RISI **2**(2), 32-39 (2005)
Rev. investig. sist. inform.
Facultad de Ingeniería de Sistemas e Informática
Universidad Nacional Mayor de San Marcos
ISSN: 1815-0268 (versión impresa)/ ISSN: 1816-3823 (versión electrónica)

GESTIÓN DE INCONSISTENCIA EN ENTORNO MULTIPERSPECTIVA PARA EL DESARROLLO DE SOFTWARE

INCONSISTENCY HANDLING IN MULTI-PERSPECTIVE FOR SOFTWARE DEVELOPMENT

Giovana Valverde, Marcos Rivas *

RESUMEN El desarrollo de sistemas de software involucra inevitablemente el descubrimiento y el manejo de inconsistencias. Estas inconsistencias pueden aparecer en los requisitos del sistema, en la especificación de desarrollo y, más frecuentemente, en las descripciones para la implementación final del software. Gran parte de los investigadores de ingeniería de software se han dedicado al mantenimiento de consistencia, o a erradicar las inconsistencias tan pronto son detectadas. Los desarrolladores del software, por otro lado, viven con la inconsistencia como una forma natural, dependiendo de la naturaleza de una inconsistencia, sus causas y su impacto, ellos a veces escogen tolerar su presencia, en lugar de resolverlas inmediatamente, en forma absoluta.

En este trabajo presentaremos el estado del arte en el tema de inconsistencia en entorno multiperspectiva en el desarrollo de software, resumiendo las principales aportaciones, comparando las diferentes propuestas y destacando los principales problemas no resueltos en la actualidad.

Palabras clave: Ingeniería de Requerimiento, Especificaciones en Software, Administración de Inconsistencia, Lógica Multivalorada.

ABSTRACT The development of software systems inevitably involves the detection and handling of inconsistencies. These inconsistencies can arise in system requirements, design specifications and, quite often, in the descriptions that form the final implemented software product. A large proportion of software engineering research has been devoted to consistency maintenance, or geared towards eradicating inconsistencies as soon as they are detected. Software practitioners, on the other hand, live with inconsistency as a matter of course. Depending on the nature of an inconsistency, its causes and its impact, they sometimes choose to tolerate its presence, rather than resolve it immediately, if at all.

This paper presents the state of art in inconsistency handling.

Keywords: software specification, requirements engineering, inconsistency management, Multi-Valued Logics.

1. INTRODUCCIÓN

Los ingenieros de requisitos viven en un mundo donde las inconsistencias son la regla, no la excepción. Las inconsistencias generalmente se originan desde diferentes puntos de vista e intereses. Ellos deben ser detectados y eventualmente resueltos aun cuando puedan ser de utilidad para elicitar información adicional, su resolución en alguna fase u otro del proceso es una condición necesaria para el desarrollo exitoso del software. Los principales problemas que hoy

* Facultad de Ingeniería de Sistemas e Informática – Universidad Nacional Mayor de San Marcos – Lima – Perú.
E-mail: gmvalverde@yahoo.com, mrivasp@unmsm.edu.pe

experimenta la IR en el manejo de inconsistencias son abordados bajo un razonamiento formal [2].

La ingeniería de software se le conoce como una disciplina de descripción donde el problema de inconsistencia lleva a equivocaciones y errores. Sin embargo, el problema está, con las inconsistencias que no han sido detectadas. En muchos casos, nosotros podemos desear tolerar una inconsistencia conocida. Balzer [1] introduce la noción de *pollution markers* para marcar porciones de código del programa que contiene inconsistencias que pueden tolerarse y así continuar con el desarrollo.

Las descripciones en ingeniería de software pueden traducirse a la lógica clásica, pero esta no permite un razonamiento útil en la presencia de inconsistencias.

En [3] se presenta un enfoque formal que soporta acciones continuas en la presencia de inconsistencias y facilita el registro y seguimiento de información durante el razonamiento, y en [4, 6] se presenta una comparación de la lógica de tres valores con la lógica de cuatro valores y muestra que esta última se puede usar para el manejo de inconsistencias.

