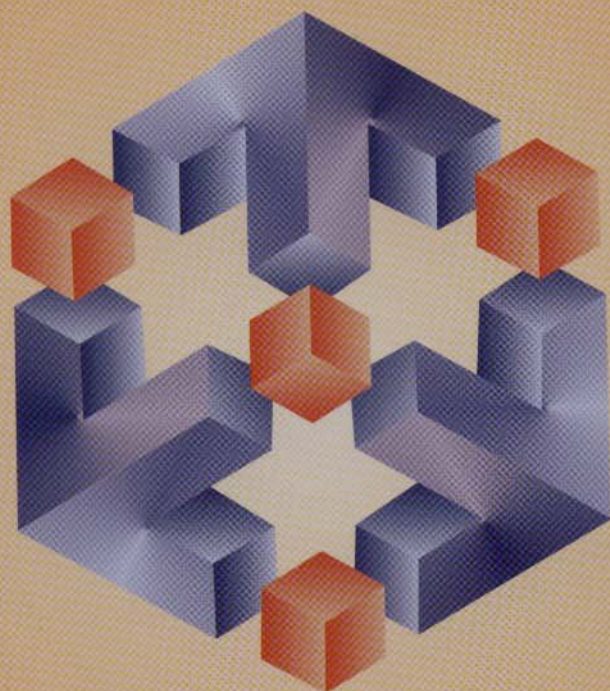


I JORNADAS DE INFORMÁTICA

Puerto de la Cruz - Tenerife
17 al 21 de julio de 1995

ACTAS

Eds. J. M. Troya Linero - C. Rodríguez León



Asociación Española de Informática y Automática
Universidad de La Laguna
Universidad de Málaga

I Jornadas de Informática

Puerto de la Cruz, 17 - 21 de Julio de 1995

Actas

Depósito Legal - MA-777/95

Jose María Torres López
Cristina Rodríguez Lallo

Indice

Conferencias

- Evolución de las arquitecturas de computadores para altas prestaciones 15
Profesor Mateo Valero
CEPBA (Centro Europeo de Paralelismo de Barcelona)
- A High Level Approach to Real-Time and High-Performance Systems 16
P. H. Welch
Professor of Parallel Processing, University of Kent
- Métodos algebraicos generales para la definición semántica y la verificación de sistemas
modulares 17
Profesor Fernando Orejas
Dpto. de L.S.I. Universitat Politècnica de Catalunya

Sesiones

Paralelismo I

- Empotrado parcial de cubos y planos sobre toros para solapar cálculos y comunicaciones ... 19
A. Suárez, C. N. Ojeda-Guerra
- Encaminador Asíncrono Wormhole para Redes de Interconexión de Multicomputadores 31
C. Carrión, F. Vallejo, R. Menéndez, J.A. Gregorio, R. Beivide
- Implementación de un entorno paralelo orientado al modelo PRAM 41
C. León, F. Sande, F. García, C. Rodríguez
- EDISIS: Un entorno de diseño y Simulación de algoritmos sistólicos y semisistólicos 53
F. Fernández, J. Gutiérrez, A. Sánchez

Sistemas de Información y Bases de Datos

- Mecanismos de autorización para los disparadores en los sistemas de bases de datos 63
O. Díaz, A. Irastorza, J. Iturrioz
- Transformación del Modelo Entidad-Relación Extendido (EER) al Modelo Deductivo 75
M. A. Dapena, N. Rodríguez
- Notación para el Comportamiento Reactivo de los Sistemas de Información 85
O. Díaz, J. Iturrioz, M. Piattini

Involucrando al usuario en el desarrollo de sistemas de información	101
<i>A. Aguayo, J. Falgueras, A. Guevara, F. Triguero</i>	

CORALFACE/SM: Un interface gráfico para CORAL en arquitectura cliente-servidor usando Smalltalk	111
<i>S. Bamonde, R. Catoira, J.M. García-Tizón, V.M. Gullás</i>	

Concurrencia

Construcción de una máquina química abstracta	121
<i>C. Herrero, J. Oliver</i>	

Asociatividad en Algebras de Procesos Probabilísticas	135
<i>M. Núñez, P. Palao, M. T. Morazán</i>	

Comunicación Monádica de Objetos Funcionales	145
<i>J. E. Gallardo, P. Guerrero, B.C. Ruiz</i>	

Corrección de programas y redes de Petri de alto nivel	155
<i>F. Tricas, J. Martínez</i>	

Tiempo Lógico en Lenguajes de Tiempo Real	165
<i>P.T. Breuer, N. Martínez, A. Marín, L. Sánchez, C. Delgado</i>	

