Descripción Técnica del Proyecto

Maximiliano Cristiá

Título del Proyecto: Aplicación de programación lógica de restricciones conjuntistas a la validación y verificación de software

Unidad Ejecutora: Centro Internacional Franco–Argentino de Ciencias de la Información y de Sistemas (CIFASIS)

Institución Beneficiaria: CONICET

Investigador Responsable: Maximiliano Cristiá

Teléfono: (0341) 4237248 interno 310

Correo Electrónico: `cristia@cifasis-conicet.gov.ar`

1. Objetivos del Proyecto

El objetivo fundamental que persigue la línea de investigación en la que se inscribe este proyecto es la automatización de la validación y verificación (V&V) funcional de software tanto como sea posible.

El fundamento de este objetivo es esencialmente económico: la V&V es la actividad con mayores costos relativos en la producción de software de calidad [51] [19, página 20] [12, página 88] [60, página 157 en versión en castellano] [57] [64, tabla ES-1 en página ES-5]. Por otro lado, el estado general de la práctica de la V&V en la industria del software revela que es una actividad que rara vez se realiza rigurosamente. Pensamos que automatizando actividades de V&V la industria comenzará a mejorar sus prácticas en ese sentido.

Una estrategia para automatizar actividades de V&V es partiendo de especificaciones formales. Sin embargo, la industria rara vez escribe tales especificaciones. Uno de los motivos radica en el costo aparente que conlleva escribirlas. Este proyecto quiere mostrar que escribir especificaciones no es un costo sino una inversión porque partiendo de ellas se pueden automatizar tareas que de otra forma es imposible hacer. En este sentido, pretendemos mostrar que es económicamente conveniente cambiar el "costo" de escribir una especificación por el "costo" de generar prototipos y casos de prueba.

Entonces, el primer problema que abordaremos es la *generación automática de prototipos a partir de especificaciones basadas en conjuntos* (en el sentido dado por notaciones tales como B [9], Event-B [10], Z [69], VDM [52], TLA [54], etc.). En general la literatura de la Ingeniería

de Software propone y enfatiza la necesidad de construir prototipos lo antes posible durante el proceso de desarrollo. Sin embargo, no son muy utilizados en la industria debido al costo que implica desarrollarlos. Al finalizar este proyecto se contará con una herramienta capaz de generar automáticamente prototipos partiendo de una especificación basada en conjuntos.

El segundo problema en el cual trabajaremos es la *generación automática de casos de prueba a partir de especificaciones basadas en conjuntos*. Es decir que con la misma especificación se podrán generar automáticamente prototipos para validar requerimientos y casos de prueba para testear la implementación. O sea que, reiteramos, el "costo" de la especificación se cambia por el "costo" de los prototipos y el testing.

La teoría de conjuntos ha probado ser una herramienta fundamental para la especificación de una amplísima variedad de sistemas de software, en particular los llamados sistemas de información que son los más desarrollados en nuestro país (sistemas de reservas, administración de empresas, contabilidad, liquidación de impuestos, comercio electrónico, etc.). De aquí la elección de este tipo de especificaciones. Por otro lado, tanto la generación de prototipos como de casos de prueba a partir de este tipo de especificaciones requiere la *resolución de restricciones conjuntistas*. Por este motivo, nuestro grupo se ha asociado desde hace dos años con un grupo de la Universidad de Parma (Italia) liderado por Gianfranco Rossi (incluido en el grupo colaborador). El grupo de Rossi trabaja en el desarrollo teórico-práctico de la *unificación conjuntista* y la *programación de restricciones conjuntistas* desde hace muchos años [35][36][37]. Resultados recientes publicados en conjunto muestran que las técnicas desarrolladas por Rossi proveen excelentes resultados cuando se trabaja con especificaciones basdas en conjuntos [31].

2. Objetivos Específicos e Hipótesis de Trabajo

La hipótesis de trabajo de este proyecto es la siguiente.

> La programación de restricciones conjuntistas es una herramienta fundamental para la generación automática de prototipos y de casos de prueba funcionales, si se parte de especificaciones basadas en teoría de conjuntos.

En este proyecto nos planteamos los siguientes objetivos específicos:

1. Automatizar la generación de prototipos funcionales a partir de especificaciones basadas en teoría de conjuntos.
2. Utilizar la programación de restricciones conjuntistas como base para la generación automática de prototipos.
3. Utilizar la programación de restricciones conjuntistas como base para la generación de casos de prueba.
4. Extender la programación de restricciones conjuntistas a funciones parciales, funciones totales, relaciones binarias y listas.
5. Integrar la programación de restricciones conjuntistas a clp(Q,R).

6. Validar empíricamente las soluciones propuestas.

Un programa es correcto si verifica su especificación. Pero la especificación puede no ser una representación fidedigna de los requerimientos. Una forma de lograr especificaciones correctas es que el usuario interactúe con un prototipo generado a partir de ellas. En consecuencia, el usuario puede corregir o aceptar el prototipo, en cuyo caso o bien se corrige la especificación o se la considera correcta. Una vez que el usuario aprueba el prototipo es razonable comenzar con la implementación definitiva, partiendo de la especificación que (ahora) se sabe correcta. Luego, la especificación puede usarse para verificar la implementación por medio de testing (basado en especificaciones). Por lo tanto, la especificación sirve, al menos, para validar los requerimientos y para testear la implementación. Esto muestra que, si se cuenta con las herramientas adecuadas, las especificaciones habilitan o automatizan actividades que de otra forma serían muy costosas o imposibles.

Notar que, si se parte de especificaciones basadas en teoría de conjuntos, tanto la generación automática de prototipos como la de casos de prueba están unidas por la necesidad de resolver restricciones conjuntistas. En efecto, un caso de prueba es esencialmente la solución a un predicado sobre la teoría de conjuntos; y la ejecución de una operación sobre un prototipo requiere resolver una restricción conjuntista (básicamente la especificación de la operación en cuestión). Es por este motivo que pensamos atacar ambos problemas en el mismo proyecto.

3. Relevancia del Problema

Como indicamos en la Sección 1 la automatización de las actividades de V&V es clave para lograr un aumento de la calidad del software sin aumentar proporcionalmente los costos de producción. La industria del software local necesita de estas herramientas y de la transferencia tecnológica y asistencia técnica correspondientes. Por consiguiente es conveniente que estas tecnologías sean desarrolladas localmente. La industria es muy renuente a escribir especificaciones formales en parte debido al costo aparente que esto implica. Mostrar que, dada una especificación, es posible realizar tareas que aumentan considerablemente la calidad del producto final sin incrementar proporcionalmente los costos, es importante para la industria.

