probada eficacia, está siendo utilizado en diversos campos como el diagnóstico. Se han desarrollado varios trabajos en los que se ha aplicado CBR para determinar el estado de operación de una planta industrial [5,8], y para llevar a cabo planificación, diagnóstico, mantenimiento y gestión de calidad en la industria [7].

En este trabajo se va a presentar el sistema CBR desarrollado para la clasificación de fallos en una planta industrial. Los datos a partir de los cuales se identifican los fallos, son series temporales, que se corresponden con valores históricos de las variables observables en el sistema. Estas series van a ser obtenidas en paralelo con un sistema de diagnóstico basado en modelos [12]. Debido a que es necesario identificar tan pronto como sea posible el fallo presente en el sistema y que las dinámicas de los distintos fallos no tienen la misma velocidad, se ha incluido en el sistema CBR la capacidad de clasificar series de distintas longitudes.