Existen framework basados en múltiples puntos de vista que apoyan el descubrimiento y resolución de inconsistencias, en [5] se presenta un framework para clasificar y razonar sobre los múltiples viewpoints .

2. ESTADO DEL ARTE

2.1. ¿Qué es inconsistencia?

Nosotros usamos el término inconsistencia para denotar cualquier situación en que dos descripciones no obedecen a ninguna relación que se prescribe debe existir entre ellos [9]. Una precondición para la inconsistencia es que las descripciones en cuestión tengan algún área de traslapo [10]. Una relación entre las descripciones puede expresarse como regla de consistencia, con la cual las descripciones pueden verificarse.

2.2. Tolerando inconsistencia

Existen momentos durante el desarrollo de software que tolerar la inconsistencia puede dar ciertos beneficios como [8]: (1) Indicar que existen desviaciones del modelo de procesos, (2) Facilitar un funcionamiento de colaboración flexible y (3) Puede usarse para identificar áreas de incertidumbre.

Los sistemas de software son raramente consistentes y las modificaciones son constantes, estas se propagan lentamente a través del sistema. Similares excepciones surgen con el manejo de datos cuando se usan aplicaciones de software. Balzer propone relajar las restricciones de consistencia durante el desarrollo [1]. El enfoque sugiere que los datos incoherentes se marquen por guardias, *(Pollution Markers),* eso tiene dos usos: (a) Identificar los datos incoherentes para codificar segmentos o identificar agentes humanos que puedan ayudar a resolver las inconsistencias, y (b) Para proteger los datos incoherentes de otros segmentos que son sensibles a las inconsistencias.

El enfoque se basa primeramente en que la inconsistencia debe ser detectada y se deberá localizar los datos que participan en la violación de la restricción para que el resto de la base de datos pueda ser procesada normalmente. Esto se logra cambiando la restricción, en lugar de prohibir alguna condición Q, se introduce una nueva condición P y requiere que Q y P ocurran simultáneamente. La nueva condición es un *pollution markers* que es una relación cuyos parámetros llevan variables cuantificadas de la condición Q, y que indica la violación de una restricción específica. Al incorporar variables cuantificadas de la condición Q el *pollution markers* identifica los datos que violan los criterios originales de consistencia.

2.3. Gestión de inconsistencia en ingeniería de requisitos basado en objetivos.

Un buen número de inconsistencias puede aparecer durante la ingeniería de requisitos basado en objetivos, los requerimientos son elicitados desde múltiples *stakeholders.* Resolver las inconsistencias lo más pronto posible en el proceso es una condición necesaria para el desarrollo exitoso del software que incluye esos requisitos.

El estado del arte en gestión de inconsistencia para ingeniería de requisitos experimenta varios problemas: (a) El tipo específico de inconsistencia que es considerada no siempre está claro. No hay ningún acuerdo referente a lo que un conflicto entre los requisitos realmente significa. (b) No hay un soporte sistemático para detectar los conflictos entre las especificaciones de operación, de objetivos o requisitos y (c) Hay una falta de técnicas para resolverse las inconsistencias a través de las transformaciones del objetivo/requerimiento.

Los tres problemas son abordados desde un razonamiento formal en [2], se revisa varios tipos de inconsistencia frecuentemente encontrados en IR. La Figura 1 introduce varios niveles donde pueden ocurrir inconsistencias relacionadas con los requisitos.

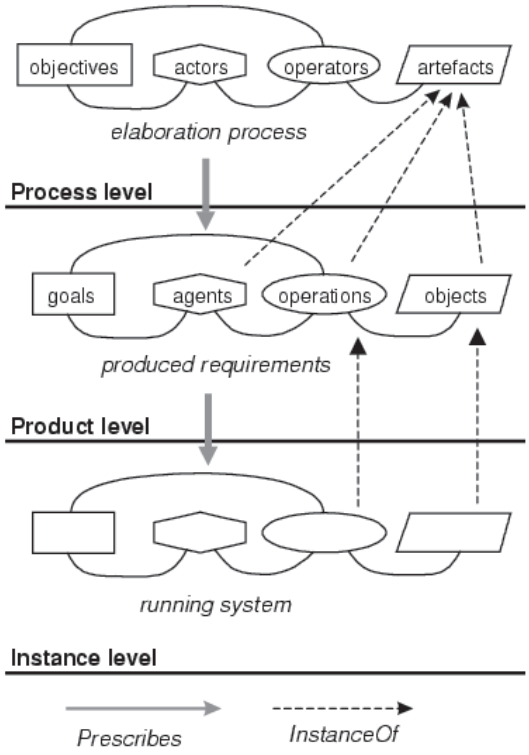

Figura 1. Niveles de inconsistencias.