Como ya hemos mencionado, el hilo conductor que une a los dos problemas que nos interesa abordar en este proyecto es la programación de restricciones conjuntistas. En particular en este proyecto nos focalizaremos en la *programación lógica de restricciones conjuntistas* (*constraint logic programming with sets*, en inglés) (CLP-S). La CLP-S ha sido ampliamente estudiada por diversos autores desde los años setenta del siglo XX a partir de los trabajos seminales de Jacob T. Schwartz sobre el lenguaje de programación SETL [66]. Pocos años más tarde, Eugenio Omodeo estudió con Schwartz el problema de encontrar procedimientos de decisión para la teoría de conjuntos [39]. A su vez Omodeo trabajó en Italia junto a, entre otros, Gianfranco Rossi en la definición de un lenguaje de programación lógica basado en conjuntos [34]. Este lenguaje definido por Dovier, Omodeo, Pontellli, Rossi y otros se denomina {*log*} (y se lee 'setlog'). Este sistema está desarrollado en Prolog y actualmente es mantenido por Rossi [62]. {*log*} está basado en la unificación conjuntista definida por el mismo grupo [38]. Esta unificación permite operar simbólicamente con conjuntos lo que en general no es posible con otros métodos. Por ejemplo, las codificaciones de la teoría de conjuntos en los lenguajes soportados por solventes SMT (*SMT solvers*, en inglés) reducen los operadores

conjuntistas a cuantificaciones universales sobre arreglos, listas o funciones no interpretadas [32][15]. Si bien estas codificaciones han probado ser útiles para la prueba automática de teoremas, según nuestros experimentos no resultan tan convenientes para resolver restricciones [26].

{*log*} admite una variedad de conjuntos y expresiones conjuntistas que otras herramientas similares no pueden procesar adecuadamente. En particular {*log*} admite conjuntos parcialmente definidos (es decir, conjuntos donde algunos de sus elementos son variables), conjuntos definidos intencionalmente, predicados definidos por el usuario, cuantificadores universales restringidos, etc. En general la clase de conjuntos soportada por {*log*} se denomina conjuntos hereditariamente finitos (*hereditarily finite sets*, en inglés). Otros sistemas similares tales como Conjunto [41], Gödel [48], o el lenguaje lógico para bases de datos definido por Beeri y otros [13] no soportan todo los elementos mencionados. Sistemas más recientes como Escher [56], Oz/Mozart [63] y ECL^{i}PS^e [11] están basados en los anteriores o continúan sin soportar la clase general de conjuntos admitida por {*log*}. Otras propuestas muy interesantes como por ejemplo CLPS-B del equipo de Bruno Legeard [14] o ProB del equipo de Michael Leuschel [55], no cuentan, sin embargo, con el soporte matemático debidamente formalizado y probado como lo hace el equipo de Rossi. En particular ProB no está basado en la unificación conjuntista sino más bien en resolver todos los términos deterministas al inicio, lo que implica cierta incapacidad para tratar expresiones no-deterministas.

Sin embargo, {*log*} presenta algunas limitaciones (compartidas por todos los otros sistemas que hemos mencionado). Si bien actualmente {*log*} soporta funciones parciales y totales, listas y relaciones binarias lo hace solo como predicados definidos por el usuario por medio de conjuntos. Es decir, estas estructuras matemáticas no son objetos de primera clase como lo son los conjuntos. Esto produce, en principio, una merma en las capacidades del solvente de restricciones (*constraint solver*, en inglés) pues debe operar al nivel de conjuntos. Por otro lado, si bien {*log*} admite restricciones sobre los números enteros utiliza como solvente a clp(FD) [20], el cual no es capaz de resolver algunas restricciones relativamente simples. En tercer lugar, dado que nuestro objetivo es trabajar con un lenguaje de especificación basado en conjuntos que sea lo suficientemente expresivo como para poder describir la funcionalidad de una amplia gama de sistemas, es necesario que el lenguaje de especificación admita funciones parciales y totales, listas, relaciones binarias y expresiones algebraicas complejas (sobre $\mathbb{Z}$, $\mathbb{Q}$ y $\mathbb{R}$). Más aun, las herramientas que se construyan para trabajar sobre este lenguaje deben ser capaces de resolver restricciones complejas sobre esas teorías. En consecuencia, durante este proyecto planeamos extender {*log*} de manera tal que soporte adecuadamente ese tipo de estructuras matemáticas y se integre con clp(Q,R) [50] en lugar de clp(FD).

Entre los varios métodos propuestos en relación a la prototipación nuestro trabajo se concentra en lo que se denomina *prototipación rápida*. Es decir, en la posibilidad de generar prototipos a muy bajo costo y de forma permanente con respecto a la aparición de los requerimientos del usuario. En particular, los prototipos en que pensamos entran dentro de la categoría de *prototipos desechables*. Es decir, son los prototipos que sirven para validar una idea pero que una vez que esta fue validada el prototipo se descarta y se comienza a implementar el sistema definitivo. Finalmente, circunscribimos nuestro trabajo a *prototipos funcionales*. En otras palabras, los prototipos no serán útiles para validar cualidades tales como desempeño, portabilidad, modificabilidad, disponibilidad, etc. O sea que en lo referente a prototipos el proyecto se focaliza en *prototipación funcional rápida desechable*. En

este sentido la prototipación que planteamos es rápida porque se genera *automáticamente* partiendo de una especificación basada en teoría de conjuntos. Y es desechable en tanto los prototipos así generados solo sirven para validar los requerimientos del usuario y, en consecuencia, la especificación. No creemos que sea conveniente utilizar estos prototipos como primera versión del sistema final puesto que su código fuente estará basado en la unificación conjuntista la cual no es necesariamente muy eficiente como para ser utilizada en un sistema en producción. Además, el código fuente reflejará la estructura de la especificación funcional la cual no necesariamente sigue conceptos de diseño y arquitectura de software tales como ocultación de información, modularización, patrones de diseño, estilos arquitectónicos, etc. En consecuencia, el código resultante no tendrá, necesariamente, bajo costo de mantenimiento y flexibilidad. Por otra parte, si los prototipos van a ser utilizados para validar requerimientos, entonces los usuarios finales deben poder utilizarlos casi como utilizarían el sistema real. Esto es posible solo si los prototipos incluyen interfaces de usuario más o menos similares a las que utilizan los sistemas reales. Por este motivo el proyecto contempla la necesidad de que los prototipos no solo sean capaces de ejecutar la funcionalidad del sistema sino que también incluyan interfaces de usuario realistas.

