VISiR: SOFTWARE DE SOPORTE PARA LA TOMA DE DECISIONES DE VERTIMIENTO DE AGUA EN LA REPRESA DEL ALTO ANCHICAYÁ USANDO PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES

Juan Francisco Díaz Frías*
Camilo Rueda**

RESUMEN

Este artículo presenta VISiR, una aplicación computacional desarrollada para ayudar a la empresa EPSA a resolver el problema de la toma de decisiones de vertimiento de agua en la Represa del Alto Anchicayá. Para su desarrollo se diseñó un modelo computacional del problema utilizando el Paradigma de Programación Concurrente por Restricciones, y se implementó utilizando *Mozart* ([Vh99]) un lenguaje de programación adecuado para este paradigma. El paradigma y el modelo computacional también se describen en este artículo.

* Ph.D. Profesor Titular, Escuela de Ingeniería de Sistemas y Computación, Facultad de Ingeniería
** Camilo Rueda, Profesor Titular, Facultad de Ingeniería, Pontificia Universidad Javeriana

Trabajo realizado en el ámbito de Avispa: Grupo de investigación en Ambientes **VIS**uales de Programación Aplicativa y **GEDI**: Grupo de Estudios Doctorales en Informática

ABSTRACT

In this paper we present VISiR, a software for helping EPSA to make decisions about the spilling of water using the spillway of the Represa del Alto Anchicayá. We made a computational model of the problem using the Concurrent Constraints Programming Paradigm and we implemented it by using the *Mozart* programming language ([VH99]). Both, the paradigm and model, are also described here.

1. INTRODUCCIÓN

Hacia mediados del primer semestre de 1999, el invierno en Colombia, y particularmente en el Valle del Cauca, arreciaba, al punto que los ingenieros responsables en EPSA de la seguridad de la represa del Alto Anchicayá volvían a sentir la necesidad, tiempo atrás expresada, de un *software de ayuda a la toma de decisiones de vertimiento de agua en la represa*.

Por seguridad, la represa se construyó dotada de unas grandes compuertas accionables mecánicamente, que permiten evacuar agua de la misma. En caso que el agua acumulada sobrepase los límites de la represa, ésta se destruiría y la inundación de la región se convertiría en un desastre de grandes proporciones. Las compuertas son, entonces, el mecanismo básico para mantener el agua en niveles de seguridad total.

Por otro lado, el agua es un recurso tan valioso, en este caso para generar energía, que las decisiones de vertimiento de agua para mantener los niveles de seguridad de la represa tienen un costo asociado a la potencial energía que se hubiera generado con el agua vertida. Adicionalmente, pueden aparecer otros costos del vertimiento asociados a posibles inundaciones o desastres causados por la circulación de agua por el antiguo lecho del río. En particular, en alguna ocasión, el vertimiento decidido causó el derrumbamiento de un puente que comunica la carretera principal con el campamento de EPSA; también podrían verse afectados una subestación y la casa de máquinas. Es natural, entonces, que se busque minimizar la cantidad de agua vertida.

La toma de decisiones en este contexto, consiste en definir el nivel de apertura de las compuertas y el tiempo de duración de dicha apertura, con el fin de garantizar la seguridad de la represa antes que todo, y presevar, en lo posible, el puente, la casa de máquinas y la subestación.

Con el fin de apoyar la toma de decisiones para este caso, se desarrolló VISiR[1], software que ayuda a la toma de decisiones de vertimiento en la represa del Alto Anchicayá.

En este artículo se presentan aspectos relacionados con el modelamiento del problema y del software de ayuda. Los detalles computacionales relacionados con la implementación misma del software y las estrategias de optimización utilizadas no se describen aquí. Inicialmente se hace una presentación global del Paradigma de Programación Concurrente por Restricciones en la sección 2 y una descripción detallada del problema en la sección 3. Posteriormente se presentan el modelo de restricciones del problema (ver sección 4), la arquitectura del software diseñado (ver sección 5), y la interfaz definitiva de la aplicación (ver sección 6). Por último se presentan las conclusiones de este trabajo (ver sección 7).

