ARTÍCULO

Diagnosis applied CSP based on models

R. Ceballos, C. Del Valle, M. T. Gómez y R. M. Gasca

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática,
Avenida Reina Mercedes s/n 41012 Sevilla (Spain)

e-mail: { ceballosg, carmelo, mayte, gasca, }@lsi.us.es

In the last decades, model-based diagnosis has been an active research topic for the Artificial Intelligence community. It uses the explicit model of a system, the system inputs and the measured system outputs, in order to identify the subsystems that can generate faults. The system or the process that incorporates diagnosis may reduce costs and provide more security.

Some models used in engineering are based on constraint logic programming (CLP) in order to obtain the system diagnosis. In this paper we propose a methodology for the system diagnosis as a constraint satisfaction problem (CSP). Using this methodology it is possible to incorporate, the advances and optimizations achieved for the search of solutions in CSP. This methodology also offers the possibility of applying diagnosis to other areas, such as software diagnosis. Software diagnosis allows the identification of the program bugs. A bug occurs when there is not matching between the specified results and the observed results after a program execution.

Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No.20 (2003)
ISSN: 1137-3601. © AEPIA (http://www.aepia.org/revista) pp137-150.

CSP aplicados a la diagnosis basada en modelos

R. Ceballos, C. Del Valle, M. T. Gómez y R. M. Gasca

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática,
Avenida Reina Mercedes s/n 41012 Sevilla (Spain)

Resumen

En las últimas décadas, parte de la comunidad científica ha dedicado sus esfuerzos al desarrollo de una metodología para la diagnosis de sistemas desde el campo de la Inteligencia Artificial. Dicha metodología se denomina diagnosis basada en modelos, y cubre un amplio abanico de posibilidades. Se parte de un modelo explícito del sistema a diagnosticar y a partir de él se razona sobre la identificación de los subsistemas que generan fallos, utilizando para ello los valores de las entradas proporcionadas y las salidas captadas del sistema. En cualquier proceso de producción o desarrollo es importante tener un control sobre los fallos en componentes o procesos. La diagnosis permite controlar estas irregularidades, lo que conlleva a los sistemas que la incorporan una mayor seguridad y reducción de costos.

Algunos modelos utilizados en ingeniería se han basado en la programación lógica con restricciones (CLP) para obtener la diagnosis de un sistema. En este artículo proponemos la metodología necesaria para poder plantear la diagnosis de un sistema como un problema de satisfacción de restricciones (CSP). De esta forma, será posible incorporar al proceso de generación de la diagnosis de un sistema, los avances y optimizaciones que se han alcanzado en el campo de la búsqueda de soluciones para problemas CSP. Plantear un problema de diagnosis de esta forma abre también la posibilidad de aplicar la diagnosis a otros campos, como por ejemplo la diagnosis del software. La diagnosis del software permite identificar y localizar el origen de los errores de un desarrollo software. Un programa tendrá un error si no existe concordancia entre los resultados especificados como correctos y los resultados observados tras la ejecución.

Palabras clave: Diagnosis basada en modelos, satisfacción de restricciones, testing, diseño por contrato, diagnosis del software.

1. Introducción

Los fallos producidos en los componentes y procesos pueden provocar paradas indeseables y deterioro de los sistemas, con el consiguiente aumento de costos y la disminución de la producción. Para mantener los sistemas en los niveles deseados de seguridad, producción y fiabilidad es necesario desarrollar mecanismos que permitan la detección y diagnosis de los fallos que se producen en ellos.

La diagnosis determina por qué un sistema diseñado correctamente no funciona como se esperaba a partir de la monitorización de las entradas y salidas observables del sistema. Mediante el sistema de monitorización disponemos de una representación fiel del sistema y de las desviaciones que en él se producen. La diagnosis es un campo muy activo de investigación, en este artículo nos centraremos en la diagnosis basada en modelos. Existen dos comunidades que trabajan en paralelo y de forma normalmente aislada en este campo: la metodología FDI (Fault De-

tection and Isolation) proveniente del campo del control automático, y la metodología DX emergida del campo de la Inteligencia Artificial. Últimamente hay una apuesta por la integración de ambas. La integración de las teorías de FDI con DX y las pruebas de sus equivalencias se han mostrado para varios supuestos en [6][12]. Este artículo se centrará en la metodología DX usando modelos cuantitativos.

En la metodología de localización y detección de fallos, FDI se usan relaciones dentro del modelo de comportamiento [20][18]. La metodología FDI permite un análisis off-line de parte del trabajo, frente a la metodología DX donde el trabajo es casi totalmente on-line. En [11] se propone una aproximación a la metodología FDI pero utilizando un modelo CLP para la resolución de la diagnosis.

En el área de la diagnosis basada en modelos el trabajo pionero en este campo se presentó con el objeto de diagnosticar sistemas de componentes basándose en la estructura y su comportamiento [7]. Las primeras implementaciones para diagnosis fueron DART[14] y GDE[9], que usaban diferentes componentes de inferencia para detectar los posibles fallos.

La formalización de la diagnosis se concreto en [21] y [8], donde se propone una teoría general para el problema de explicar las discrepancias entre los comportamientos observados y correctos de los componentes. Los modelos centrados en componentes describen los sistemas mediante relaciones input-output. La mayoría de las aproximaciones para componentes caracterizan la diagnosis de un sistema como una colección de conjuntos mínimos de componentes que fallan, para explicar los comportamientos observados (síntomas). De ahí la importancia de disponer de un buen modelo para determinar la diagnosis de un sistema. Con este tipo de modelos es posible diagnosticar rápidamente partes importantes de los sistemas de componentes.

La diagnosis basada en modelos examina los modelos para determinar la causa de fallo. Muchos métodos de diagnóstico usan modelos centrados en componentes que requieren a los ingenieros desarrollar modelos de fallos así como el modelo de operación normal. Construir modelos de fallos en un sistema es útil cuando los fallos son bien conocidos y fáciles de modelar, pero ello limita el sistema de diagnosis a fallos conocidos. Una revisión de las aproximaciones acerca de la automatización de las tareas de diagnosis se puede encontrar en [10] y para una discusión de las aplicaciones de la diagnosis basada en modelos se puede consultar [5]. La generalización de la diagnosis basada en la consistencia se ha propuesto [16] para cubrir sistemas que contienen procesos y que cambian su estructura dinámicamente.

En [12] se mejora y automatiza el proceso de diagnosis en sistemas cuyos modelos se componen de restricciones de igualdad polinómica. Para ello se utilizan algoritmos de procesamiento simbólico del modelo inicial (bases de Gröbner), que generan los posibles conflictos del modelo de acuerdo a su estructura y comportamiento. Se consigue una reducción y mejora del tratamiento computacional de la red de contextos. Otra gran ventaja es que permite la obtención de parte del conocimiento de forma off-line, por lo que no tenemos que realizar un recálculo de la satisfacción de todas las restricciones para cada observación. En [22] se presenta un prototipo para un sistema automático de diagnosis basado en un modelo CSP para redes de computadores. Básicamente se trata de formular el problema de diagnosis como un problema de satisfacción de restricciones parcial, donde las restricciones capturan la estructura y comportamiento del sistema a modelar, y representan las relaciones entre el sistema de componentes. Las restricciones violadas son los indicadores de las causas del fallo, es decir, encontramos un fallo cuando alguna de las restricciones no puede ser satisfecha.