Clasificación de inconsistencias

Un conjunto de descripciones es inconsistente si no hay una forma de satisfacer a todas las descripciones juntas, dependiendo de las descripciones que se capturen se podrían identificar los siguientes tipos de inconsistencias: (a) *Desviación en el nivel de proceso*. Es un estado transitorio en el proceso de IR que resulta de una inconsistencia entre una regla en el nivel de proceso de la forma, (∀ r: Requerimiento) R(r) y un estado de proceso específico caracterizado por ¬ R(req), para un requerimiento req en el nivel de producto. (b) *Desviación en el nivel de instancia.* Es un estado transitorio en la ejecución del sistema que resulta de una inconsistencia entre un requerimiento en el nivel de producto de la forma ,(∀x:X) R(x) y un estado específico de la ejecución del sistema caracterizado por ¬ R(a), para algún valor específico a de X. (c) *Conflicto de terminología.* Ocurre cuando un simple concepto del sistema real es nombrado con una sintaxis distinta al especificar los requerimientos. Este tipo de inconsistencia podría encontrarse entre los diferentes puntos de vista de los *stakeholders.* (d) *Conflicto de interpretación.* Ocurre cuando un simple nombre en la especificación de requerimientos puede ser interpretado de diferentes maneras en el sistema real. (f) *Estructura de conflictos.* Ocurre cuando para un simple concepto del sistema real se le da diferentes estructuras en la especificación de requerimientos. (g) *Conflictos.* Si A_1, A_2, A_3, ..., An representan objetivos, requerimientos o suposiciones y Dom un dominio de descripciones, entonces un conflicto entre A_1, A_2, A_3, ..., An ocurre dentro del dominio Dom sólo si las siguientes condiciones se llevan a cabo: (1) Los estados de A_1, A_2, A_3, ..., An son lógicamente inconsistentes en el dominio. (2) Los estados que son quitados de algún A_1, A_2, A_3, ..., An no tardan en resultar inconsistentes dentro del dominio Dom. (h) *Divergencia*. Una divergencia entre A_1, A_2, A_3, ..., An ocurre dentro del dominio Dom, sólo si existe una condición de límite B tal que las siguientes condiciones se llevan a cabo: (1) La condición de límite captura una combinación particular de circunstancias que hace que A_1, A_2, A_3, ..., An genere conflicto si se juntan con ellos. (2) En los estados que la condición de límite es satisfecha por lo menos a través de un comportamiento debería haber por lo menos un escenario que establezca la condición del límite. (i) *Competencia.* Es un caso particular de divergencia dentro de un único objetivo / requerimiento. (j) *Obstrucción.* Es un caso límite de divergencia donde solo un elemento de {A_1, A_2, A_3, ..., A_n} es involucrado.

Dos técnicas formales que proponen para detectar divergencia [2]. La primera técnica (Regressing Negated Assertions), es basada en una variante común de la primera condición que caracteriza la divergencia. La segunda (Using Divergence Paterns), indica que al lado de la regresión reiterativa a través de las especificaciones de objetivos y de descripciones del dominio, se puede identificar modelos de divergencia frecuentes que pueden establecerse formalmente por última vez junto con sus condiciones de límite asociadas.

La heurística para identificar divergencia que proporcionan como reglas para ayudar a los usuarios a identificar las divergencias sin necesidad de tener que pasar por las técnicas formales [2]. Las divergencias identificadas necesitan ser resueltas de alguna manera y depende de la probabilidad de ocurrencias de las condiciones de límite y de las consecuencias del conflicto en cada caso. Varias estrategias pueden seguirse para resolver divergencias, cada una de ellas corresponden a una clase específica de solución de operador definido en el proceso de nivel (Figura 1). Una de ellas agrupa todos los operadores que crean, modifican y eliminan objetivos y otra agrupan a todos los operadores que crean, modifican y eliminan tipos de objetos.