2. EL PARADIGMA DE PROGRAMACIÓN CONCURRENTE POR RESTRICCIONES.

2.1. El modelo de programación por restricciones

Una gran variedad de problemas combinatorios puede expresarse como la búsqueda de uno o

[1] Vertimiento Inteligente en Situaciones de Riesgo

varios elementos en un vasto espacio de posibilidades. En general el espacio de búsqueda se define como la combinación de todos los valores posibles de un conjunto predeterminado de variables. Los elementos buscados se identifican mediante valores particulares de esas variables. La mayoría de las veces los valores deseados para un elemento no se especifican (o no se conocen) con precisión sino a través de propiedades que deben cumplir los unos con relación a los otros. A estas propiedades se les llama *restricciones*. Las restricciones se expresan usualmente como predicados sobre las variables. Los lenguajes de programación por restricciones consideran esos predicados como sus "instrucciones" básicas.

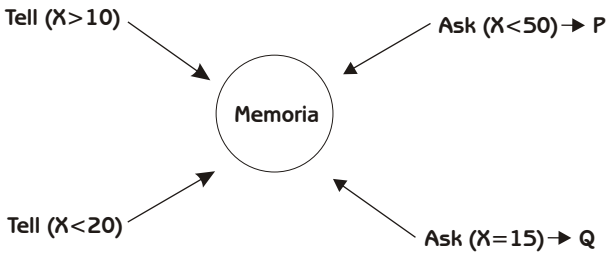

Figura 1: Procesos Ask-Tell

La figura 1 muestra el modelo básico de la programación por restricciones. La memoria contiene predicados que representan información parcial sobre las variables. Agentes interactúan con la memoria agregando información (agentes *tell*) o haciendo preguntas (agentes *ask*). Los agentes *tell* agregan una restricción a la memoria y los agentes *ask* preguntan si de las restricciones ya contenidas en la memoria se puede deducir una restricción. Leyendo los agentes en el sentido de las manecillas del reloj, se agrega la información $X>10$, luego se pregunta si $X<50$ es cierto. Como esto no se puede afirmar, el agente *ask* se bloquea. Similarmente, el agente *ask*$(X=15)\rightarrow Q$ se bloquea. Cuando el agente *tell*$(X<20)$ agregue su información, el agente *ask*$(X<15)\rightarrow P$ podrá "despertar" y continuar con el proceso P. El resultado de un cálculo es en este modelo el conjunto de valores de las variables compatibles con la totalidad de las restricciones que se han agregado a la memoria.

A diferencia de los lenguajes tradicionales, la memoria denota aquí información *parcial*. Cada vez que se impone una restricción mediante la operación *tell*, hay un proceso de *propagación* de sus efectos que consiste, en términos generales, en identificar restricciones adicionales que son su consecuencia lógica. Es perfectamente posible que el conjunto de restricciones así obtenido no determine de manera única los valores de las variables. En este caso se hace necesario un proceso de *exploración*, básicamente una estrategia de ensayo y error inteligente, que busca valores precisos compatibles con la información parcial en la memoria.

2.2. Lenguajes de programación por restricciones

En la década de los noventa surgieron propuestas de varios lenguajes de programación por restricciones. Fundamentalmente se identifican dos corrientes: Una, llamada "programación lógica por restricciones" (CLP), que parte de los lenguajes de programación lógica ya existentes, particularmente de Prolog, y otra, llamada "programación concurrente por restricciones" (CCP) que se fundamenta en el modelo descrito en la sección anterior. En la primera se busca aprovechar todo el desarrollo teórico existente en programación lógica, reemplazando de manera incremental la noción de *unificación* de términos por la de consistencia de restricciones (es decir, por la operación *tell*). Lenguajes como CHIP y Sicstus Prolog pertenecen a este linaje. En lenguajes CCP se añade la operación *ask*, lo que permite hacer que la noción de restricción sea la base sobre la que se construyen todos los demás componentes. Lenguajes como Janus [SK90], Cordial [RAD+01] y *Mozart* [Vh99] pertenecen a este paradigma. El formalismo CCP tiene la ventaja de ser "de grano fino" en el sentido de

exigir solamente unas pocas nociones (*ask, tell* y sus propiedades) sobre las que se construye un sistema. Lo que se gana es una gran flexibilidad porque el tipo de expresiones del lenguaje (es decir, el tipo o sistema de restricciones) es un parámetro que el usuario puede adaptar según convenga a sus aplicaciones.