La estructura que seguiremos en el artículo será la siguiente: en primer lugar se presenta la metodología DX, exponiendo las definiciones y notación necesaria para formalizar dicha área a través un ejemplo propuesto. En el siguiente apartado se proponen los pasos a seguir para obtener un modelo CSP con el objetivo de diagnosticar un sistema de componentes. A continuación se presenta la forma de aplicar esta metodología a la diagnosis del software. Se finaliza presentando las conclusiones.

2. Diagnosis basada en modelos, metodología DX

La mayoría de las aproximaciones aparecidas en la última década para realizar diagnosis se han basado en el uso de modelos. Estos modelos se apoyan en el conocimiento del sistema a diagnosticar, que puede estar bien estructurado for-

Tabla 1: Modelo para la diagnosis del ejemplo de la figura 1

Concepto:	Valor para el ejemplo:		
COMPS	A1, A2, M1, M2, M3		
SD	ADD(c) $\wedge\neg$AB(c) $\Rightarrow$ out(c) = in1(c) + in2(c)		$\neg$AB(A1) $\Rightarrow$ f = x + y
	MULT(c) $\wedge\neg$AB(c) $\Rightarrow$ out(c) = in1(c) + in2(c)		$\neg$AB(A2) $\Rightarrow$ g = y + z
	ADD(A1), ADD(A2), ADD(A3), MULT(M1), MULT(M2)	equivalente a	$\neg$AB(M1) $\Rightarrow$ x = a * c
	out(M1) = in(A1), out(M2) = in(A1), out(M2) = in(A2)		$\neg$AB(M2) $\Rightarrow$ y = b * d
	out(M3) = in(A2), in(M1) = in(M3)		$\neg$AB(M3) $\Rightarrow$ z = c * e
OBS	MO_1={a=3, b=2, c=2, d=3, e=3, f=10, g=12}		
	MO_2={a=3, b=2, c=2, d=3, e=3, f=10, g=10}		
CCM	Para MO_1: {M1, M2, A1} y {M1, M3, A1, A2}		
	Para MO_2: {M1, M2, A1} y {M2, M3, A2}		
Diagnosis Mínima	Para MO_1: Δ1={A1}, Δ2={M1}, Δ3={A2, M2}, Δ4={M2, M3}		
	Para MO_2: Δ1 = {M2}, Δ2 = {A1, A2}, Δ3 = {A1, M3}, Δ4 = {A2, M1}, Δ5 = {M1, M3}		

malmente y de acuerdo con teorías establecidas, o bien, puede conocerse a través de los conocimientos de un experto y datos del sistema o proceso. También se presentan algunas veces combinaciones de ambos tipos de información. Los primeros estudios donde la formalización de la diagnosis se concretó se presentaron en [21], donde se propone una teoría general para el problema de explicar las discrepancias entre los comportamientos observados y los correctos de los componentes de un sistema. Para explicar las bases de esta formalización seguiremos un ejemplo muy utilizado en la bibliografía relativa [7], [14], [8] y [6], formado por tres multiplicadores y dos sumadores, tal como aparece en la figura 1.

Definición 2.1. *Modelo del sistema*: El modelo del sistema estará formado por el par (SD,COMPS) donde: SD (System Description) será un conjunto de ecuaciones lógicas de igualdad de primer orden; y COMPS (Components) será un conjunto finito de componentes. A partir del modelo del sistema podemos conocer cuáles son las interconexiones (SD) entre los diferentes componentes (COMPS).

La definición de diagnosis se sustenta en el concepto de comportamiento anormal. Básicamente si un componente tiene un comportamiento no anormal, entonces su funcionamiento es correcto. SD hace uso del predicado AB para identificar si un comportamiento es anormal o no. Dado un componente c perteneciente a COMPS, si el predicado $\neg$AB(c) es verdadero, entonces el componente funciona correctamente.

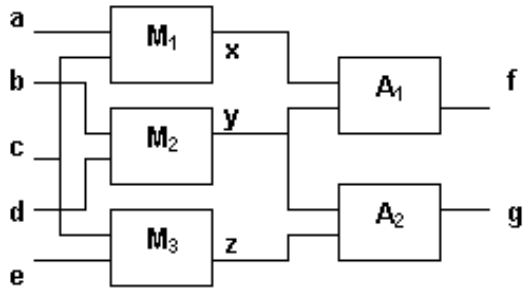

Figura 1. Circuito de ejemplo: *a*, *b*, *c*, *d*, y *e* son entradas; *f* y *g* son salidas; *x*, *y* y *z* son valores internos no observables. M_1,M_2 y M_3 son multiplicadores y A_1 y A_2 son sumadores

En la tabla 1 aparece el modelo del sistema para el ejemplo de la figura 1. El conjunto de ecuaciones lógicas de igualdad que forman el SD, describen el comportamiento de los diferentes componentes (sumadores y multiplicadores) y las interconexiones entre ellos. Por ejemplo, la ecuación lógica que describe los sumadores expresa que si c es un sumador y no tiene un comportamiento anormal, entonces la salida del sumador es la suma de las entradas. Sin embargo, no sabemos como se comportará el sumador si su comportamiento es anormal. Nombrando las entradas como a,b,c,d, y e, y las salidas como f y g tal como aparece en la figura 1, es posible reducir el SD al equivalente que aparece en la misma tabla 1.

Definición 2.2. *Modelo observacional (OBS)*: Será una tupla que asigna valores a las variables observables. Un problema de diagnosis vendrá dado por la tripleta (SD, COMPS, OBS). En la tabla 1 aparecen ejemplos de OBS para el sistema propuesto.

Definición 2.3. *Diagnosis*: La diagnosis para (SD, COMPS, OBS) será un conjunto de componentes $\Delta \subseteq$ COMPS tal que SD $\bigcup$ OBS $\bigcup$ {AB(c) | c $\in \Delta$} $\bigcup$ {$\neg$AB(c) | c $\in$ COMPS-Δ} puede ser satisfecho. El número de posibles diagnosis es exponencial, concretamente 2^{COMPS}. Nuestro objetivo es refinar la diagnosis y quedarnos con un número menor de posibilidades con la misma información.

Definición 2.4. *Diagnosis mínima*: La diagnosis mínima será aquella diagnosis Δ tal que $\forall\Delta' \subset \Delta$, Δ' no es una diagnosis.

Definición 2.5. *Conjuntos conflictivos*: Será un conjunto de componentes C = {c_1, ..., c_k} $\subseteq$ COMPS tal que SD $\bigcup$ OBS $\bigcup$ {$\neg$AB(c) | c $\in$ C} es inconsistente.

Definición 2.6. *Conjuntos conflictivos mínimos (CCM)*: Será un conjunto conflictivo que no incluye ningún otro conjunto conflictivo. En la tabla 1 aparecen los conjuntos conflictivos mínimos para los modelos observacionales propuestos.