Existen varias propuestas para generar prototipos partiendo de especificaciones basadas en conjuntos. En particular existen trabajos que proponen traducir ese tipo de especificaciones a programas Prolog [33][71][18][45][42][68][80][46][84][43][61][82][81][16][44]. Sin embargo, casi todos estos trabajos traducen los conjuntos a listas de Prolog y definen una serie de operadores conjuntistas que operan sobre estas listas. Es decir que casi ninguna de estas propuestas usufructúa los beneficios de la implementación de una teoría que permite operar simbólicamente con conjuntos. Trabajos preliminares realizados entre nuestro grupo y Gianfranco Rossi probarían que el uso de un sistema como $\{log\}$ ofrece importantes ventajas con respecto a estas otras propuestas [31][30]. Los trabajos que sí utilizan lenguajes basados en conjuntos [16, 61, 44] carecen de una formalización matemática, como mencionamos más arriba. Más aun, prácticamente ninguno de estos trabajos genera prototipos o anima una especificación de forma tal que incluyan interfaces de usuario similares a las reales. Nuestros experimentos preliminares mostrarían que esto es posible si a la especificación funcional se la acompaña con una simple especificación de la interfaz de usuario y su relación con la funcional [30]. Sterling y otros [71] apuntan a un objetivo similar pero ellos proponen un compilador Z-a-Prolog que genera programas que se animan mediate una línea de comandos. B-Motion [67] y Brama [53] son dos herramientas basadas en el método B que van en la misma dirección que proponemos nosotros. B-Motion se basa en ProB. Pero, como ya dijimos, ProB no se basa en la unificación conjuntista la cual, creemos, es clave para el problema en cuestión.

En lo concerniente a la generación de casos de prueba a partir de especificaciones basadas en conjuntos, apuntamos a utilizar también $\{log\}$ como generador de casos de prueba. La generación de casos de prueba a partir de especificaciones entra dentro del área denominada *testing basado en modelos* (*model-based testing*, en inglés) (MBT) [47][79][64]. Entornos de ejecución de casos de prueba como JUnit [2] automatizan solo las actividades más simples, dejando, en lo esencial, el trabajo analítico a los ingenieros [58]. En particular, estos entornos: (a) no generan casos de prueba automáticamente, sino que es el desarrollador quien debe determinar cuántos y cuáles casos se ejecutarán; y (b) los desarrolladores son quienes deben indicar el comportamiento esperado para cada caso de prueba (en otras palabras, no se proveen oráculos de forma automática).

La mayoría de los trabajos en el área de MBT utilizan alguna forma de máquina de estado finito para construir los modelos a partir de los cuales se extraen los casos de prueba [47, 59]. Si bien estos modelos son relativamente fáciles de construir, en general carecen del poder expresivo suficiente como para especificar programas o sistemas complejos. Esta es una de las razones fundamentales para haber elegido especificaciones basadas en conjuntos. Existen varias herramientas que automatizan diferentes métodos de MBT [1][7][4][3][6][8][5] pero solo FASTEST, producida por nuestro grupo, lo hace para especificaciones basadas en conjuntos mediante el método de MBT llamado Test Template Framework (TTF) [72][24]. Las herramientas más cercana son Smartesting [79] y ProTest [65] (basada en ProB) pues utilizan especificaciones B, aunque no aplican el TTF. Esta diferencia no es trivial puesto que el TTF permite posibilidades de testing que otros métodos no pueden proveer; el TTF es una teoría general de MBT basada en partición de dominios. Por ejemplo, algunas de sus ventajas son: no requiere de la generación de un autómata para dirigir el testing y ofrece varias reglas para generar casos de prueba tales que cubren la especificación (y en consecuencia la implementación).

Sin embargo, una de las desventajas del TTF es la dificultad para generar casos de prueba de forma automática. En efecto, en el TTF un caso de prueba es una solución a un predicado sobre la teoría de conjuntos, lo que en general es un problema indecidible. Por consiguiente, es necesario contar con una herramienta, que en la práctica, sea capaz de determinar si un tal predicado es satisfacible, y en tal caso retornar una solución, o es insatisfacible. Hasta el momento FASTEST utilizaba métodos desarrollados dentro del grupo para realizar estas tareas [24]. No obstante, estos métodos resultan poco prácticos cuando los predicados son muy complejos. Por este motivo, desde hace un par de años venimos trabajando con Rossi en la posibilidad de utilizar $\{log\}$ como generador de casos de prueba para FASTEST. Hasta el momento hemos obtenido resultados preliminares que nos animan a continuar por este camino [31]. Por otra parte, estos resultados necesitan ser extendidos. Por ejemplo, $\{log\}$ muestra ciertas limitaciones a la hora de determinar la insatisfacibilidad de predicados que incluyen funciones parciales o relaciones binarias. Creemos que la raíz de este problema yace en el hecho de que las funciones parciales y las relaciones binarias no son objetos de primera clase en $\{log\}$. Al mismo tiempo, $\{log\}$ presenta dificultades a la hora de encontrar soluciones a predicados que incluyen restricciones sobre los números enteros. Esto se debería al hecho de que $\{log\}$ delega estos predicados en clp(FD) el cual presenta limitaciones. Por estos motivos, es que nos proponemos trabajar en los objetivos específicos 4 y 5 enunciados en la sección 2.

4. Resultados Preliminares y Aportes del Grupo al Estudio del Problema en Cuestión

Nuestro grupo viene trabajando con especificaciones basadas en conjuntos desde 2007 (concretamente utilizamos la notación Z). Comenzamos desarrollando FASTEST como implementación del TTF. El desarrollo de esta herramienta implica la solución de varios problemas complejos:

- Definición e implementación de un mecanismo para la aplicación de tácticas de testing. Las tácticas de testing son reglas generales que permiten particionar el espacio de

entrada de una operación.

- Definición e implementación de un método para satisfacción de predicados sobre la teoría de conjuntos.

  Concretamente, implementamos un algoritmo que calcula un modelo finito para un predicado dado, evalúa el predicado en cada uno de los elementos del modelo (utilizando ZLive de CZT [40]) y termina cuando se encuentra un elemento que satisface el predicado o cuando el modelo finito se agota. Es este algoritmo el que pensamos mejorar al introducir $\{log\}$ como algoritmo de satisfacción de predicados de la teoría de conjuntos.

- Definición e implementación de un método, extensible por el usuario, muy simple pero efectivo y eficiente, para encontrar predicados insatisfacibles [25].

  Este método complementa al algoritmo mencionado en el punto anterior puesto que al aplicar el TTF se tiende a generar un gran número de predicados insatisfacibles. El método se basa en una biblioteca de *teoremas de eliminación* extensible por el usuario. Cada teorema de eliminación representa una familia de contradicciones. Cuando Fastest analiza una condición de test la compara con cada teorema de eliminación y si hay concordancia la elimina.