Los lenguajes del modelo CCP proponen sistemas de restricciones que permiten modelar eficazmente una gran variedad de problemas combinatorios. Los llamados sistemas *de dominio finito*, por ejemplo, se adaptan convenientemente a problemas de planeamiento o asignación de recursos.

El estado actual de los trabajos de investigación reconocen fundamentalmente tres tipos de dominios para las variables que participan en las restricciones: Dominios finitos (intervalos finitos de enteros), intervalos reales y conjuntos. En dominios finitos sobre los enteros (*indexicals*) la referencia actual son los trabajos del INRIA en Francia, de la universidad de Upsala en Suecia y del DFKI en Alemania ([CD96, findom1],[Wür97]). En dominios finitos generales, el estado del arte son los estudios de desempeño en la práctica de los algoritmos conocidos de exploración y consistencia ([DM94, minimalFC] [CBU94, consist]). En dominios sobre intervalos reales los trabajos actuales buscan definir algoritmos eficientes de consistencia extendiendo la noción de relación sobre los reales a relación sobre intervalos de reales ([CDR98], [BGGP99, boxconsist]). En dominios sobre conjuntos la referencia actual son los trabajos del instituto alemán de inteligencia artificial (DFKI [MM97]). . El grupo *Avispa* ha implementado un sistema de restricciones de dominio finito para el lenguaje *Cordial*.

En dominios sobre los reales hay muchos problemas sin resolver. El punto principal es el de conocer el comportamiento de estrategias tales como "consistencia de caja" (*box consistency,* que trata de encerrar los valores que satisfacen las restricciones en una "caja" delimitada por intervalos en el espacio real) o "consistencia de envolvente", ante errores de aproximación en los cálculos numéricos. También saber si los mejores algoritmos de consistencia, tales como HC4 y BC4 ([BGGP99)], son óptimos y cuáles son las mejores estrategias de implementación, particularmente en el marco de CCP.

En la aplicación que se plantea en este proyecto los valores sobre los que se define el modelo son números reales.

3. DESCRIPCIÓN DEL PROBLEMA

A la represa del Alto Anchicayá llegan dos afluentes principales, el río Anchicayá y el río Verde. Antes de desembocar en la represa, el río Anchicayá recibe como afluente al río Grande. La represa, cuya capacidad es de 45 Millones de metros cúbicos, tiene dos vertederos: Un túnel que lleva el agua a la central de generación y un rebosadero constituido por el antiguo lecho del río Anchicayá. Estos dos vertederos desembocan en el río Digua. Entre los puntos de desembocadura de los dos vertederos, sobre el río Digua, está el puente de acceso a los campamentos de la represa (Yatacué). El caudal vertido en cada uno de los vertederos está controlado por compuertas en la represa. El caudal liberado por el túnel no puede sobrepasar las capacidades de generación de la central. El caudal liberado por el antiguo lecho del río debe estar controlado para evitar que su contribución al caudal del Digua ocasione perjuicios al puente de acceso a los campamentos. Por las características particulares de la represa, es vital que el agua no rebase por encima de ella. Existen medidores de caudal (limnígrafos) en el río Digua (La Torre) y en el afluente Anchicayá (El Higuerón). Se planea la instalación de uno adicional en el río Verde. Existe también un medidor de lluvia (pluviómetro) en la represa. Los datos de los limnígrafos se leen por satélite. El caudal medido tarda entre 1.5 y 2

horas en llegar a la represa (para El Higuerón). Actualmente los datos de caudal llegan a EPSA cada 4 horas. Existen actualmente modelos que relacionan el volumen de agua en la represa con su nivel. El problema de toma de decisiones en este contexto consiste en determinar alternativas de vertimiento o acumulación de agua en la represa en función de los datos de los medidores y de las posibles consecuencias del caudal vertido (o volumen acumulado en la represa).