Para obtener los conjunto conflictivos se suele utilizar un motor de inferencia lógica [21], y luego seleccionar cuáles son los CCM. En [13] se mejora la obtención de los CCM, consiguiendo realizar parte de este proceso de forma off-line.

Definición 2.7. *Hitting Set*: Será un conjunto de componentes intersección de los conjuntos conflictivos mínimos. El conjunto mínimo afectado (*Minimal Hitting Set*) será un conjunto de componentes que incluirá un componente de cada uno de los conjuntos conflictivos mínimos.

Proposición 2.1. *Diagnosis mínima*: Δ será la diagnosis mínima para (SD, COMPS, OBS) si y sólo si, es el conjunto mínimo afectado de la colección de conjuntos conflictivos mínimos para (SD, COMPS, OBS).

Para obtener el conjunto mínimo afectado partiremos de los conjuntos conflictivos mínimos, y seleccionaremos primero cuáles son los componentes que pertenecen a todos los CCM. En el caso del MO_1, seleccionaremos a {A1} y {M1} como parte de la diagnosis mínima, además, estos dos componentes no podrán pertenecer a las próximas diagnosis mínimas que incluyan más componentes. Con los otros tres componente intentaremos formar grupos de dos componentes que cubran todos los CCM. De esta forma obtendremos {A2, M2}, y {M2, M3} como posibles diagnosis mínima. Si aún quedarán componentes de los que forman los CCM por cubrir, seguiríamos formando grupos de tres, cuatro, o más componentes hasta utilizar todos. Para los modelos observacionales MO_1 y MO_2 en la tabla 1 se muestran cuáles serían las diagnosis mínimas.

El proceso de diagnosis basado en modelos es en su mayoría un proceso online, a diferencia de la metodología FDI, que permite realizar parte del trabajo previamente al proceso de diagnosis. Otra diferencia notable con FDI, es la utilización de un motor de inferencia lógica para calcular la solución.

Como se puede observar, la diagnosis de componentes se basa en detectar cuáles son los síntomas, es decir, las diferencias entre las predicciones hechas por el motor de inferencias y las observaciones obtenidas por la monitorización. Los síntomas indican que componentes pueden estar fallando, y cuáles son los conflictos, es decir, los conjuntos de componentes que no funcionan bien conjuntamente. En definitiva, la diagnosis basada en modelos busca realizar una hipótesis de por qué el sistema difiere del modelo especificado.

3. Diagnosis utilizando técnicas CSP

La programación con restricciones nos permite modelar y resolver problemas reales a través de conjunto de variables asociadas a unos dominios, y un conjunto de ecuaciones definidas para un subconjunto de variables. Al plantear un problema CSP el objetivo es encontrar para las variables una asignación que satisfaga todas las restricciones. Pero existen determinados problemas en los que es inviable satisfacer todas las restricciones. El objetivo al plantear un problema **Max-CSP** (Maximization Constraint Satisfaction Problem) es encontrar una asignación que satisfaga el mayor número de restricciones, y que minimice el número de restricciones violadas.

Básicamente se trata de definir una función objetivo que el motor de inferencias debe intentar maximizar mientras busca la solución para el CSP planteado. Para plantear la función objetivo, las restricciones que puedan ser violadas deben estar definidas como restricciones de *satisfacción concretada* (*reified constraints*), es decir, con un valor de verdad asociado a la satisfacción o no de cada restricción. Para cada una de estas restricciones se añade una variable booleana, que llamaremos

Tabla 2: Diagnosis para el ejemplo de la figura 1 usando metodologías DX, CSP y Max-CSP

DX		CSP		Max−CSP	
SD:	¬AB(A1) ⇒ f = x + y	Restricciones:	AB(A1) ∨ f = x + y	Restricciones:	¬AB(A1) = $(f = x + y)$
	¬AB(A2) ⇒ g = y + z		AB(A2) ∨ g = y + z		¬AB(A2) = $(g = y + z)$
	¬AB(M1) ⇒ x = a * c		AB(M1) ∨ x = a * c		¬AB(M1) = $(x = a * c)$
	¬AB(M2) ⇒ y = b * d		AB(M2) ∨ y = b * d		¬AB(M2) = $(y = b * d)$
	¬AB(M3) ⇒ z = c * e		AB(M3) ∨ z = c * e		¬AB(M3) = $(z = c * e)$
OBS:	a = 3, b = 2, c = 2,	Dominios:	a = 3, b = 2, c = 2,	Dominios:	a = 3, b = 2, c = 2,
	d=3, e=3,		d = 3, e = 3,		d = 3, e = 3,
	f=10, g=12		f = 10, g = 12		f = 10, g = 12
COMPS:	A1, A2, M1,		AB(M1), AB(M2),		AB(M1), AB(M2),
	M2, M3		AB(M3), AB(A1),		AB(M3), AB(A1),
			AB(A2)=		AB(A2)=
			{verdadero, falso}		{verdadero, falso}
			x, y, z = libre		x, y, z = libre
				F. objetivo:	Max(N c : c ∈ {M1,
					M2, M3, A1, A2} :
					¬AB(c) = verdadero)

variable de satisfactibilidad, que almacenará si se viola o no dicha restricción al alcanzar la solución del problema Max-CSP. La cardinalidad del conjunto de *variables de satisfactibilidad* que toman el valor verdadero será la función objetivo que buscamos maximizar, o lo que es lo mismo, satisfacer el mayor número de restricciones. Determinados problemas exigen que ciertas restricciones deban cumplirse preferentemente, en este caso se podrán utilizar pesos para las diferentes *variables de satisfactibilidad* dentro de la función objetivo.

La diagnosis de sistemas busca encontrar qué componentes tienen un comportamiento anormal, es decir, cuáles de los componentes tienen un comportamiento diferente al especificado en las ecuaciones que forman la descripción del sistema. Si pensamos en las ecuaciones que forman el SD como un conjunto de restricciones podremos plantear la diagnosis de sistemas mediante un problema CSP. En la tabla 2 aparece, para el ejemplo de la figura 1, como a partir del modelo del sistema se pueden obtener directamente cuáles serían las restricciones para un problema CSP. Cada una de las ecuaciones lógicas del modelo del sistema se traduce a una restricción de tipo OR que debe cumplirse al alcanzar la solución para el problem CSP. En el caso de la primera restricción del ejemplo, debe cumplirse al menos una de las dos siguientes condiciones: o bien que el componente tenga un comportamiento anormal, o bien que la salida f del componente sea igual a la suma de las entradas x e y. Con los valores del modelo observacional es posible acotar los dominios de las variables implicadas en las restricciones, sin embargo, otras variables quedarán libres, como las variables no observables x, y, y z, así como el predicado $\mathrm{AB(C}_i\mathrm{)}$ aplicado a cada componente.