Sistemas Informáticos Aplicados I

Sistema de visión artificial basado en transputers con invariancia del color	175
<i>A. Ruiz, I. Llorens, V. Arnau, M. Vicens, J.M. Artigas</i>	

Descripciones declarativas de distinto nivel: aplicación al diseño de invernaderos	185
<i>J.R. García, J.F. Bienvenido, J. Sagrado, L.F. Iribarne, A. Becerra, R. Guirado, J.R. Díaz, F. Rodríguez, R.M. Ayala</i>	

Sistema en tiempo real para la representación topográfica de las señales del EEG/EP	195
<i>J. J. Merino, J. D. Piñeiro, S. Mañas, J. F. Sigut, L. Moreno</i>	

Una propuesta de invariantes basados en momentos: la transformada de Fourier-Laguerre	205
<i>F. Pérez, A. Falcón</i>	

Técnicas de Inteligencia Artificial

Arquitectura para el aprendizaje de clasificadores en un sistema de visión basado en conocimiento. Consideraciones de diseño	215
<i>J. Lorenzo, M. Hernández, M. Castrillón</i>	

Una utilización eficiente de búsquedas locales integradas con técnicas globales en un sistema de producción en planta	225
<i>J. Arrieta, J.R. Zubizarreta</i>	
Problemas del control óptimo: una aproximación mediante redes neuronales	235
<i>A. Hamilton, L. Acosta, N. Marichal, J.A. Méndez, L. Moreno</i>	
Compresión de imágenes mediante redes neuronales con aprendizaje competitivo hebbiano	245
<i>M.A. García, F. López</i>	
Sistema basado en el conocimiento para el estudio de señales cerebrales	251
<i>J.D. Piñeiro, J.L. Sánchez, S. Mañas, R.M. Aguilar, J.J. Estévez, L. Moreno</i>	
<u>Técnicas Algorítmicas</u>	
Reflexiones sobre el diseño de Sistemas de Visión Basados en Conocimiento	257
<i>J. Cabrera, D. Hernández, A. Falcón, J. Méndez</i>	
Una regla de parada para la Búsqueda con Arranque Múltiple	271
<i>J. M. Moreno-Vega, J. A. Moreno</i>	
Una Implementación del Algoritmo de Compresión Lempel-Ziv Welch	281
<i>V. G. Ruiz, I. García</i>	
Un algoritmo para medir la seguridad práctica de algunos cifrados en flujo	293
<i>P. Caballero, A. Fúster</i>	
Métodos de preselección de píxeles para filtros tipo mediana	301
<i>L. García, P. L. Luque, R. Román</i>	
<u>Lenguajes y Programación</u>	
Síntesis constructiva de Programas Lógicos	311
<i>F.J. Galán, M. Toro</i>	
Cremalleras y espirales o cómo co-diseñar mejor	323
<i>N. Martínez, A. Marín, S. Deprés, C. Delgado, L. Sánchez</i>	
CASALE I 1.0.B: un lenguaje de simulación discreta basado en objetos	333
<i>R. Corchuelo, P. Carrillo</i>	
<u>Sistemas Informáticos Aplicados II</u>	
Sistema de Interfaz Persona-Computador para Entornos Virtuales	343
<i>R. Rodríguez, M. Fernández, M. Pérez, S. Bayarri, F. Martínez</i>	

Control informático global de un sistema ferroviario a escala	353
<i>J. M. Vicente, G. Bandera</i>	
Análisis y siseño de tablas de pseudocolor para visualización de imágenes médicas en PC ..	363
<i>V. Arnau, M. Vicens, J.M. Orduña, A. Felipe, A. Palomares</i>	
Programación de un sistema de eculización digital	373
<i>J.V. Francés, J. Guerrero, J. Calpe, A. Rosado</i>	

Paralelismo II

Programación de Técnicas Algorítmicas en dos Sistemas Distribuidos: PVM y Transputers	381
<i>F. Almeida, F. García, J. Roda, D. Morales, C. Rodríguez</i>	
Implementación de un algoritmo thinning paralelo: El desbalanceo de la carga	393
<i>M.D. Gil, J. Roca, I. García</i>	
Paralelización de un algoritmo de validación de protocolos	405
<i>F. Rus, P. Merino, M. Díaz</i>	
Programación dinámica paralela para problemas de la mochila 0/1	415
<i>F. Almeida, D. Morales, F. García, C. Rodríguez</i>	