4. MODELO DE RESTRICCIONES

El modelo que representa el sistema de flujo de agua en el Alto Anchicayá puede representarse mediante la figura 2. En esta figura, los círculos representan puntos críticos de información con base en los cuales se toman las decisiones adecuadas y se hace la simulación del sistema. Las líneas gruesas representan los ríos principales que intervienen en el sistema. Las demás figuras representan la represa del Alto Anchicayá, el módulo de generación de energía y el puente cercano a la estación de Monos respectivamente.

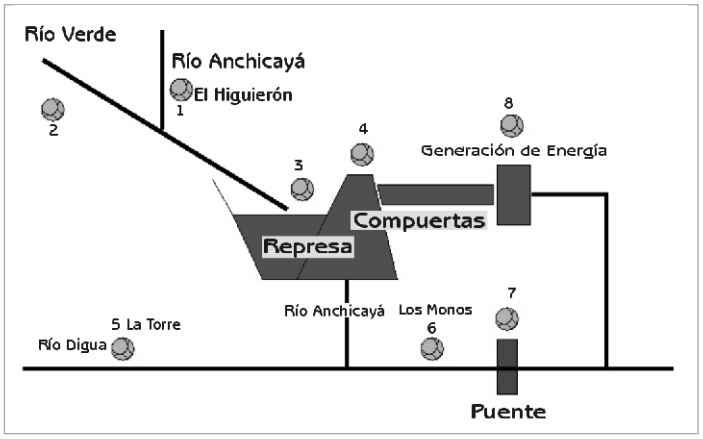

Figura 2: Sistema de Flujo de Agua

Para el funcionamiento del software de ayuda se hace necesario que el sistema cuente con la información de:

- El caudal de agua (dado en 3m_3/s) en los puntos 1, 2 y 5.
- La relación del caudal (m_3/s) en los puntos 1 y 2 por unidad de nivel (m) en el punto 3.
- El caudal vertido en función del tiempo y de la apertura de cada compuerta dada en 4.
- El máximo nivel (m) de agua permitido, dado en 4.
- El mínimo nivel (m) bajo el cual la apertura de las compuertas en 4 se constituye en un vertimiento de agua.
- La relación entre el caudal medido en 5 y el caudal medido en 6.
- El máximo nivel de agua permitido en 7, o el caudal crítico en 7.
- La relación entre el caudal (m_3/s) medido en 6 y el nivel (m) medido en 7.
- El volumen de agua vertido (m_3) en función del tiempo (s) dado por la generación de energía en 8.
- Una tabla de generación de energía por día en función del costo generado.
- El volumen de agua necesitado para generar 1 MV en 8.
- Tiempos de retardo entre 1 y 3, 4 y 6, 6 y 7, 5 y 6 y 2 y 3.

El modelo de restricciones que intervienen en el sistema vienen dadas por la figura 3.

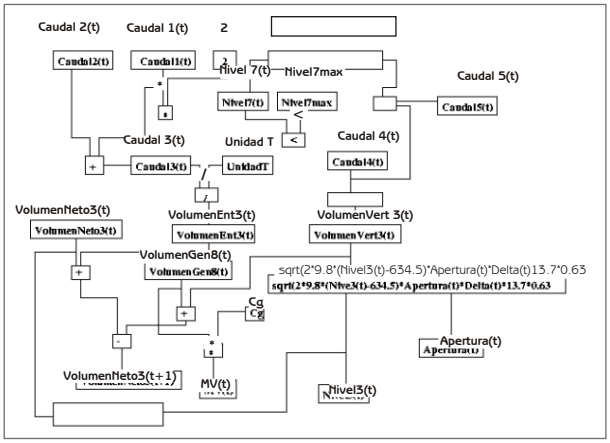

Figura 3: Diagrama de Restricciones

5. ARQUITECTURA DE VISiR

El sistema de toma de decisiones puede representarse mediante la figura 4.