  Al igual que en el caso anterior creemos que la introducción de $\{log\}$ permitirá mejorar este método puesto que el sistema de Rossi es capaz de detectar predicados insatisfacibles.

- A su vez, validamos FASTEST aplicándolo a diez casos de estudio, algunos de los cuales son problemas reales de la industria [24][23]. Esto nos permitió confirmar, al menos experimentalmente, que el TTF puede ser automatizado como cualquier otro método de MBT. En particular FASTEST realiza automáticamente, en promedio, el 80 % del trabajo de generación de casos de prueba.

  Con la inclusión de $\{log\}$ esperamos poder mejorar ese porcentaje y reducir los tiempos de cálculo.

  Algunos de los casos de estudio son el resultado de la interacción del grupo con investigadores de otras instituciones tales como Instituto Nacional de Pesquisas Espacias[1] (INPE, Brasil), el Instituto de Aeronáutica e Espaço[2] (IAE, Brasil), INVAP[3] de Argentina y The Open University (Gran Bretaña).

- También trabajamos en la traducción automática de casos de prueba a lenguaje natural [29].

  Esta necesidad surge a raíz de que en muchos proyectos ciertas actividades de V&V las realizan revisores externos que no necesariamente comprenden notaciones formales. Por lo tanto, ellos deben tener acceso a descripciones en lenguaje natural de los casos de prueba que se han ejecutado.

[1] `www.inpe.br`
[2] `www.iae.cta.br`
[3] `www.invap.com.ar`

- Luego extendimos el TTF y FASTEST a la concretización de casos de prueba [27].

  Como la mayoría de los métodos de MBT el TTF genera *casos de prueba abstractos*. Es decir, casos de prueba escritos en la misma notación del modelo del cual se extraen. Esto significa que no necesariamente son ejecutables directamente sobre la implementación. Por lo tanto, fue necesario desarrollar un método que automatiza la transformación de los casos de prueba abstractos en casos de prueba que son directamente ejecutables sobre la implementación.

- Recientemente extendimos el TTF a testing de integración [28].

  Como mencionamos más arriba, el TTF fue definido esencialmente para testing de unidad. Por otro lado, la literatura de Ingeniería de Software establece que el testing de unidad continúa en el testing de integración. Es decir, que las unidades de implementación se integran una a una y cada uno de los sucesivos incrementos debe ser testeado.

- En la actualidad trabajamos en la posibilidad de anotar la implementación con especificaciones basadas en conjuntos de forma tal que mejorar aun más la automatización de todo el proceso de testing.

Desde hace aproximadamente dos años el grupo mantiene una colaboración con Gianfranco Rossi de la Universidad de Parma (Italia). Rossi es el desarrollador principal de $\{log\}$. A raíz de esta colaboración hemos obtenido resultados preliminares interesantes en relación a los temas que se pretenden abordar en este proyecto. Por un lado, hemos definido e implementado una primera versión de un traductor entre FASTEST y $\{log\}$ de forma tal que este último actúa como generador de casos de prueba del primero [31]. Por el otro, recientemente presentamos en el 1st International Workshop about Sets and Tools (SETS 2014), algunos prototipos generados a partir de especificaciones Z[4]. Si bien estos prototipos no fueron generados automáticamente todo indica que esto es razonablemente posible. En particular estos prototipos se implementan sobre una combinación de $\{log\}$, Prolog y XPCE [83]. Por otra parte, estos prototipos generados presentan una interfaz de usuario realista. Esto lo logramos definiendo un sencillo lenguaje que permite indicar los elementos de la interfaz de usuario y cómo estos se relacionan con la especificación funcional. Obviamente el proyecto intentará generalizar y extender estas ideas iniciales.

Si bien hasta el momento hemos aplicado todos nuestros resultados a la notación Z, ninguno de ellos depende completamente de aquella. Por lo tanto, los resultados que hemos obtenido hasta el momento se pueden aplicar a otras notaciones similares tales como B, Event-B, VDM, TLA, etc., puesto que todas ellas se basan en teorías de conjuntos similares (aunque en lógicas diferentes). De hecho la teoría de conjuntos soportada por $\{log\}$ es no tipada e impone muy pocas restricciones sobre los conjuntos que se pueden formar. Por estos motivos, el proyecto se presenta sin una mención específica a una notación particular (como Z, por ejemplo).

[4]Nos parece conveniente resaltar que SETS 2014 fue organizado por y participaron los principales referentes internacionales en el tema. Por ejemplo, participaron del comité de programa, entre otros, Rossi, Michael Leuschel, Stepan Merz y Mark Utting.

Por otro lado, el grupo tiene experiencia en la vinculación entre métodos formales del área de Ingeniería de Software y métodos de simulación ampliamente utilizados en el área de modelado de sistemas a eventos discretos. Esto no da la posibilidad de aplicar en este proyecto técnicas que se utilizan en otras áreas aledañas. En [70, 77, 76, 78, 73, 74, 75] hemos analizado la relación entre DEVS [85] y Z; en tanto que en [21, 22] se analiza la relación entre DEVS y TLA [54]; en [49] aplicamos técnicas de testing basado en modelos a la validación de modelos DEVS por medio de simulación.

5. Tipo de Diseño de Investigación y Métodos

En esta sección describiremos con más detalle cómo pensamos abordar cada uno de los objetivos específicos listados en la sección 2.

5.1. Automatizar la generación de prototipos funcionales a partir de especificaciones basadas en teoría de conjuntos

Esta actividad requiere la definición e implementación de un compilador de un lenguaje de especificación basado en conjuntos y un lenguaje de especificación de interfaces de usuario en el lenguaje de entrada de una herramienta capaz de resolver restricciones conjuntistas y capaz de ejecutar programas generales.

El lenguaje de especificación basado en conjuntos se utilizaría para especificar la funcionalidad del sistema a prototipar (en otras palabras la especificación de los requerimientos funcionales).

El lenguaje de especificación de interfaces de usuario se utilizaría, obviamente, para especificar la interfaz de usuario del prototipo y, fundamentalmente, para establecer la relación entre las variables de la especificación funcional y los elementos de la interfaz de usuario. Se pueden ver ejemplos de lo que decimos en `https://www.dropbox.com/s/rgub9d3i10coht8/sets2014-examples.tar.gz`. De todas formas un ejemplo simple es el siguiente: supongamos que $usr : DNI \pfun DOMICILIO$ es una función parcial que asocia el DNI de una persona con su domicilio y supongamos que existe una operación que debe tomar un DNI como entrada y mostrar el domicilio correspondiente. Una forma de leer el DNI de entrada es con un simple cuadro de texto pero otra forma más realista sería que el usuario pudiera elegir el DNI en una lista con los DNIs presentes en el sistema en ese momento. En este último caso esa lista debería contener el resultado de $\mathrm{dom}\, usr$. Por lo tanto, la variable de entrada de la operación a nivel de la especificación, digamos $u?$, debería asociarse con una lista desplegable en cierto formulario la que a su vez se llena, cuando es presentada, con el valor de $\mathrm{dom}\, usr$ en ese momento.