Encontrar la diagnosis mínima implica encontrar un conjunto de cardinalidad mínima de componentes con funcionamiento anormal. Pero esta propiedad de la diagnosis mínima no la estamos utilizando si planteamos el problema como un simple problema CSP. Para optimizar y guiar la búsqueda de la diagnosis mínima, se debe plantear como un problema Max-CSP. Un componente tendrá un comportamiento anormal, si es diferente al especificado en las ecuaciones que hacen referencia a dicho componente, es decir, si las restricciones asociadas a dicho componente no pueden satisfacerse. Por tanto, el valor del predicado AB vendrá dado, por la satisfacción o no de las restricciones asociadas a dicho componente. Tal como aparece en la tabla 2, para el ejemplo de la figura 1, plantearemos un problema Max-CSP donde los predicados $\mathrm{AB(C}_i\mathrm{)}$ actuarán como *variables de satisfactibilidad*. La función objetivo debe maximizar el número de predicados $\neg\mathrm{AB(C}_i\mathrm{)}$ que tomen el valor verdadero. De esta forma, aunque los predicados $\mathrm{AB(C}_i\mathrm{)}$ parten inicialmente como variables libres, tendrán su dominio intrinsicamente ligado a la satisfacción de las restricciones asociadas a los componentes.

Con la resolución del problema Max-CSP planteado se tendrán los valores asociados a los predicados $\mathrm{AB(c}_i\mathrm{)}$ para cada uno de los componentes. Estos valores definirán el grupo de componentes que forman la diagnosis. Para el problema planteado en la tabla 2, se buscarán primero aquellas solu-

Tabla 3: Ejemplo 1

Ejemplo 1:	Forma SSA del ejemplo 1: (para todos los OBS)
{Pre: a, b, c, d, e>0}	{Pre: a0, b0, c0, d0, e0>0}
(S01) x=a*c;	(S01) x0=a0*c0;
(S02) y=b*d;	(S02) y0=b0*d0;
(S03) z=c*e;	(S03) z0=c0*e0;
(S04) f=x+y;	(S04) f0=x0+y0;
(S05) g=y+z;	(S05) g0=y0+z0;
{Post:f=a*c+b*d ∧ g=b*d+c*e}	{Post:f0=a0*c0+b0*d0 ∧ g0=b0*d0+c0*e0}

ciones que permitan alcanzar un funcionamiento correcto del sistema modificando un sólo componente. En este caso resultan ser {A1} y {M1}. Pero, ¿es ésta la única diagnosis posible?. Evidentemente no, el proceso Max-CSP en principio ha conseguido maximizar la función objetivo utilizando sólo un componente, pero existe la posibilidad de que sean dos o más componentes los que fallen simultáneamente.

Para seguir buscando el resto de posibles diagnosis mínimas el problema Max-CSP debe continuar buscando más soluciones. Pero previamente resulta necesario añadir como restricciones que los predicados AB(c_i) correspondientes a los componentes que ya han sido obtenidos en el proceso de diagnosis, deben ser falsos obligatoriamente, y de esta forma garantizar que no se generarán nuevas soluciones cuyo conjunto de componentes englobe los componentes que formen alguna de las diagnosis mínimas ya obtenidas.

Para el ejemplo de la tabla 2 se obtendrían como diagnosis mínima utilizando dos componentes los grupos {A2, M2} y {M2, M3}. Si aún quedarán componentes por cubrir, seguiríamos buscando soluciones formadas por grupos de tres, cuatro, o más componentes hasta utilizar todos, añadiendo las restricciones necesarias para que no se repitan las diagnosis mínimas obtenidas previamente.

4. Diagnosis del software utilizando técnicas Max-CSP

En el apartado anterior se ha mostrado la forma de realizar la diagnosis de componentes utilizando técnicas Max-CSP. A continuación vamos a utilizar la misma metodología para encontrar errores en el desarrollo de software. Para desarrollar software existen multitud de lenguajes, en este artículo nos centraremos en los lenguajes imperativos orientados a objetos, como por ejemplo Java^{TM}.

La diagnosis del software permite identificar y localizar el origen de los errores de un programa software. Los errores (*bugs*) que contemplaremos serán aquellos que surgen como una pequeña variación del programa correcto, como por ejemplo fallos en las guardas (de bucles o sentencias selectivas) o errores en la asignación a variables. Un programa tendrá un error si no existe concordancia entre los resultados especificados como correctos y los resultados observados tras la ejecución. No tendremos en cuenta los errores en tiempo de compilación, como errores en la sintaxis del lenguaje; ni los errores dinámicos, tales como excepciones, violaciones de acceso a memoria, etc.

Comparando el programa que aparece en el ejemplo 1 de la tabla 3, y el sistema de componentes del ejemplo de la figura 1, y pensando en las sentencias y expresiones como si fuesen componentes, y en las variables como si fuesen las conexiones entre componentes, nos daremos cuenta que ambos ejemplos, el sistema de componentes y el programa software, pretenden modelar el mismo comportamiento. Si para un sistema de componentes hemos logrado alcanzar la diagnosis mínima utilizando técnicas Max-CSP, podemos utilizar la misma metodología para la diagnosis de un programa software con el objetivo de encontrar la sentencia o grupo de sentencias que impidan alcanzar el resultado especificado como correcto. En la diagnosis de componentes, es el componente el que falla por un funcionamiento anómalo, pero en la diagnosis del software la instrucción que se ejecuta nunca falla, el error se produce por que no hemos escogido la instrucción adecuada.

Para poder aplicar las técnicas Max-CSP a la diagnosis del software necesitaremos: Disponer de la especificación del comportamiento esperado del programa; encontrar la manera adecuada de transformar a restricciones las sentencias del código fuente del programa que permitan modelar el

Tabla 4: Ejemplo 2

Ejemplo 2:	Forma SSA del ejemplo 2: (OBS≡{x0=5, y0=7})
{Pre: $x > 0 \wedge y > 0$}	{Pre: $x0 > 0 \wedge y0 > 0$}
(-) public int calculo(int x, int y){	(-) public int calculo(int x0, int y0){
(-) int max, min, sum, z;	(-) int max0, min0, sum0, z0;
(S01) if (x>=y){	(S01) if (x0>=y0){
(S02) min=y;	(S02) min0=y0;
(S03) max=x;	(S03) max0=x0;
(-) }else{	(-) }else{
(S04) min=x;	(S04) min0=x0;
(S05) max=y;	(S05) max0=y0;
(-) }	(-) }
(S06) z=max-min;	(S06) z0=max0-min0;
(S07) sum=min*(z+1);	(S07) sum0=min0*(z0+1);
(S08) while (z>0){	(S08a) if (z0>0){
(S09) sum=sum+z;	(S09a) sum1=sum0+z0;
(S10) z=z-1;	(S10a) z1=z0-1;}
(-) }	(-) }
(S11) return sum;	(S08b) if (z1>0){
(-) }	(S09b) sum2=sum1+z1;
{Post:($x >= y \wedge sum = \sum \alpha$:y≤$\alpha$≤x:$\alpha$)	(S10b) z2=z1-1;
$\vee$ ($y > x \wedge sum = \sum \alpha$:x≤$\alpha$≤y:$\alpha$) }	(-) }
	(S11) return sum2;
	(-) }
	{Post:($x0 >= y0 \wedge sum2 = \sum \alpha$:y0≤$\alpha$≤x0:$\alpha$)
	$\vee$ ($y0 > x0 \wedge sum2 = \sum \alpha$:x0≤$\alpha$≤y0:$\alpha$) }

comportamiento de dichas sentencias; y encontrar la forma de generar modelos observacionales para ejecutar el programa y saber si cumplen las especificaciones establecidas para el comportamiento del programa. En este apartado veremos como obtener estos requisitos y como alcanzar finalmente la diagnosis del software.