Las entradas al sistema comprenden datos de caudal provenientes de lignígrafos y pluviómetros, así como información sobre caudal

crítico para estructuras que deben protegerse. Adicionalmente, el modelo del sistema físico de la represa determina un conjunto de variables y un conjunto de relaciones entre ellas que determinan restricciones sobre las alternativas posibles de acción sobre los controles disponibles en la represa. El sistema identifica acciones plausibles, categorizadas por nivel de riesgo y las entrega para su análisis por parte de los usuarios. Una vez hecho el análisis el usuario podrá interactuar con el software a través del módulo de simulación, o Refinar el modelo, o tomar decisiones.

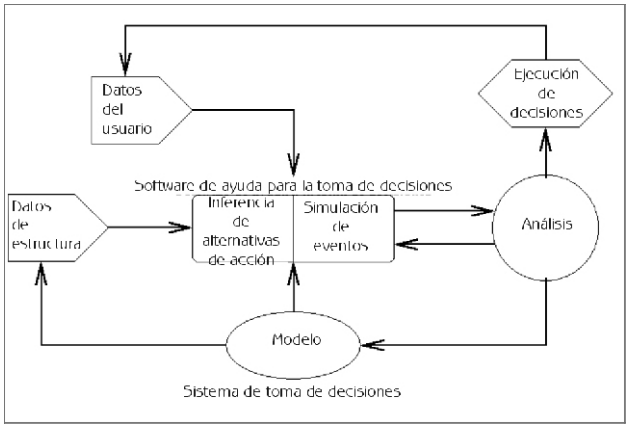

Figura 4: Diagrama general del sistema

5.1. Inferencia en VISiR

El software implanta un sistema de inferencia con base en un conjunto de restricciones (ver figura 5). Las restricciones son las ecuaciones e inecuaciones mostradas en la figura 3 que establecen las propiedades que deben cumplir las mediciones (o condiciones) y las variables de control que, a su vez, determinan las acciones posibles.

El sistema realiza inferencias sobre información parcial, es decir, aún en el caso de ausencia de datos de algunos medidores.

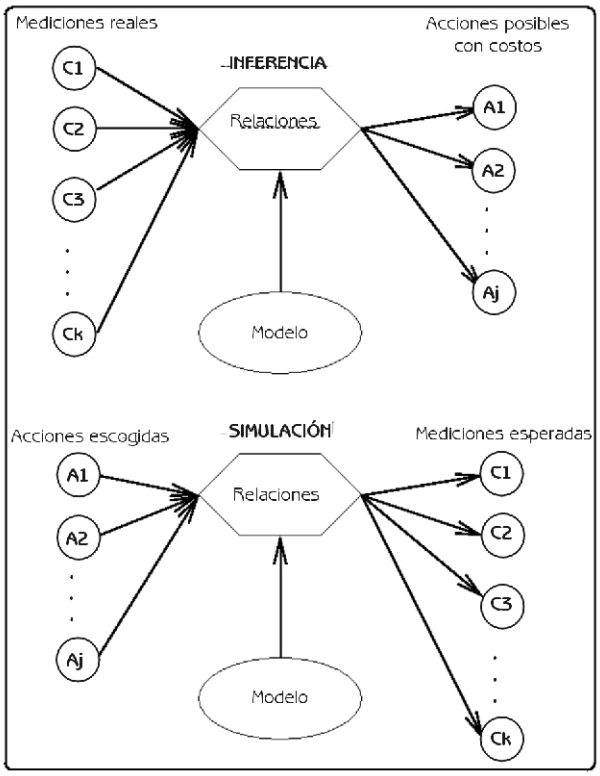

Figura 5: Arquitectura de *software*

5.2. Simulación en VISiR

Adicionalmente, el software de ayuda tiene la capacidad de simulación del comportamiento de los diferentes caudales ante los diferentes eventos causados por las posibles decisiones. Los resultados de la simulación se visualizan de manera gráfica (2D) o textual.