Gestión de inconsistencia en especificaciones

Existen trabajos que usan la lógica clásica para representar parcialmente las especificaciones y para

identificar inconsistencias entre ellas. En [3] se presenta una adaptación de la lógica clásica llamada *quasi-classical* (QC) *logic,* para continuar con el razonamiento en presencia de inconsistencias. El enfoque es necesario porque necesitamos tratar con las especificaciones inconsistentes de tal manera que nos permita analizar las probables fuentes de las inconsistencias, y podramos continuar con el razonamiento en una forma racional en la presencia de inconsistencia y proveer una base de actuación sobre las inconsistencias.

Captura de información de desarrollo en fórmulas lógicas

Una inconsistencia en lógica es el resultado de las aserciones simultáneas de un hecho a y su negación $\neg\alpha$. Usando esta definición se traslada las especificaciones de ingeniería de software dentro de fórmulas lógicas facilitando las detección de inconsistencias y permitiendo concentrarse en el razonamiento de estas inconsistencias.

Aparte de la lógica clásica existen muchas formas de representar alguna especificación dada, más bien existen maneras obvias de representar cualquier especificación como un juego de fórmulas lógicas. Por ejemplo consideremos una descripción entidad-relación de un banco (que está basado en un conjunto de nodos, denotando entidades y arcos binarios que evidencian relaciones entre entidades).

Nosotros podemos capturar fácilmente un esquema de representación que se ilustra con las siguientes fórmulas.

Posee (Banco, Cuenta) Tiene (Cliente, cuenta)

Hereda (Cajero-Transacción, Transacción)

En la lógica los símbolos banco, cuenta, cliente, cuenta, cajero-transacción y transacción son, posee, tiene y herencia, todas ellas son relaciones en la lógica. Adicionalmente podemos asumir fórmulas que capturan la naturaleza especial de ciertas relaciones, tales como:

$\forall$ X, Y, Z hereda (X, Y) $\wedge$ hereda (Y, Z) $\rightarrow$ hereda (X, Z)

Similarmente podemos escribir relaciones entre esquemas de representación o especificaciones parciales. Por ejemplo considere la siguiente regla para asegurar la consistencia de uso de la sintaxis para multiplicidad de asociaciones entre dos clases en un modelo de objeto.

Asumamos la relación tiene-exactamente-uno y tiene-exactamente-dos entre un par de clases, por ejemplo, tiene-exactamente-uno (Hijo, Mamá), para esto tenemos la siguiente regla:

$\forall$ X, Y, Z tiene-exactamente-uno (X, Y) $\leftrightarrow \neg$ tiene-exactamente-dos (X, Z)

La restricción también puede ser útil para la información de especificaciones o información de dominio.

Razonamiento en la presencia de inconsistencia

Aquí se enfoca el problema de razonamiento con información que podría ser inconsistente. La lógica clásica es aparente para las especificaciones, pero presenta problemas en el razonamiento con inconsistencias sobre todo cuando hay múltiples requerimientos contradictorios acerca de un sistema y el uso de la lógica clásica es trivializada, porque, por definición varias inferencias se pueden realizar a partir de información inconsistentes. Una alternativa para resolver estos problemas es la QC logic.

La prueba teórica de QC logic está basada en razonamiento con fórmulas que son un conjunto de formas normales (CNF). Estas fórmulas son de la siguiente forma, $\alpha_1 \wedge \ldots \wedge \alpha_n$, donde cada α_i es de la forma, $\beta_1 \wedge \ldots \wedge \beta_m$ y cada β_i es un literal.