Para continuar con el resto del artículo se hace necesario previamente desarrollar las siguientes definiciones.

4.1. Notación y definiciones

Definición 4.1. *DbC, design by contract*: El diseño por contrato [19] permite especificar cual debe ser el comportamiento de un programa, a través de precondiciones, postcondiciones, asertos, etc. A través de estas especificaciones dispondremos del comportamiento esperado del programa.

Definición 4.2. *Camino*: Dado el código fuente de un programa, definiremos *camino* como una de las posibles secuencias de sentencias del grafo de control de flujo (CFG, control flow graph). Un CFG [2] es un grafo dirigido que representa la estructura de control de un programa. Un CFG se compone de un conjunto de bloques secuenciales y sentencias de decisión. Un bloque secuencial será una secuencia de sentencias sin ninguna instrucción condicional interna, por tanto solo dispone de un punto de entrada y otro de salida. Una sentencia de decisión es un punto del programa donde el control de flujo puede divergir debido a una condición, por ejemplo una sentencia selectiva o un bucle.

Las sentencias de decisión nos permiten incorporar diferentes sentencias dependiendo de las entradas iniciales de nuestro programa. Esta característica del software no existe en los sistemas de componentes, ya que un sistema de componentes no cambia su comportamiento en el tiempo, ni incorpora nuevos componentes en función del modelo observacional. Pero un programa software puede cambiar su comportamiento a medida que se ejecuta en función de los valores de entrada, e incorporar unas sentencias u otras. He aquí la primera dificultad que debemos salvar, ya que dependiendo de las entradas del programa, debemos incorporar unas sentencias (componentes) u otras. Para diferentes entradas es posible que los modelos asociados sean diferentes, aunque el pro-

grama sea el mismo.

Definición 4.3. *Modelo Observacional (OBS)*: Será una tupla que asigne valores a las variables de entrada y salida de un programa. Para cada OBS tendremos un camino asociado. Las técnicas de *testing* permiten seleccionar que entradas son las más significativas y aportan más información a la hora de detectar errores en programas. El objetivo es ejecutar el programa con esas entradas y comprobar si las salidas cumplen la especificación del programa y por tanto son correctas. Utilizar las entradas proporcionados por las técnicas de *testing* nos permite saber si existen o no errores, para posteriormente aplicar el proceso de diagnosis y localizar el origen de éstos. Las entradas que nos proporcione el *testing* deben satisfacer la precondición del código a diagnosticar. Las salidas correctas del código vendrán dadas por la postcondición si el diseño por contrato es lo suficientemente fuerte; si esto no sucede, un experto es quien debe indicar qué salidas son las correctas. El modelo observacional para la diagnosis de componentes corresponde con las mediciones de los valores reales que se producen como entradas y salidas del sistema, pero en la diagnosis del software, los valores del modelo observacional corresponden con los valores que deben darse en las salidas, si el programa funcionase correctamente con las entradas establecidas. Esto es debido a que como se ha visto anteriormente, una sentencia de un programa nunca falla, el error está en colocar la sentencia inadecuada, por tanto, si el modelo observacional correspondiese con las mediciones de los valores reales nunca habría discrepancias con el modelo del sistema basado en el código fuente , y por tanto sería imposible localizar el origen del error.

Definición 4.4. *Modelo del sistema (SD)*: Estará compuesto por un conjunto de restricciones y variables con sus dominios, igual que ocurría con la diagnosis de componentes. El comportamiento del programa se determina a través del conjunto de restricciones gracias a las relaciones entre las variables del código fuente. El modelo del sistema se obtendrá, tal como se verá mas adelante, a partir del código fuente y la especificación del contrato software a través de precondiciones, postcondiciones y asertos.

Definición 4.5. *Diagnosis*: Será el conjunto de sentencias que provocan el comportamiento anómalo del programa, y que por tanto deben ser modificadas.

Definición 4.6. *Diagnosis mínima*: Será la diagnosis que implica modificar el menor número de sentencias del código fuente.

4.2. Metodología de diagnosis

Los pasos a seguir para realizar la diagnosis son:

1. Aplicación de las técnicas de testing
2. Obtención del modelo del sistema
3. Generación de la diagnosis

4.2.1. Aplicación de las técnicas de testing

El objetivo de las técnicas de *testing* es seleccionar una serie de valores que sirvan de iniciación, o como parámetros de entrada, para un determinado código fuente, para posteriormente detectar si existen errores en la ejecución de los programas, es decir, sino se cumplen las especificaciones establecidas en el contrato software. Tras aplicar las técnicas de *testing* a un programa y utilizando las entradas seleccionadas por el *testing*, podemos deducir, comparando las salidas del programa con las salidas estipuladas en la especificación del contrato software, si el código fuente no es correcto. En nuestro caso, además las técnicas de *testing* nos proporcionarán valores iniciales para los diferentes modelos observacionales que utilizaremos en el proceso de diagnosis. Existen diferentes formas de aplicar las técnicas de *testing*, aquí nos centraremos en la generación de test con criterios de control de flujo y cobertura de caminos (las entradas generadas provocarán la ejecución de un camino concreto). Para generar los datos o entradas que conforman el test existen muchas aproximaciones. En nuestro caso usaremos las denominadas *goal-oriented*, que se basan en traducir la generación de las entradas del test a un conjunto de restricciones que pueden ser resueltas, como en el caso de [15], por un esquema CLP.

4.2.2. Obtención del modelo del sistema

Determinar la forma SSA: Como se ha visto en el apartado anterior, utilizando las entradas proporcionados por las técnicas de *testing*, podemos seleccionar el código fuente que vamos a diagnosticar y que no cumple el comportamiento que

Tabla 5: Modelo del sistema del ejemplo 1

Tipo restricción:		Restricciones:
Restric. Precondición		a0>0, b0>0, c0>0, d0>0, e0>0
Restric. Postcondición		f0=a0*c0+b0*d0 ∧ g0=b0*d0+c0*e0
Restricciones del código	S1:	{¬AB(S_1)∧P(S_1)⇒x0=a0*c0}
	S2:	{¬AB(S_2)∧P(S_2)⇒y0=b0*d0}
	S3:	{¬AB(S_3)∧P(S_3)⇒z0=c0*e0}
	S4:	{¬AB(S_4)∧P(S_4)⇒f0=x0+y0}
	S5:	{¬AB(S_5)∧P(S_5)⇒g0=y0+z0}

se especificó como correcto. El siguiente paso es traducir el programa bajo análisis en la forma SSA (Static Single Assignment). La forma SSA es equivalente al programa original pero sólo se permite una asignación para cada variable en el programa completo, y cada referencia que se realice sobre una variable será por tanto para un valor en concreto de todos los que la variable almacenaría en el programa original. De esta forma tendremos almacenados todos los valores por los que ha pasado una variable en la ejecución de un programa. Por ejemplo el código *x=a*c; ...x=x+3;...* {*Post:x =...*} cambiará a *x0=a0*c0; ...x1=x0+3;...* {*Post:x1 =...*}.