En definitiva, esta actividad implica definir o seleccionar el lenguaje de especificación funcional, el lenguaje de especificación de interfaces de usuario y el o los lenguajes de programación que fungirán de lenguaje objetivo del compilador.

5.2. Utilizar la programación de restricciones conjuntistas como base para la generación automática de prototipos

Más allá de que en la sección anterior dejamos abierta la elección de la tecnología de programación que fungirá de lenguaje objetivo del compilador, la idea es utilizar la combinación que ya hemos estado probando, es decir $\{log\}$ +Prolog+XPCE [30].

Esencialmente, $\{log\}$ se utilizará para implementar la funcionalidad del prototipo, XPCE [83] se utilizará para implementar la interfaz de usuario y Prolog para integrar las dos partes anteriores. Un de los conceptos fundamentales detrás de esta elección es ver a Prolog como un lenguaje de programación de propósito general y no exclusivamente como un lenguaje para problemas vinculados a la Inteligencia Artificial[5].

Como la especificación funcional estará expresada en base a conjuntos resulta natural traducirla (o implementarla) a un lenguaje donde los conjuntos y sus operaciones son elementos de primera clase, es decir $\{log\}$. Entonces, por ejemplo, un predicado de la forma $act' = act \cup \{x?\}$ se traduce como un programa de la forma:

```
b_getval(act,A), setlog(un(A,{x},Anew)), nb_setval(act,Anew)
```

donde el segundo término es una llamada a $\{log\}$ que implementa la unión entre act y $\{x?\}$ en tanto que `b_getval` y `nb_setval` son predicados Prolog que permiten gestionar variables de estado.

La especificación de la interfaz de usuario se traducirá a programas XPCE, que es una implementación Prolog de los principales elementos de las interfaces gráficas. Por lo tanto, XPCE posee primitivas para definir marcos, ventanas, cuadros de diálogo, botones, menús, listas desplegables, cuadros de texto, etc. Además habrá código Prolog que vinculará el programa XPCE con el programa $\{log\}$.

Queremos remarcar que estas ideas requieren la implementación de un compilador capaz de generar de forma automática los prototipos.

5.3. Utilizar la programación de restricciones conjuntistas como base para la generación de casos de prueba

Esta sección refiere a utilizar $\{log\}$ como generador de casos de prueba para Fastest. Como mencionamos más arriba, ya contamos con algunos resultados preliminares al respecto [31]. De todas formas restan varios problemas que aun deben resolverse.

Hasta el momento la integración entre $\{log\}$ y Fastest se realizó a través de un traductor entre Z y el lenguaje de $\{log\}$. Este traductor no es completo puesto que algunas construcciones de Z no fueron tenidas en cuenta. Además, deberá ser modificado cuando la actividad descripta en la siguiente sección se haya finalizado. Por otra parte, la traducción definida es una de las tantas posibles por lo que sería conveniente probar otras alternativas.

Otro problema que aun no hemos abordado está relacionado con la capacidad de $\{log\}$ para eliminar condiciones de test insatisfacibles. Pruebas manuales preliminares nos indican que $\{log\}$ no superaría al método utilizado por Fastest sino que serían complementarios. Esto implicaría que sería conveniente implementar el método de Fastest en Prolog. Una de

[5]Este es un concepto sostenido por Rossi.

las razones para hacerlo es que, por las características del método de eliminación implementado por FASTEST, su implementación en Prolog sería considerablemente más simple y por lo tanto más simple de verificar.

5.4. Extender la programación de restricciones conjuntistas a funciones parciales, funciones totales, relaciones binarias y listas

Esta actividad involucra dos tareas. La primera es definir el lenguaje con el cual trabajará el solvente y la segunda es probar que ese lenguaje (más el lenguaje actual de $\{log\}$) es consistente y completo. En este sentido se sigue el mismo camino que recorre habitualmente la comunidad CLP en problemas similares.

El lenguaje que se defina tiene que ser el mínimo posible porque de otro modo la demostración de su consistencia y completitud puede resultar extremadamente compleja. En general se trata de encontrar un conjunto de operadores tales que el resto de los que se necesitan para que el usuario pueda expresar problemas complejos, se escriba en función de los primeros. Este conjunto mínimo de operadores se implementa como *vínculos irreducibles*. Por ejemplo, para funciones parciales se puede pensar en cuatro vínculos irreducibles: $pfun(f)$, $dom(f, d)$, $ran(f, r)$ y $apply(f, x, y)$. De esta forma, otros operadores tales como la restricción de dominio, se pueden escribir en términos de los vínculos irreducibles. En definitiva, se logra un lenguaje muy expresivo tal que la demostración de su consistencia y completitud es relativamente simple.

También se tendrá en cuenta que algunas de las estructuras matemáticas a contemplar pueden ser escritas en términos de otras. Por ejemplo, las listas o secuencias finitas son un subconjunto de las funciones parciales:

$$\operatorname{seq} X = \{f : \mathbb{N}_1 \pfun X \mid \exists\, n : \mathbb{N} \bullet \operatorname{dom} f = 1 \mathinner{..} n\}$$

por lo tanto, en algunos casos, podría ser suficiente agregar muy pocos vínculos irreducibles o especializar los ya existentes.

5.5. Integrar la programación de restricciones conjuntistas a clp(Q,R)

Esta actividad refiere a que $\{log\}$ no delegue la resolución de las restricciones algebraicas[6] en clp(FD) como lo hace actualmente, sino en clp(Q,R). clp(Q,R) cumple una función similar a la de clp(FD) solo que es capaz de resolver más problemas que aquel. clp(Q,R) resuelve ecuaciones lineales sobre variables racionales o reales, cubre el tratamiento de ecuaciones no lineales, incluye un algoritmo de decisión para inecuaciones lineales que detecta ecuaciones implicadas, elimina redundancias, elimina cuantificadores, permite disecuaciones lineales y provee optimización lineal.

El reemplazo de clp(FD) por clp(Q,R) en $\{log\}$ tiene un impacto importante para los dos problemas que queremos investigar en este proyecto. Hemos detectado no pocas restricciones para las cuales $\{log\}$ no es capaz de encontrar una solución o determinar que son insatisfacibles debido a la presencia de restricciones algebraicas que clp(FD) no es capaz de resolver,

[6] Nos referimos a términos que incluyen uno o más de los símbolos $>, <, =, \neq, \leq, \geq, +, -, *, /$ con el significado habitual en la matemática clásica.

y no a la presencia de restricciones conjuntistas complejas. Es decir, la limitación yace en clp(FD) y no en clp($\mathcal{SET}$) (la teoría de restricciones que subyace a {*log*}).