6. LA INTERFAZ DE VISiR: EJEMPLOS DE USO Y FUNCIONALIDAD

Uno de los retos en este proyecto consistió en diseñar una interfaz gráfica que fuera cercana al problema y sencilla de entender para los usuarios. Para ello el grupo se basó en las figuras utilizadas por los ingenieros de EPSA para describir el problema. La Interfaz principal resultante se presenta en la figura 6.

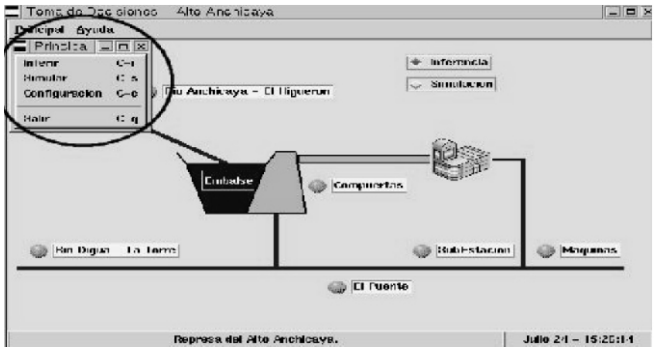

Figura 6: Interfaz del *software*

Cada objeto en la interfaz representa de manera natural un objeto del modelo. Y, el menú principal ofrece las dos posibilidades de uso de la aplicación: para hacer inferencia o para hacer simulación.

En el primer caso, el sistema se alimenta de los datos reales mencionados en la sección 4, realiza la inferencia y produce el programa de apertura de las compuertas para las siguientes dos horas (ver figura 7). La información se presenta en una ventana en la cual se visualiza una tabla que contiene información por intervalos de 15 minutos. Si el tiempo de inferencia corresponde a 2 horas se pueden utilizar las flechas designadas como 1 y 2 en la figura 7 para desplazarse y lograr consultar toda la información de la solución.

Si no existe una solución, es decir un programa de apertura de compuertas que garantice la seguridad de la represa, el sistema lo informará. La flecha marcada con el número 3 debe ser usada cuando se quiera buscar una solución diferente a la que el sistema haya encontrado. Si no existe al menos otra, el sistema lo informará.

Adicionalmente, si el usuario ha buscado varias soluciones y desea revisarlas, se pueden utilizar las flechas marcadas con el número 3 y 4 para verlas.

Por último, mientras el sistema calcula una solución (en algunos casos toma hasta 4 minutos), él informa sobre el estado (procesando información, calculando, proceso terminado) en el espacio señalado con el número 5 en la figura.

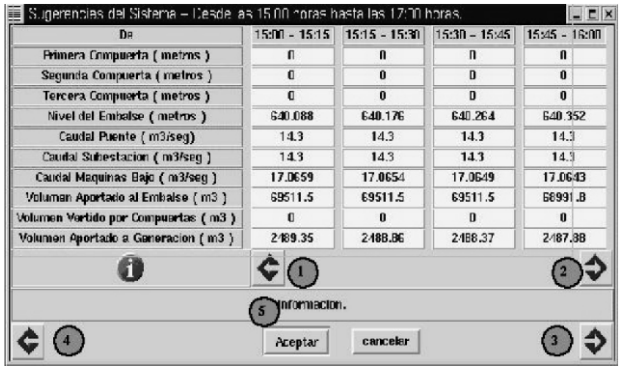

Figura 7: Resultados de inferencia

En cuanto al modelo de simulación VISiR permite:

1. Indagar el comportamiento del sistema dados los caudales en los ríos, el programa de generación de Energía de la Planta, el estado de las compuertas y los niveles máximo, mínimo y actual del embalse.