Este paso es previo y necesario a la generación de las restricciones que formarán el modelo del sistema, ya que como se verá al obtener las restricciones asociadas a las sentencias de asignación, estas se traducirán en igualdades que deben cumplirse, y sucesivas asignaciones a una misma variable podrían implicar una serie de restricciones de igualdad que a la vez serían no se podrían satisfacer (incluso aunque el programa fuera correcto).

Las sentencias selectivas o los bucles contienen ramificaciones y puntos de unión. En un punto de unión, aunque existan diferentes asignaciones a una misma variable provenientes de diferentes ramas, se debe alcanzar el mismo resultado que obtendríamos con el programa original, por tanto los posibles valores que se alcanzarían en cada una de las ramas para una misma variable, deben agruparse en uno solo cuando lleguemos al punto de unión en la forma SSA. Para una mayor concreción de la forma SSA puede consultarse [1].

Determinación de las variables y sus dominios: El conjunto de variables X={x_1, x_2... ,x_n} estará compuesto de todas las variables que aparezcan en la forma SSA. El dominio de cada variable vendrá determinado por la declaración de la variable, y será el mismo que fije el compilador para los diferentes tipos de datos.

Determinación del camino: Dado un modelo observacional podremos determinar un camino formado por una secuencia de sentencias pertenecientes al CFG, tal como aparece en el apartado de definiciones. Como hemos visto, a partir de dicho conjunto de sentencias obtendremos la forma SSA del código fuente.

Definición 4.7. *Predicado P*: Cuando un programa se ejecuta utilizando un determinado OBS, el predicado P aplicado sobre cada una de las sentencias de la forma SSA nos indicará si dicha sentencia forma parte del camino seguido en la ejecución del código. Dada una sentencia S_x, si *P(S_x)* es verdadero entonces S_x pertenece al camino seguido en la ejecución del código; en caso contrario no pertenecerá al camino.

Si en un programa tenemos una sentencia condicional, por ejemplo una sentencia selectiva *if*, la condición de dicha sentencia puede ser el origen de un error. Con el uso del predicado P, al diagnosticar un programa es posible probar si el camino alternativo es el adecuado y por tanto la condición es incorrecta. Es decir, podemos detectar errores en las guardas de las sentencias condicionales y en base a ello, cambiar el camino seguido por el programa. Este predicado no era necesario en la diagnosis de componentes ya que un sistema de componentes no cambia su estructura en el tiempo, ni incorpora nuevos componentes en función del modelo observacional. Sin embargo en el caso del software, las sentencias de decisión nos permiten incorporar diferentes instrucciones dependiendo de las entradas del programa.

Determinación de las restricciones del modelo del sistema: Las restricciones del modelo del sistema serán un conjunto formado por las restricciones que obtengamos de la especificación del diseño bajo contrato y del código fuente. Las

Tabla 6: Modelo del sistema del ejemplo 2

Tipo restricción:		Restricciones:
Restric. Precondición		x0>0, y0>0
Restric. Postcondición		$(x0 >= y0 \wedge sum2 = \sum \alpha$:y0≤$\alpha$≤x0:$\alpha) \vee$ $(y0 > x0 \wedge sum2 = \sum \alpha$:x0≤$\alpha$≤y0:$\alpha)$
Restricciones del código	S1:	{¬AB(S_1) ⇒ P(S_2) = (x0>y0), ¬AB(S_1) ⇒ P(S_3) = (x0>y0) , ¬AB(S_1) ⇒ P(S_4) = ¬ (x0>y0), ¬AB(S_1) ⇒ P(S_5) = ¬ (x0>y0) , P(S_2) ≠ P(S_4) , P(S_2) ≠ P(S_5) } P(S_3) ≠ P(S_4) , P(S_3) ≠ P(S_5)}
	S2:	{¬AB(S_2)∧P(S_2)⇒min0=y0}
	S3:	{¬AB(S_3)∧P(S_3)⇒max0=x0}
	S4:	{¬AB(S_4)∧P(S_4)⇒min0=x0}
	S5:	{¬AB(S_5)∧P(S_5)⇒max0=y0}
	S6:	{¬AB(S_6)∧P(S_6)⇒z0=max0-min0}
	S7:	{¬AB(S_7)∧P(S_7)⇒sum0=min0*(z0+1)}
	S8a:	{¬AB(S_{8a}) ⇒ P(S_{9a}) = (z0>0), ¬AB(S_{8a}) ⇒ P(S_{10a}) = (z0>0)}
	S9a:	{¬AB(S_{9a})∧P(S_{9a})⇒sum1=sum0+z0 }
	S10a:	{¬AB(S_{10a})∧P(S_{10a})⇒z1=z0-1 }
	S8b:	{¬AB(S_{8a})∧¬AB(S_{8b}) ⇒ P(S_{9b}) = (z1>0)} {¬AB(S_{8a})∧¬AB(S_{8b}) ⇒ P(S_{10b}) = (z1>0)}
	S9b:	{¬AB(S_{9b})∧P(S_{9b})⇒sum2=sum1+z1 }
	S10b:	{¬AB(S_{10b})∧P(S_{10b})⇒z2=z1-1}

restricciones procedentes del diseño por contrato serán obtenidas directamente de las precondiciones, asertos y postcondiciones. Además, deberán cumplirse siempre, ya que expresan cuáles son las condiciones iniciales, finales e intermedias que debe satisfacer el código fuente libre de errores.

Definición 4.8. *Predicado AB*: Será aquel que aplicado a las restricciones asociadas a una sentencia S_x, nos devolverá *verdadero* si la sentencia S_x forma parte de la diagnosis mínima. Aquellas sentencias S_x pertenecientes al código fuente que verifiquen AB(S_x)=falso serán las que forman parte de la diagnosis mínima, y por tanto son las sentencias que provocan el comportamiento anómalo del programa.

Las restricciones del código fuente se obtendrán transformando las sentencias que forman el SSA. La forma de obtener las restricciones asociadas a las sentencias será la siguiente:

- Bloques secuenciales:

  Declaración de variables: El dominio de cada variable será el que fije el compilador para cada tipo definido. Las variables enteras tendrán un dominio finito que se restringirá en función de lo especificado por el compilador y por la inicialización de cada variable. En los ejemplos que vamos a tratar solo usaremos variables enteras y booleanas, ya que el resto de tipos necesitarían un estudio más detallado.