La prueba teórica de QC logic provee el poder para derivar un CNF de alguna fórmula junto con el poder de solución:

$$\frac{\neg\alpha \quad \alpha \vee \beta}{\beta}$$

Sólo como un último paso en alguna derivación es permitida introducir una disyunción. Esto significa que alguna solución de un conjunto de fórmulas puede ser derivada, pero ninguna trivial. Esta prueba teórica es presentada como un conjunto de deducciones naturales, tal como:

$$\frac{\neg(\alpha \vee \beta)}{\neg\alpha \wedge \neg\beta)}$$

Para ilustrar el uso de QC en situaciones de desarrollo multiperspectiva, veamos el siguiente ejemplo:

Supongamos que tenemos dos especificaciones parciales VP1 Y VP2.

VP1 tiene la asociación tiene-exactamente-uno (cajero, terminal).

VP2 tiene la asociación tiene-exactamente-dos (cajero, terminal).

Consideremos la restricción:

$\forall$ X, Y tiene-exactamente-uno (X, Y) $\leftrightarrow \neg$ tiene-exactamente-dos (X, Y)

Esta regla entre punto de vista junto con VP1 Y VP2 es inconsistente.

Sin embargo la definición de la relación tiene-exactamente-uno y tiene-exactamente-dos también implica la siguiente restricción:

$\forall$ X, Y tiene-exactamente-uno (X, Y) $\leftrightarrow$ tiene-exactamente-uno-o-más (X, Y)

$\forall$ X, Y tiene-exactamente-dos (X, Y) $\leftrightarrow$ tiene-exactamente-uno-o-más (X, Y)

Estas restricciones podrían ser adicionalmente útiles en el dominio de conocimiento, descubiertos quizás durante el desarrollo o como resultado de un acuerdo entre desarrolladores. Entonces para VP1 y VP2 y a pesar de la inconsistencia entre ellos, podemos derivar la inferencia no trivial potencialmente útil:

tiene-exactamente-uno-o-más (Cajero, Terminal)

La definición formal de QC logic se presenta en [3].

Para el razonamiento de inconsistencia existen otras herramientas que evitan la trivialización, llamada lógicas paraconsistentes:

El valor de los cuatro valores [4], introduce una estructura llamada FOUR con la finalidad de tratar de una forma conveniente las inconsistencias y la información incompleta. La estructura FOUR tiene cuatro valores de verdad, la clásica V y F, y dos adicionales: $\perp$ que intuitivamente denota falta de información y $\leq\approx$ que indica inconsistencias. Belanp argumenta que la manera que las computadoras deben de pensar, debería basarse en estos cuatros valores.

Análisis lógico de especificaciones inconsistentes

Hay dos formas de realizar el análisis lógico de especificaciones inconsistentes, calificando inferencia desde información inconsistente e identificando probables fuentes de una inconsistencia.

Para el primer caso veamos el siguiente ejemplo, consideremos la siguiente suposición:

{a}: $\alpha \wedge \beta$ {b}: $\neg\alpha \wedge \beta$ {c}: γ

Esto da como máximo dos subconjuntos consistentes:

Subconjunto 1	Subconjunto 2
{a}: $\alpha \wedge \beta$	{a}: $\neg\alpha \wedge \beta$
{c}: γ	{c}: γ

Entonces se tiene que a y $\neg\alpha$ son sólo inferencias existenciales, considerando que g es una inferencia libre y b una inferencia universal [3].

Estos tipos de calificaciones son útiles cuando razonamos con información inconsistente porque ellos proporcionan una relación clara e inequívoca entre las inferencias y los datos problemáticos.

En el segundo caso cuando identificamos una inconsistencia en nuestra información de desarrollo, queremos analizar las inconsistencias antes de decidir algún curso de acción. Por ejemplo, supongamos que tenemos las siguiente especificación:

{a}: α {b}: $\neg\alpha \wedge \neg\beta$ {c}: β

Y supongamos que {a}: α y {b}: $\neg\alpha \wedge\neg\beta$ han sido parte de especificaciones estables y bien aceptadas por un tiempo y por lo contrario {c}: β es justo una nueva y tentativa parte de especificación. Entonces para la inconsistencia {a, b, c}: $\perp$ podríamos considerar {c}: β como la fuente de la inconsistencia.