2. Indagar por el valor adecuado de apertura de las compuertas y los tiempos correspondientes, para lograr vertir un determinado volumen de agua durante el tiempo de simulación. Como datos de entrada para esta simulación se toman los caudales en los ríos, el programa de generación de Energía de la Planta y los niveles máximo, mínimo y actual del embalse.

3. Indagar sobre el programa de generación requerido para mantener el nivel del embalse en un rango determinado, a partir del nivel inicial del embalse, del estado de las compuertas y los caudales del Anchicayá y del Digua durante el tiempo de simulación.

7. CONCLUSIONES

El Paradigma de Programación Concurrente por Restricciones usado para resolver este problema se mostró bastante adecuado tanto para la etapa de diseño como en la etapa de

implementación.

El lenguaje de Programación utilizado (*Mozart*) cumple con todas las expectativas para el diseño de interfaces (elemento esencial para los usuarios de una aplicación) y soporta de manera estricta el Paradigma de programación escogido. Esto nos permitió ahorrar tiempo tanto en el diseño como en la implementación de la aplicación.

Con este proyecto se logró aplicar el conocimiento adquirido por el Grupo de Investigación en la tecnología de programación por restricciones a la solución de un problema real de una empresa como EPSA y de manera innovadora.

La realización de esta aplicación en una plataforma estándar de Programación hubiera tomado más tiempo y muy probablemente el desempeño final de la aplicación no competiría con el desempeño actual de VISiR.

7. REFERENCIAS

[BGGP99] F. Benhamou, F. Goualard, L. Granvilliers, an J-F. Puget. Revising hull and box consistency. *Proceedings ICLP'99*, pages 230-244, 1999.

[CBU94] Freuder E.C. C. Bessière an Regin J.C. U. Using inference to reduce arc consistency computation. In *Proceedings of EACAI'94*, Amsterdarm, The Netherlands, 1994.

[CD96] P. Codognet and D. Díaz. Compiling constraints in clp(fd). *J. Logic Programming*, 27:1:1-199, 1996.

[CDR98] Hélène Collavizza, François Delobel, and Michel Rueher. A note on partial consistencies over continuous domains solving techniques. In *Proceedings of the foruth International Conference on Principles and Practice of Constraint Programming (CP '98)*, Lecture Notes in Computer Science , No. 1520, Berlin, 1998. Springer-Verlag.

[DM94] M.J. Dent and R.E. Mercer. Minimal forward checking. In *6th International Conference on Tools with Artificial Intellingene*, New Orleans, USA, 1994.

[MM97] Tobías Müller and Marrtin Müller. Finite set constraints in Oz. In François Bry, Burckhard Freitag, and Dietmar Seipel, editors, 13. *Workshop Logische Programmierung*, pages 104-115, Technische Universität München, 17-19 September 1997.

[RAD+01] C.Rueda, G. Alvarez, J. F. Díaz, L. Quesada, F. Valencia, and G. Assayag. Integrating constraints and concurrent objects in musical applications: A calculus and its visual langage. *Constraints*, 6:1:21-52, 2001

[SK90] V. Saraswat and K. Kahn. Janus: A step towards distributed constraint programming. In *Proceedings of the North American Conference on Logic programming*, October 1990.

[Vh99] Peter Van Roy and Seif Haridi. Mozart: A programming system for agent applications. In *International Workshop on Distributed an Internet Programming with Logic and Constraint Languages*, November 1999. Part of Internatinal Conference on Logic Programming (ICLP 99)

[Wür97] Jörg Würtz. Constraint-baed scheduling in Oz. In U. Zimmermann, U. Derigs, W. Gaul, R. Möhrig, and K.-P. Schuster, editors, *Operations research Proceedings 1996*, pages 218-223. Springer-Verlag, Berlin, Heidelberg, New York, 1997. Selected Papers of the Symposium on Operations Research (SOR 96), Braunschweig, Germany, September 3-6, 1996.