  Asignaciones:

  Asignación::= Ident=Exp

  A partir de la sentencia de asignación obtendremos la siguiente restricción del modelo:

  ¬AB(S_{Asig}) ∧ P(S_{Asig}) ⇒ Ident=Exp

  Con esta restricción indicamos que si la sentencia S_{Asig} pertenece al grupo de sentencias que forman el camino y además es una sentencia que no forma parte de la diagnosis mínima, tras la ejecución de S_{Asig} debe cumplirse la igualdad entre la variable asignada y la expresión que se asigna. Si la sentencia S_{Asig} no pertenece al camino las sentencia no se ejecutaría y la asignación no tendría efecto sobre el resultado final. Si S_{Asig} forma parte de la diagnosis mínima, no tiene por qué cumplirse la igualdad *Ident=Exp*, ya que esta sentencia debe cambiar para alcanzar un resultado correcto. De esta forma hemos cubierto todas las posibilidades de comportamiento asociadas a la sentencia de asignación. En el ejemplo 2 que aparece en la tabla 4, para la sentencia S_6 obtendremos:

$\neg AB(S_6) \wedge P(S_6) \Rightarrow z0=max0\text{-}min0$

Llamadas a métodos: Supondremos que todos los métodos especifican su precondición y postcondición. Para cada llamada a un método, añadiremos al *SD* las restricciones definidas en la precondición y la postcondición del método. Cuando nos encontremos ante una método recursivo, las llamadas internas al método recursivo deben suponerse válidas para poder llevar a cabo la verificación formal de las llamadas recursivas.

- Bloques condicionales:

Selectiva ::= if (b) {B_1} [else {B_2}]

Los caminos posibles para una sentencia condicional son:

Camino 1: bB_1 (condición b es verdadera)
Camino 2: $\neg bB_2$ (condición b es falsa)

Dependiendo del OBS, iremos por un camino u otro. Para cada sentencia selectiva se definirán las restricciones siguientes:

$\forall\ S_x \in B_1$: $\{\neg AB(S_{Selec}) \Rightarrow P(S_x) = b\}$
$\forall\ S_y \in B_2$: $\{\neg AB(S_{Selec}) \Rightarrow P(S_y) = \neg b\}$
$\forall\ S_x \in B_1 \wedge S_y \in B_2$: $P(S_x) \neq P(S_y)$ }

Con las dos primeras restricciones estamos declarando que si la sentencia selectiva es correcta (no forma parte de la diagnosis), sólo debe existir un camino posible que vendrá dado por la condición de la sentencia selectiva. Con la última restricción estamos declarando que debe seguir cumpliéndose que sólo uno de los caminos puede ejecutarse, aunque la sentencia selectiva forme parte de la diagnosis y no sepamos cual debe ser la condición de la sentencia selectiva (ya que es probable que deba ser modificada). De esta forma estamos y cubriendo toda las posibilidades de comportamiento que puede implicar una sentencia selectiva; y podremos detectar fallos en la guarda de la sentencia condicional y en las sentencias incluidas dentro de cada bloque B_1 y B_2.

Como ejemplo puede verse la transformación de la sentencia S_{11} del ejemplo 2 (tabla 4) que aparece en la tabla 6. Además de las restricciones asociadas a la sentencia selectiva aparecen las restricciones asociadas a las sentencias incluidas dentro de cada bloque B_1 y B_2.

- Bloques tipo bucle:

Bucle ::= while (b) {B}

Podemos deducir que los caminos posibles son:

Camino 1: $\neg b$ (Ninguna iteración)
Camino 2: $bB_1\neg b$ (1 iteración es ejecutada)
Camino 3: $b_1B_1b_2B_2...b_nB_n\neg b$ (2 o más iteraciones ejecutadas)

El principal problema viene dado en la imposibilidad de establecer una cota para el número de veces que debe iterar un bucle, ya que dependerá del OBS. La estructura de un bucle puede ser simulada a través de varias sentencias selectivas anidadas. Cada una de las iteraciones sería transformada a una sentencia selectiva, en la cual además de la condición de finalización del bucle, tendríamos que tener en cuenta que previamente se han ejecutado las iteraciones anteriores de forma secuencial y en el orden previsto. En [4] y [3] se propone una técnica para reducir el modelo a menos de n iteraciones, siendo n el número de iteraciones del bucle, y obtener una mayor eficiencia en el proceso de diagnosis.

Para cada bucle se definirán las restricciones siguientes:

$\forall\ S_x \in \text{Sentencias}_{Bucle}$: $\{\neg AB(S_{Bucle}) \Rightarrow P(S_x) = \text{cond}\}$

Con estas restricciones estamos declarando que siendo el bucle correcto, si iteramos o no, dependerá de la condición del bucle. Para la sentencia S_8 del ejemplo 2 podemos observar en la tabla 6 cuáles serían las restricciones asociadas al bucle y las restricciones asociadas a las sentencias incluidas dentro del bucle. En este caso se ha simulado la estructura del bucle a través de varias sentencias selectivas anidadas. Por tanto, el número de iteraciones variará de un OBS a otro. Para cada una de las iteraciones se ha utilizado una letra, de forma que la iteración primera corresponde a la letra 'a', la segunda a la letra 'b', etc.

Tabla 7: Diagnosis del método *calculo*

OBS	Cambio realizado	Diagnosis
{x = 3, y = 7}	S_7 por 'sum = min * z'	{S_7, S_9}
{x = 5, y = 2}	S_7 por 'sum = min * z'	{S_7, S_9}
{x = 4, y = 4}	S_7 por 'sum = min * z'	{S_2, S_3, S_6, S_7, S_8}

4.2.3. Generación de la diagnosis

Nuestro objetivo es encontrar el conjunto de sentencias que forman la diagnosis. Pretendemos además que dicho conjunto tenga la cardinalidad mínima. Esto implica que nuestro objetivo se traduzca en intentar maximizar el número de predicados $AB(S_x)$ que toman el valor falso, donde S_x serán las sentencias que formen parte del código fuente.

Con las restricciones del modelo del sistema y la función a maximizar que hemos descrito, podemos plantear un problema Max-CSP. De esta forma obtendremos los valores de los predicados $AB(S_x)$ para cada una de las sentencias que forman el código fuente. Estos valores definirán el grupo de sentencias que forman la diagnosis y que además tiene la cardinalidad mínima. Modificando el grupo de sentencias que forman la diagnosis mínima, será posible alcanzar el resultado correcto.

4.3. Obtención de la diagnosis usando varios modelos observacionales

Una vez aplicado el proceso de diagnosis para cada uno de los modelos observacionales por separado habremos obtenido un conjunto de sentencias que formarán la diagnosis para cada modelo observacional. Para encontrar la diagnosis mínima de todos los OBS en conjunto buscaremos el conjunto mínimo afectado (*Minimal Hitting Set*) para el conjunto de soluciones de los diferentes OBS. De esta forma tendremos el menor grupo de sentencias que cubra la diagnosis de todos los OBS.