La integración de {*log*} con clp(Q,R) no solo mejorará las capacidades de {*log*} sino que habiendo definido los tipos $\mathbb{Q}$ y $\mathbb{R}$ en notaciones como Z y B será posible traducirlos a {*log*} y operar con ellos.

La integración se hará de manera similar a la forma en que {*log*} actualmente trabaja con clp(FD).

5.6. Validar empíricamente las soluciones propuestas

Aplicar a casos de estudio, de creciente complejidad hasta llegar a problemas reales de la industria, los métodos que hemos definido para los problemas que ya hemos resuelto no solo nos permitió validar nuestras soluciones sino que, fundamentalmente, nos dio una idea de las limitaciones y falencias de aquellos. Por lo tanto, tomamos como parte inherente de nuestra tarea aplicar los nuevos métodos que definamos en las etapas anteriores a casos de estudio y medir su utilidad, complejidad, aplicabilidad, etc.

6. El grupo de trabajo

El grupo de trabajo está compuesto por los siguientes miembros:

- Maximiliano Cristiá - Investigador responsable - Jefe del grupo de Ingeniería de Software de CIFASIS; profesor asociado de la Universidad Nacional de Rosario.
- Laura Pomponio - Grupo de responsables del proyecto - Becaria posdoctoral de CONICET en el grupo de Ingeniería de Software de CIFASIS bajo supervisión de M. Cristiá.
- Pamela Viale - Grupo de responsables del proyecto - Becaria posdoctoral de CONICET en el grupo de Ingeniería de Software de CIFASIS bajo supervisión de M. Cristiá.
- Gianfranco Rossi - Grupo de colaboradores - Profesor titular de la Universidad de Parma (Italia); experto en unificación conjuntista y programación lógica de restricciones basadas en conjuntos; desarrollador de {*log*}.

  Su aporte al proyecto será clave dado que pensamos utilizar {*log*}, y la teoría subyacente, a lo largo del proyecto.
- Claudia Frydman - Grupo de colaboradores - Profesora titula de la Aix-Marseille Université (Francia); experta en modelado y simulación de sistemas a eventos discretos. La Dra. Frydman se encuentra con delegación de tareas en el CIFASIS.

  Su aporte al proyecto consistirá mayormente en el asesoramiento y supervisión general del proyecto en tanto investigadora senior radicada en el CIFASIS. Notar que si bien Rossi realizará aportes técnicos él vive en Italia por lo que un equipo de trabajo de reciente formación como el nuestro necesita de una guía local.

- Diego Hollmann - Grupo de colaboradores - Becario doctoral de CONICET en el grupo de Ingeniería de Software de CIFASIS bajo supervisión de C. Frydman y M. Cristiá.

  Su aporte al proyecto se concretará una vez finalizado su doctorado (fines de 2014) puesto que continuará trabajando como miembro del grupo de Ingeniería de Software del CIFASIS. Realizará tareas de desarrollo avanzado de software y contribuirá en todo lo relacionado con la generación de casos de prueba.

7. Cronograma de Trabajo

Tareas	**Cuatrimestres**								
	1	2	3	4	5	6	7	8	9
Automatizar la generación de prototipos funcionales a partir de especificaciones basadas en teoría de conjuntos			X	X				X	X
Utilizar la programación de restricciones conjuntistas como base para la generación automática de prototipos					X	X	X	X	X
Utilizar la programación de restricciones conjuntistas como base para la generación de casos de prueba					X	X	X	X	X
Extender la programación de restricciones conjuntistas a funciones parciales, funciones totales, relaciones binarias y listas	X	X	X	X	X				X
Integrar la programación de restricciones conjuntistas a clp(Q,R)			X	X	X				X
Validar empíricamente las soluciones propuestas								X	X

Referencias

[1] Conformiq, `http://www.conformiq.com`, last access: November 2011

[2] JUnit.org Resources for Test Driven Development, `http://junit.org/`, last access: November 2011

[3] Model JUnit, `http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/`, last access: November 2011

[4] Rational Statemate, `http://www.ibm.com/developerworks/rational/`, last access: November 2011

[5] Reactis, `http://www.reactive-systems.com`, last access: November 2011

[6] Requirements-Based Automated Verification, `http://www.t-vec.com/solutions/rave.php`, last access: November 2011

[7] Smartesting, `http://www.smartesting.com`, lass access: November 2011

[8] T-vec Tester for Simulink and Stateflow, http://www.t-vec.com/solutions/simulink.php, last access: November 2011

[9] Abrial, J.R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)

[10] Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York, NY, USA, 1st edn. (2010)

[11] Apt, K.R., Wallace, M.: Constraint Logic Programming using ECL^{i}PS^e. Cambridge University Press, New York, NY, USA (2007)

[12] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edn. (2003)

[13] Beeri, C., Naqvi, S.A., Shmueli, O., Tsur, S.: Set constructors in a logic database language. J. Log. Program. 10(3&4), 181–232 (1991)

[14] Bernard, E., Legeard, B., Luck, X., Peureux, F.: Generation of Test Sequences from Formal Specifications: GSM 11-11 Standard Case Study. International Journal of Software Practice and Experience 34(10), 915–948 (2004)

[15] Bobot, F., Filliâtre, J.C., Marché, C., Paskevich, A.: Why3: Shepherd your herd of provers. In: Boogie 2011: First International Workshop on Intermediate Verification Languages. Wrocław, Poland (August 2011), http://proval.lri.fr/submissions/boogie11.pdf

[16] Bouquet, F., Legeard, B., Peureux, F.: CLPS-B - A constraint solver to animate a B specification. STTT 6(2), 143–157 (2004)

[17] Bowen, J.P., Hall, J.A. (eds.): Z User Workshop, Cambridge, UK, 29-30 June 1994, Proceedings. Workshops in Computing, Springer/BCS (1994)

[18] Breuer, P.T., Bowen, J.P.: Towards correct executable semantics for Z. In: Bowen and Hall [17], pp. 185–209

[19] Brooks, Jr., F.P.: The mythical man-month (anniversary ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)

[20] Codognet, P., Diaz, D.: Compiling constraints in clp(FD). J. Log. Program. 27(3), 185–226 (1996)

[21] Cristiá, M.: A TLA+ encoding of DEVS models. In: International Modeling and Simulation Multiconference. pp. 17–22 (2007)