Actuando en la presencia de inconsistencia

Usando el reporte de inconsistencias identificadas podemos decidir sobre las posibles acciones a tomar en relación a las inconsistencias identificadas. Estas posibles acciones dependen del tipo de inconsistencia que fue detectada y el grado de tolerancia de las inconsistencias que pueden soportarse. Se identifica cuatro posibles tipos de acciones: (a) Ignorando completamente la inconsistencia y continuar con el desarrollo sin tomarlas en cuenta. (b) Maquillando la inconsistencia completamente en la especificación desarrollada y continuar con el desarrollo. (c) Eliminando la inconsistencia totalmente corrigiendo cualquier error o resolviendo los conflictos. (d) Mejorando las situaciones de inconsistencia desarrollando acciones que mejoren estas situaciones e incrementen la posibilidad de solución en el futuro.

Decidir qué tipo de acciones es apropiado o factible de desarrollar en la presencia de inconsistencia es el primer paso hacia una especificación efectiva del manejo de reglas de inconsistencias.

Framework para razonamiento Multivalor sobre inconsistencia en entorno multiperspectiva

En [5], se presenta χbel *(Multi-Valued Belief Exploration Logics)* herramienta para combinar y razonar acerca de múltiple viewpoint. La herramienta usa una familia de lógica multivalor, llamada Quasi-Bollean logics, para soportar razonamiento sobre mo-

delos inconsistentes. La lógica multivalor nos permite introducir un número de diferentes valores de verdad entre verdadero y falso, representando diferentes tipos de desacuerdos e indecisiones. La herramienta incluye también un modelo revisor multivalor χchek para razonar acerca de las propiedades temporales de la inconsistencia en el modelo.

Lógica multivalor

La lógica multivalor es útil para combinar información desde inconsistencia multiperspectiva porque nos permite explícitamente representar diferentes niveles de acuerdos. Por ejemplo, si nosotros guardamos los valores boleanos usuales verdadero y falso para el significado «Unánimemente Verdadero» y « Unánimemente Falso», entonces podemos agregar cualquier número de valores intermedios para representar diferentes tipos de desacuerdos. Por ejemplo podemos incluir los siguientes valores: «una mayoría dijo verdadero», «cuatro dijeron verdadero», «uno dijo falso», «todos dijimos es desconocido», «el experto designado dijo falso», «todos los demás dijimos verdadero», adicionando estos valores en la lógica podemos razonar sobre ciertos niveles de acuerdo para cualquier proposición arbitraria.

Quasi-Boolean Multi-Value Logics

Una lógica de valores finitos [4] puede describirse dando su axiomatización o manteniendo las tablas de conjunción, disyunción y operadores de la negación en los elementos de la lógica. Para asegurar este razonamiento se escoge la lógicas donde la axiomatización sea tan cerca a la lógica clásica como posible. Las propiedades útiles, como la asociativa, la distributiva, las leyes de Morgan, etc., deben llevarse a cabo, mientras que el estado incompleto y las leyes de inconsistencia, como la ley de medio excluido ($\alpha \vee \neg\alpha = \leq\approx$) y la ley de no contradicción ($\alpha \wedge \neg\alpha = \perp$), pueden desecharse.

Una revisión del framework

El framework está compuesto de los siguientes elementos: (a) Un modelo básico de viewpoint, o χview el cual es un modelo de máquina de estados extendidos para el caso de multi-valor. (b) Una interconexión primitiva para definir la combinación χview dentro de un modelo combinado. (c) Quasi-boolean logics para razonar sobre χview. (d) Un conjunto de plantillas de combinación que guían para escoger el multivalor lógico y las interconexiones para los diferentes tipos de análisis. (e) Un modelo revisor multivalor χchek para verificar las propiedades de las combinaciones χview.

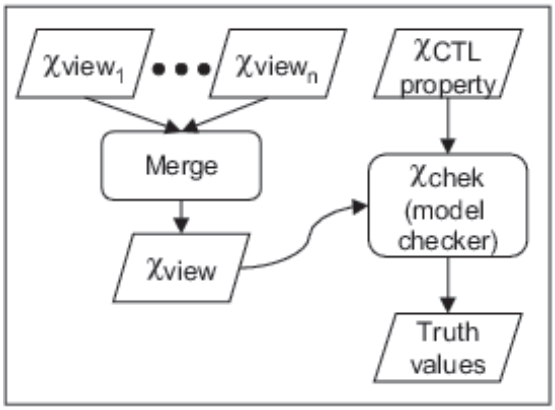

Figura 2. Modelo del proceso.