4.4. Ejemplos

Para probar la validez de la metodología nos vamos a centrar en los dos ejemplos de la tabla 3 y 4. Haremos cambios en los ejemplos propuestos lo que provocará salidas que no cumplirán las especificaciones fijadas por el diseño por contrato. Para implementar la búsqueda Max-CSP hemos utilizado la herramienta ILOG-Solver^{TM} (Librería comercial para programación con restricciones en C++ [17])

- Cambio en el ejemplo 1 sobre la sentencia S_5: Al cambiar la sentencia S_5 por *g0=y0-z0* y aplicando la metodología propuesta, obtendremos las restricciones que forman el modelo del sistema. Serán prácticamente las mismas que aparecen en la tabla 5 salvo para S_5. Utilizando el modelo observacional *OBS*≡{*a0=2, b0=2, c0=3, d0=3, e0=2, f0=12, g0=12*}, obtendremos como diagnosis mínima las sentencias S_5, que es en efecto la instrucción que hemos modificado; y además la sentencia S_3. Si cambiamos S_3, no tendrá ninguna influencia sobre S_4 pero si sobre S_5, por tanto colocando la asignación adecuada sobre S_3 podremos alcanzar un resultado que cumpla el DbC. S_5 depende también de S_2, pero un cambio en S_2 puede implicar un error en S_4.
- Cambio en el ejemplo 2 sobre la sentencia S_7: En la tabla 7 aparece el resultado de aplicar la metodología propuesta tres veces al método *calculo*, usando tres modelos observacionales diferentes. Aplicando *hitting set* la diagnosis mínima que se obtiene es la sentencia {S_7}, que es precisamente la instrucción que hemos cambiado. Como puede apreciarse en este ejemplo, la diagnosis es más certera cuando disponemos de varios modelos observacionales. Escoger un buen conjunto de modelos observacionales resulta vital si queremos una diagnosis más precisa.

5. Conclusiones

En este trabajo hemos presentado los fundamentos de la metodología DX de diagnosis basada en modelos. La diagnosis permite saber por qué un sistema diseñado correctamente no funciona como

se esperaba. Se basa en detectar los síntomas, es decir, las diferencias entre las predicciones hechas por un motor de inferencias a partir de las ecuaciones que forman el modelo del sistema, y las observaciones obtenidas por la monitorización. Los síntomas indican qué componentes pueden estar fallando y cuáles son los conflictos. En los procesos de desarrollo y producción, tener un control sobre los fallos y las irregularidades aporta una mayor seguridad y reducción de costos. La diagnosis nos permite mantener los sistemas en los niveles deseados de seguridad y fiabilidad.

Posteriormente se han expuesto los pasos a seguir para poder plantear la diagnosis de un sistema como un problema Max-CSP. De esta forma será posible incorporar al proceso de generación de la diagnosis de un sistema, los avances y optimizaciones que se han alcanzado en el campo de la búsqueda de soluciones para problemas CSP. Por último hemos aplicado la diagnosis al software. La diagnosis del software presupone que los componentes (sentencias) no fallan, y busca por qué el sistema (programa software) no se ha diseñado correctamente. Para llevar a cabo el proceso de diagnosis usaremos el diseño por contrato que especificará cual es el comportamiento esperado de un programa, y las técnicas de *testing* para generar los modelos observacionales. Como se ha expuesto, la diagnosis del software plantea nuevos problemas que no existían en la diagnosis de componentes, debido a la posibilidad de incorporar diferentes sentencias a la traza ejecutada, en función de las entradas recibidas por el programa. Un diseño por contrato detallado y el uso de varios modelos observacionales implica una diagnosis más certera, tal como se ha mostrado con los ejemplos.

6. Agradecimiento

Este trabajo ha sido parcialmente financiado por la Comisión Internacional de Ciencia y Tecnología (DPI2000-0666-C02-02).

Referencias

[1] B. Alpern, M. Wegman, y F. K. Zadeck. Detecting equality of variables in programs. *ACM Symp. on Principles of Programming Languages (PoPL)*, pag. 1–11, Enero 1988.

[2] T. Ball y J. R. Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1319–1360, 1994.

[3] R. Ceballos, R. M. Gasca, Carmelo Del Valle, y Miguel Toro. A constraint-based methodology for software diagnosis. *Constraints in Formal Verification WorkShop, International Conference on Principles and Practice of Constraint Programming, CP2002*, Septiembre 2002.

[4] R. Ceballos, R. M. Gasca, Carmelo Del Valle, y Miguel Toro. Max-csp approach for software diagnosis. *VIII Conferencia Iberoamericana de Inteligencia Artificial, LNAI*, 2527:172–181, Noviembre 2002.

[5] L. Console y O. Dressler. Model-based diagnosis in the real world: Lessons learned and challenges remaining. *Proceedings IJCAI99*, pag. 1393–1400, 1999.

[6] M. Cordier, F. Lévy, J. Montmain, L. Travé-massuyès, M. Dumas, M. Staroswiecki, y P. Dague. A comparative analysis of AI and control theory approaches to model-based diagnosis. *14th European Conference on Artificial Intelligence*, pag. 136–140, 2000.

[7] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[8] J. de Kleer, A. Mackworth, y R. Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[9] J. de Kleer y B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[10] O. Dressler y P. Struss. *The Consistency-based Approach to Automated Diagnosis of Technical Devices*. Principles of Knowledge Representation. CSLI Publications, Stanford, 1996.

[11] Y.el Fattah y C. Holzbaur. A CLP approach to detection and identification of control system component failures. *Proceedings of the Workshop on Qualitative and Quantitative Approaches to Model-Based Diagnosis, Second International Conference on Intelligent Systems Engineering*, pag. 97–107, 1994.

[12] R. M. Gasca, C. Del Valle, R. Ceballos, y M. Toro. An integration of FDI and DX approaches to polynomial models. *DX-2003,*

14th International Workshop on Principles of Diagnosis, pag. 153–158, Junio 2003.

[13] R. M. Gasca, J. A. Ortega, y M. Toro. Diagnosis basada en modelos polinómicos usando tecnicas simbólicas. *Monografía: Diagnosis, Razonamiento Cualitativo y Sistemas Socioeconómicos; Revista Iberoamericana de Inteligencia Artificial*, pag. 68–77, 2001.

[14] M. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[15] A. Gotlieb, B. Botella, y M. Rueher. A CLP framework for computing structural test data. *Computational Logic*, pag. 399–413, 2000.

[16] U. Heller y P. Struss. G^+DE - The generalized diagnosis engine. *DX-2001, 12th International Workshop on Principles of Diagnosis*, pag. 79–86, 2001.

[17] ILOG. *ILOG Solver 4.4, User's Manual*. 1999.

[18] R. Iserman. Supervision, fault detection and fault diagnosis an introduction. *Control Engineering Practice*, 5(5):639–652, 1997.

[19] B. Meyer. Applying 'design by contract'. *IEEE Computer*, 25(10):40–51, Octubre 1992.

[20] R. J. Patton y J. Chen. A review of parity space approaches to fault diagnosis. *IFAC-SAFEPROCESS Symposium*, 1991.

[21] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1), 1987.

[22] D. Sabin, M. Sabin, R. D. Russell, y E. C. Freuder. A constraint-based approach to diagnosing software problems in computer networks. *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pag. 463–480, 1995.