[22] Cristiá, M.: Formalizing the semantics of modular DEVS models with temporal logic. In: 7ème Conférence on Modélisation, Optimisation et Simulation des Systèmes MOSIM 08 (2008)

[23] Cristiá, M., Albertengo, P., Frydman, C.S., Plüss, B., Monetti, P.R.: Applying the Test Template Framework to aerospace software. In: Rash, J.L., Rouff, C. (eds.) SEW. pp. 128–137. IEEE Computer Society (2011)

[24] Cristiá, M., Albertengo, P., Frydman, C.S., Plüss, B., Rodríguez Monetti, P.: Tool support for the Test Template Framework. Softw. Test., Verif. Reliab. 24(1), 3–37 (2014)

[25] Cristiá, M., Albertengo, P., Rodríguez Monetti, P.: Pruning testing trees in the Test Template Framework by detecting mathematical contradictions. In: Fiadeiro, J.L., Gnesi, S. (eds.) SEFM. pp. 268–277. IEEE Computer Society (2010)

[26] Cristiá, M., Frydman, C.: Applying SMT solvers to the Test Template Framework. In: Petrenko, A.K., Schlingloff, H. (eds.) Proceedings 7th Workshop on Model-Based Testing, Tallinn, Estonia, 25 March 2012. Electronic Proceedings in Theoretical Computer Science, vol. 80, pp. 28–42. Open Publishing Association (2012)

[27] Cristiá, M., Hollmann, D., Albertengo, P., Frydman, C.S., Monetti, P.R.: A language for test case refinement in the Test Template Framework. In: Qin, S., Qiu, Z. (eds.) ICFEM. Lecture Notes in Computer Science, vol. 6991, pp. 601–616. Springer (2011)

[28] Cristiá, M., Mesuro, J., Frydman, C.S.: Integration testing in the Test Template Framework. In: Gnesi, S., Rensink, A. (eds.) FASE. Lecture Notes in Computer Science, vol. 8411, pp. 400–414. Springer (2014)

[29] Cristiá, M., Plüss, B.: Generating natural language descriptions of Z test cases. In: Kelleher, J.D., Namee, B.M., van der Sluis, I., Belz, A., Gatt, A., Koller, A. (eds.) INLG. pp. 173–177. The Association for Computer Linguistics (2010)

[30] Cristiá, M., Rossi, G.: Rapid prototyping and animation of Z specifications using {log}. In: 1st International Workshop about Sets and Tools (SETS 2014). p. n/a (2014), affiliated to ABZ 2014

[31] Cristiá, M., Rossi, G., Frydman, C.S.: {log} as a test case generator for the Test Template Framework. In: Hierons, R.M., Merayo, M.G., Bravetti, M. (eds.) SEFM. Lecture Notes in Computer Science, vol. 8137, pp. 229–243. Springer (2013)

[32] Déharbe, D., Fontaine, P., Guyot, Y., Voisin, L.: SMT solvers for Rodin. In: Derrick, J., Fitzgerald, J.A., Gnesi, S., Khurshid, S., Leuschel, M., Reeves, S., Riccobene, E. (eds.) ABZ. Lecture Notes in Computer Science, vol. 7316, pp. 194–207. Springer (2012)

[33] Doma, V., Nicholl, R.A.: EZ: A system for automatic prototyping of Z specifications. In: Prehn, S., Toetenel, W.J. (eds.) VDM Europe (1). Lecture Notes in Computer Science, vol. 551, pp. 189–203. Springer (1991)

[34] Dovier, A., Omodeo, E.G., Pontelli, E., Rossi, G.: {log}: A logic programming language with finite sets. In: Furukawa, K. (ed.) ICLP. pp. 111–124. MIT Press (1991)

[35] Dovier, A., Omodeo, E.G., Pontelli, E., Rossi, G.: A language for programming in logic with finite sets. J. Log. Program. 28(1), 1–44 (1996)

[36] Dovier, A., Piazza, C., Pontelli, E., Rossi, G.: Sets and constraint logic programming. ACM Trans. Program. Lang. Syst. 22(5), 861–931 (2000)

[37] Dovier, A., Piazza, C., Rossi, G.: A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. ACM Trans. Comput. Log. 9(3) (2008)

[38] Dovier, A., Pontelli, E., Rossi, G.: Set unification. Theory Pract. Log. Program. 6(6), 645–701 (Nov 2006), http://dx.doi.org/10.1017/S1471068406002730

[39] Ferro, A., Omodeo, E.G., Schwartz, J.T.: Decision procedures for some fragments of set theory. In: Bibel, W., Kowalski, R.A. (eds.) CADE. Lecture Notes in Computer Science, vol. 87, pp. 88–96. Springer (1980)

[40] Freitas, L., Utting, M., Malik, P., Miller, T.: Community Z Tools (CZT) project, http://czt.sourceforge.net, last access: November 2011

[41] Gervet, C.: Interval propagation to reason about sets: Definition and implementation of a practical language. Constraints 1(3), 191–244 (1997)

[42] Goodman, H.S.: The Z-into-Haskell tool-kit: An illustrative case study. In: Bowen, J.P., Hinchey, M.G. (eds.) ZUM. Lecture Notes in Computer Science, vol. 967, pp. 374–388. Springer (1995)

[43] Grieskamp, W.: A computation model for Z based on concurrent constraint resolution. In: Bowen, J.P., Dunne, S., Galloway, A., King, S. (eds.) ZB. Lecture Notes in Computer Science, vol. 1878, pp. 414–432. Springer (2000)

[44] Hallerstede, S., Leuschel, M., Plagge, D.: Validation of formal models by refinement animation. Sci. Comput. Program. 78(3), 272–292 (2013)

[45] Hasselbring, W.: Animation of Object-Z specifications with a set-oriented prototyping language. In: Bowen and Hall [17], pp. 337–356

[46] Hewitt, M.A., O'Halloran, C., Sennett, C.T.: Experiences with PiZA, an animator for Z. In: Bowen, J.P., Hinchey, M.G., Till, D. (eds.) ZUM. Lecture Notes in Computer Science, vol. 1212, pp. 37–51. Springer (1997)

[47] Hierons, R.M., Bogdanov, K., Bowen, J.P., Cleaveland, R., Derrick, J., Dick, J., Gheorghe, M., Harman, M., Kapoor, K., Krause, P., Lüttgen, G., Simons, A.J.H., Vilkomir, S., Woodward, M.R., Zedan, H.: Using formal specifications to support testing. ACM Comput. Surv. 41(2), 1–76 (2009)

[48] Hill, P.M., Lloyd, J.W.: The Gödel programming language. MIT Press (1994)

[49] Hollmann, D., Cristiá, M., Frydman, C.S.: Adapting model-based testing techniques to DEVS models validation. In: Wainer, G.A., Mosterman, P.J. (eds.) SpringSim (TMS-DEVS). p. 6. SCS/ACM (2012)