Todo el proceso es mostrado en la figura 2. El framework toma un conjunto de vistas de la fuente y las combina usando un conjunto de interconexiones primitivas y una plantilla de combinación escogida por el analista. El resultado unido en χview puede ser verificado por el modelo mediante un conjunto de propiedades expresado en χCTL, nuestra lógica temporal multivalor o una extensión de CTL *(Computational Tree Logic)*. El modelo verificador retorna varios valores desde la *lattice* que representa los valores que cada propiedad tiene en cada estado inicial.

Xlinkit: Un verificador de consistencia y servicio de generación inteligente de vínculos

Xlinkit, es una aplicación ligera que proporciona una generación de vínculos basado en reglas y verificación de consistencia de contenido en web distribuido. Existen demostraciones en línea en http://www.xlinkit.com.

Una descripción con un ejemplo se tiene en [7], además se presenta una nueva semántica de la vinculación para el lenguaje basado en lógica de primer orden, el cual retorna los hipervínculos entre los elementos inconsistentes en lugar de los valores boleanos.

Xlinkit está influenciada por la tecnología estándar de Internet, soporta distribución de documentos y puede soportar modelos de múltiples despliegues.

Puede usarse xlinkit para construir un sistema CRM (*Customer Relationship Management*) basado en web que permita navegar entre todos los segmentos de información que refleje los intereses de un solo cliente.

Podemos concluir que xlinkit es un producto de larga duración enfocado en la administración de consistencias.

3. COMPARACIÓN DE PROPUESTAS

Balzer [1] describe un enfoque para tolerar inconsistencia e introduce la noción de *pollution markers* para marcar porciones de código del programa que contiene inconsistencias, estas pueden engañarse para continuar el desarrollo. El enfoque no provee ningún mecanismo para especificar acciones que se necesitan realizar para ocuparse de estas inconsistencias, con respecto a las otras propuestas su alcance se da a nivel de programa.

La propuesta de A. Van Lamsweerde, R. Darimont and E. Letier en [2] está orientada a la administración de conflictos en las descripciones en ingeniería de requisitos orientada a Objetivo, nos facilita capturar puntos de vista, detectar, clasificar y resolver inconsistencias usando el lenguaje de la metodología KAOS. Su alcance se da a nivel de especificaciones de requisitos.

A. Hunter and B. A. Nuseibeh en [3] proponen una adaptación de la lógica clásica llamada quasi-classical (QC) logic para tolerar inconsistencia razonando, analizando y actuando en presencia de inconsistencia. Su alcance está a nivel de especificaciones en desarrollo de software.

S. M. Easterbrook and M. Chechik en [5] proponen un framework usando lógica multivalorada presentada en [4, 6] para razonar en presencia de inconsistencia en las especificaciones de requerimientos en ingeniería de requisitos.

C. Nentwich and L. Capra and W. Emmerich and A. Finkelstein en [5] proponen una aplicación para generar vínculos basados en reglas y verificar inconsistencia entre documentos en web distribuidos. Proponen una nueva semántica para generar vínculos a través del lenguaje de lógica de primer orden y para implementarlo usa XML.

Framework para administrar inconsistencias

Las diferentes propuestas pueden ser usadas a través de un framework para administrar inconsistencia propuesto en [8]. El centro a este framework es el uso explícito de un conjunto de reglas de consistencia que nos provee de una base para la mayoría de las actividades de administración de inconsistencia. Las reglas de consistencia se usan para monitorear la evolución de un conjunto de descripciones por causa de las inconsistencias. Cuando se descubren las inconsistencias, se debe realizar un diagnóstico, localizar e identificar su causa. Una de las diferentes estrategias para manejar la inconsistencia puede escogerse incluyendo la posibilidad de resolverlo inmediatamente, ignorándolo completamente, o tolerándolo durante algún tiempo. Cualquiera sea la acción escogida, el resultado es supervisado para las consecuencias indeseables.