Docencia I

Sobre la enseñanza de las bases de datos: teoría y diseño	427
<i>M. Celma, L. Mota, M. A. Pastor, J.C. Casamayor</i>	
Implementación de Tipos Abstractos de Datos sobre Lenguajes Imperativos	437
<i>J. M. Molina</i>	
Diseño digital basado en dispositivos lógicos programables: una experiencia docente con ABEL-PLD	447
<i>A. Rosado, O. Farpón, M. Bataller, J. Guerrero, J. V. Francés</i>	
Visualización de la ejecución de programas en una CPU segmentada.....	455
<i>A. Suárez, M. Marrero</i>	

Docencia II

Docencia sobre las implicaciones sociales y éticas de la informática	465
<i>M. Barceló</i>	
Análisis y valoración de la formación informática en los estudios universitarios	475
<i>L. García, L.A. Ureña</i>	

Una Experiencia Docente en Interacción Hombre-Ordenador 485
J. Lorés, C. Abarca

Motivaciones para el trabajo relacionado con empresa del profesorado (las experiencias reales) 495
R. Cores, J. Ayude, M. Pérez

Proyectos I

Entorno general de producción automática de prototipos de software orientado a objetos: OOAP (Object-Oriented Assistant Prototyper) 501
Grupo OASIS

CCASE - Una herramienta MetaCASE Europea Avanzada Parametrizable 515
J.R. Freixanet, J.M. Espejo, J. Canal

Bases de Datos: Gráficas + Alfanuméricas integradas en la gestión de recursos materiales de la U.P.V. 531
A. Martínez

La versión 2 del programa SSD_CIABSI. Un ejemplo real de utilización de medios técnicos en la decisión administrativa 545
G. Nistal

Proyectos II

Evaluación de un sistema de ficheros paralelo para máquinas masivamente paralelas 555
J. Carretero, F. Pérez, P. de Miguel, F. García, L. Alonso

PACOTE: Paralelización de un código termohidráulico para la simulación de plantas nucleares 567
M. Díaz, L. Llopis, J.L. Pastrana, F. Rus, E. Soler, J.M. Troya

Proyecto TIMS: Tarjeta Inteligente Multiservicio 577
V. Cerverón, V. Cavero, J.V. Pelechano, G. Martín

DAMOCIA: Diseño asistido mediante ordenador para la construcción de invernaderos automatizados 587
J. F. Bienvenido

Proyectos III

EUROWARE, una solución para fomentar la reusabilidad en redes de área extendida 597
G. Sánchez

EVP : Un entorno para la integración de técnicas de descripción formal 607
Grupo de Ingeniería del Software. Universidad de Málaga

I + D en tecnología software. Una experiencia de colaboración Universidad-Empresa 617
J.J. Gil, J.C. Yelmo

Un modelo para comunicaciones multimedia isócronas 627
S.P. Labajo, A. Pacheco, E. Rendón, C. Muñoz

Proyectos IV

MIX: Un Enfoque Multiagente para Sistemas Híbridos Simbólico-Conexionistas 637
J.C. González, J.R. Velasco, C.A. Iglesias, J. Alvarez, A. Escobero

Proyecto INVAID II : Integración y evaluación de un sistema de D.A.I. basado en visión artificial en áreas urbanas 647
J.J. Martínez, R. Ferrís, V. Cavero, J. Martínez, A. Fuertes, G. Martín

Proyecto ARTIS WA 3 : Integración y evaluación de un sistema de D.A.I. basado en visión artificial en la M-40 (Madrid) 657
J.J. Martínez, R. Ferrís, J. Martínez, J. Pelechano, G. Martín

Descubrimiento de conocimiento para diagnosis de fallos en producción de circuitos integrados 667
A. Ribeiro, M.D. del Castillo, L.J. Barrios

Pósters

Un planteamiento de las sociedades de objetos como técnica de validación y prototipación de software 675
J. Rodeiro, M. Pérez, F.J. Vázquez

Complejidad y gestión del riesgo en el desarrollo de software orientado a objetos 677
E. Barreiro, P. Vázquez

Planteamientos de seguridad para redes de comunicaciones 679
J. Areitio, M. T. Areitio

Resolución de sistemas dispersos de ecuaciones lineales sobre PVM explotando paralelismo en el método de Gauss-Seidel 681
B. Sahelices, A. de Dios, V. Cardeñoso

Algoritmo para la aproximación de ecuaciones diferenciales con retardos 683
C.F. Alastruey, J.R. González de Mendívil