[50] Holzbaur, C., Menezes, F., Barahona, P.: Defeasibility in clp(q) through generalized slack variables. In: Freuder, E.C. (ed.) CP. Lecture Notes in Computer Science, vol. 1118, pp. 209–223. Springer (1996)

[51] Jones, C., Bonsignour, O.: The Economics of Software Quality. Addison-Wesley (2011), http://books.google.com.ar/books?id=_t5l5Cn0NBEC

[52] Jones, C.B.: Systematic Software Development Using VDM (2Nd Ed.). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1990)

[53] Ladenberger, L., Bendisposto, J., Leuschel, M.: Visualising event-b models with b-motion studio. In: Alpuente, M., Cook, B., Joubert, C. (eds.) Formal Methods for Industrial Critical Systems, Lecture Notes in Computer Science, vol. 5825, pp. 202–204. Springer Berlin Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-04570-7_17

[54] Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)

[55] Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale B models with ProB. Formal Aspects of Computing 23(6), 683–709 (2011)

[56] Lloyd, J.W.: Programming in an integrated functional and logic language. Journal of Functional and Logic Programming 1999(3) (1999)

[57] McConnell, S.: Code Complete, Second Edition. Microsoft Press, Redmond, WA, USA (2004)

[58] Meyer, B., Fiva, A., Ciupa, I., Leitner, A., Wei, Y., Stapf, E.: Programs that test themselves. Computer 42, 46–55 (Sep 2009), http://portal.acm.org/citation.cfm?id=1638584.1638626

[59] Neto, A.D., Subramanyan, R., Vieira, M., Travassos, G.H., Shull, F.: Improving evidence about software technologies: A look at model-based testing. IEEE Softw. 25(3), 10–13 (2008)

[60] Pfleeger, S.L.: Software Engineering: Theory and Practice. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)

[61] Plagge, D., Leuschel, M.: Validating Z specifications using the ProB animator and model checker. In: Davies, J., Gibbons, J. (eds.) Integrated Formal Methods. Lecture Notes in Computer Science, vol. 4591, pp. 480–500. Springer-Verlag (2007)

[62] Rossi, G.: {*log*} (2008), http://www.math.unipr.it/~gianfr/setlog.Home.html, last access: December 2013

[63] Roy, P.V. (ed.): Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers, Lecture Notes in Computer Science, vol. 3389. Springer (2005)

[64] RTI: The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, National Institute of Standards and Technology, Gaithersburg, MD (May 2002), http://www.nist.gov/director/prog-ofc/report02-3.pdf

[65] Satpathy, M., Leuschel, M., Butler, M.: ProTest: An automatic test environment for B specifications. Electronic Notes in Theroretical Computer Science 111, 113–136 (January 2005)

[66] Schwartz, J.T.: Optimization of very high level languages - ii. deducing relationships of inclusion and membership. Comput. Lang. 1(3), 197–218 (1976)

[67] Servat, T.: Brama: A new graphic animation tool for b models. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007: Formal Specification and Development in B, Lecture Notes in Computer Science, vol. 4355, pp. 274–276. Springer Berlin Heidelberg (2006), http://dx.doi.org/10.1007/11955757_28

[68] Sherrell, L.B., Carver, D.L.: FunZ: An intermediate specification language. Comput. J. 38(3), 193–206 (1995)

[69] Spivey, J.M.: The Z notation: a reference manual. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1992)

[70] Sqali, M., Trojet, W., Torres, L., Frydman, C.: Combining interaction and state based modelling to validate system specification via simulation and formal methods. In: Winter Simulation Conference (WSC 2009) Poster Session. Austin, Texas (descember 2009)

[71] Sterling, L., Ciancarini, P., Turnidge, T.: On the animation of "not executable"specifications by Prolog. International Journal of Software Engineering and Knowledge Engineering 6(1), 63–87 (1996)

[72] Stocks, P., Carrington, D.: A Framework for Specification-Based Testing. IEEE Transactions on Software Engineering 22(11), 777–793 (Nov 1996)

[73] Trojet, W., Frydman, C., el-amine Hamri, M.: Practical application of "lightweight"Z in DEVS framework. In: Proceedings of the 2009 Spring Simulation Multiconference (22 march 2009)

[74] Trojet, W., el-amine Hamri, M., Frydman, C.: Integrating Z in DEVS: a case study lift control system. In: World Congress in Computer Science Computer Engineering and Applied Computing (WORLDCOMP'08) - International Conference on Software Engineering Research and Practice (SERP'08). Las Vegas, Nevada, USA (july 2008)

[75] Trojet, W., el-amine Hamri, M., Frydman, C.: Logical analysis of DEVS models using Z. In: Proceedings of International Simulation Multi-conference (ISMc'08). Edinburgh - Scotland (june 2008)

[76] Trojet, W., Sqali, M., Frydman, C., Torres, L.: MSC scenarios analysis via simulation and formal verification techniques. In: Summer Computer Simulation Conference (GCMS 09), International Simultaion MultiConference (ISmc'09). Istanbul, Turkey (july 2009)

[77] Trojet, W., Sqali, M., Frydman, C., Torres, L., el-amine Hamri, M.: Validating the global behaviour of a system described with scenarios using GDEVS and Z. In: 21th European Modeling and Simulation Symposium (EMSŚ09). Tenerife - Canary Islands, Spain (september 2009)

[78] Trojet, W., Sqali, M., Torres, L., Frydman, C.: Using simulation techniques and formal methods for validating interaction based models. In: The 2009 International Conference on Modeling, Simulation and Visualization Methods (MSV'09) - the 2009 World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'09). as Vegas, Nevada, USA (july 2009)

[79] Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2006)

[80] Valentine, S.H.: The programming language Z−. Information & Software Technology 37(5-6), 293–301 (1995)

[81] Waeselynck, H., Behnia, S.: B model animation for external verification. In: ICFEM. pp. 36–45 (1998)

[82] West, M.M.: The use of a logic programming language in the animation of Z specifications. In: Dahl, V., Niemelä, I. (eds.) ICLP. Lecture Notes in Computer Science, vol. 4670, pp. 451–452. Springer (2007)

[83] Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. TPLP 12(1-2), 67–96 (2012)

[84] Winikoff, M., Dart, P., Kazmierczak, E.: Rapid prototyping using formal specifications. In: In Proceedings of the 21st Australasian Computer Science Conference. pp. 279–294. Springer-Verlag (1998)

[85] Zeigler, B.P., Kim, T.G., Praehofer, H.: Theory of Modeling and Simulation. Academic Press, Inc., Orlando, FL, USA (2000)