4. CONCLUSIONES

La inconsistencia surge a lo largo del ciclo de vida de desarrollo de software, de acuerdo a cómo evolucionan los diferentes documentos de desarrollo. Aparecen debido a las interdependencias dentro de los documentos y entre los documentos.

El uso de lógica nos proporciona un lenguaje preciso e inequívoco en identificar las inconsistencias involucradas en las especificaciones multiperspectiva, también nos proporciona los medios para administrar inconsistencia de una manera genérica que es independiente de cualquier método de Ingeniería de Software.

Tabla 1. Resumen de los diferentes enfoques para administrar inconsistencias.

Enfoque	Mecanismo	Alcance
Tolerar inconsistencia [1]	Pollution markers	Programación
Administrar inconsistencia [2]	Metodología KAOS	Especificación de requerimientos en IR orientado a objetivos
Tolerar inconsistencia [3]	Quasi-classical (QC) logic	Especificación en desarrollo de software
Razonamiento en presencia de inconsistencia [5]	Lógica paraconsistente	Especificación de requerimiento en IR
Verificación de inconsistencia [7]	XML, nueva semántica para el lenguaje de lógica de primer orden	Documentos en web distribuidos

Resolver la inconsistencia puede ser tan simple como adicionar o eliminar información de una descripción del software. Sin embargo, se confía en resolver las inconsistencias tomando decisiones importantes como que la resolución inmediata no es la mejor opción y se puede elegir entre ignorarla, maquillarla, eliminarla o mejorarla.

Los trabajos futuros para desarrollar un framework en un ambiente de desarrollo de software en que la administración de inconsistencia se vuelva la actividad central a lo largo del ciclo de vida depende del desarrollo de software y que además incorpore el uso de lógica multivalorada.

Cuando se resuelve una inconsistencia es necesario propagarla a través del sistema, por lo tanto es necesario desarrollar técnicas de propagación de consistencia.

Si se toma alguna acción para tolerar la inconsistencia se debe considerar qué pasa si la inconsistencia no es resuelta.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] R. Balzer (1991); «Tolerating Inconsistency»; *Proceedings of 13th International Conference on Software Engineering* (ICSE-13), Austin, Texas, USA, 13-17th May 1991, 158-165; IEEE Computer Society Press.

[2] A. Van Lamsweerde, R. Darimont and E. Letier. *Managing Conflicts in Goal-Driven Requirements Engineering*. IEEE Transactions on Software Engineering, 1998.

[3] Dd A. Hunter and B. A. Nuseibeh. *Managing Inconsistent Specifications: Reasoning, Analysis and Action*. ACM Transactions on Software Engineering and Methodology. 4, 335-367. 1998.

[4] O. Arieli and A. Avron. *The Value of the Four Values.* Artificial Intelligence. 102, 97-141. 1998.

[5] S. M. Easterbrook and M. Chechik. «A Framework for Multi-Valued Reasoning over Inconsistents Viewpoints». *Proceedings, 23rd International Conference on Software Engineering* (ICSE-01),2001.

[6] S. Konieczny and P. Marquis. «Three-valued Logics for Inconsistency Handling». *Logics in Artificial Intelligence, European Conference,* (JELIA-02), 2002.

[7] C. Nentwich and L. Capra and W. Emmerich and A. Finkelstein. *xlinkit:* «A Consistency Checking and Smart Link Generation Service». ACM *Transactions on Internet Technology.* Vol. 2, (2), 151-185. 2002.

[8] B. Nuseibeh, S. Easterbrook and A. Russo. *Making Inconsistency Respetable in Software Development.* Expanded version of *Laveraging Inconsistency in Software Development*, computer, 33(4):24-29, IEEE Computer Society Press, April 2000.

[9] B. Nuseibeh, J. Kramer, and A. C. W. Finkelstein, «A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification», *Transactions on Software Engineering,* 20(10):760-773, IEEE Computer Society Press, October 1994.

[10] G. Spanoudakis, A. C. W. Finkelstein, and D. Till, «Overlaps in Requirements Engineering», *Automated Software Engineering*, 6(2):171-198, Kluwer, April 1999.