Computación Universal en Autómatas Celulares 685
A. J. Tomeu

Monitores CRT colorimétricamente controlados 687
A. Palomares, A. Lorente, V. Arnau, J.M. Artigas

Enseñanza Asistida por Ordenador del Lenguaje Ensamblador MC68000	689
<i>F. Quintana, L. Herrera</i>	
La enseñanza de la Ingeniería del Software según el nuevo plan de estudios en la Universidad Jaime I	691
<i>C. Campos, O. Coltell</i>	
Técnicas de Multimedia para el autoaprendizaje de conceptos básicos de informática	693
<i>O. Coltell, P. J. Sanz, P. Valero, F. Ester</i>	
Multimedia	695
<i>A. Rodríguez, J.F. Gálvez, M. Pérez</i>	
El procesado digital de la señal en la titulación de la ingeniería electrónica de la Universitat de Valencia	697
<i>J. Calpe, J.F. Guerrero, E. Soria, C. Hernández, J. Espf</i>	
La enseñanza de la electrónica digital en las ingenierías	699
<i>V. Arnau, M. Vicens, J.M. Orduña</i>	
Descripción de la red de interconexión SCI (Scalable Coherent Interface) IEEE P1596	701
<i>V. Arnau, V. González, E. Sanchls, J. Ferrer, J.M. López, F. Mora, A. Sebastiá</i>	
MEDIDA: Hacia un modelo de medición	703
<i>A. Bernaras, I. Ortiz</i>	
Modelado del Análisis de la Replicación del ADN por Electroforesis Bidimensional en Geles de Agarosa	705
<i>A. Rodríguez, E. Viguera, P. Hernández, J.B. Schwartzman, O. Trelles</i>	
Verificación de Conexiones Telefónicas con ESTEREL	707
<i>J. Graña, M. Vilares, R. Bernhard</i>	
CASE DFDC. Una herramienta cooperativa de diagramas de flujo de datos	709
<i>A. Guevara, A. Aguayo, J. Falgueras, F. Triguero</i>	
Interface QBE para consultas a Bases de Datos Deductivas	711
<i>M.A. Pardavila, N. Rodríguez</i>	
Desarrollo de una aplicación para la Diputación Provincial de La Coruña	713
<i>J.A. Martínez, A. Rico, J.M. García-Tizón, N. Rodríguez, M. Rodríguez-Losada</i>	
Implementación de un sistema gestor de bases de datos orientadas a objetos	715
<i>J. L. Berrocal, G. Rodríguez, J. F. Aldana</i>	

EVP: UN ENTORNO PARA LA INTEGRACIÓN DE TÉCNICAS DE DESCRIPCIÓN FORMAL¹

*Grupo de Ingeniería del Software²
Dpto. Lenguajes y C. Computación
Fac. Informática. E.T.S.I. Telecomunicación
Universidad de Málaga*

Resumen

EVP es un entorno para la especificación y análisis de sistemas distribuidos y protocolos de comunicación, cuyo objetivo principal es ofrecer la posibilidad de utilizar diferentes técnicas de descripción formal (TDFs) de una forma integrada. El entorno se ha desarrollado utilizando una implementación distribuida del lenguaje lógico concurrente DRL (Distributed Real-Time Logic), diseñado especialmente para su construcción. Dicho lenguaje es utilizado al mismo tiempo como TDF y como lenguaje para la implementación de las herramientas que componen el entorno. La integración de otras TDFs se consigue mediante la traducción a DRL. Actualmente se dispone de traductores para las TDFs LOTOS y SDL.

Las tareas de análisis se realizan directamente sobre DRL, aunque el usuario percibe los resultados en relación a los lenguajes de especificación originales. El uso de un núcleo de ejecución declarativo facilita la construcción de herramientas para simulación y validación, al poderse utilizar técnicas avanzadas de programación, como la metainterpretación. La implementación distribuida del mismo permite, además, ampliar las posibilidades de la validación, tanto por su tamaño como por su calidad.

1. INTRODUCCION

La utilización de Técnicas de Descripción Formales (TDFs) en el desarrollo de protocolos se ha visto cada vez más necesaria, a medida que aumentaba la complejidad de los sistemas distribuidos. En los últimos años, la aparición de estándares (LOTOS [15], SDL [3], ESTELLE [16]) y de herramientas que los soportan, ha permitido aplicar dichas técnicas a la especificación formal de protocolos, sentando las bases para el desarrollo de un software de comunicación más robusto y fiable.

La gran complejidad de las especificaciones formales desarrolladas con estas técnicas obliga a asegurar su corrección y validez respecto a los requerimientos iniciales del sistema. Dado que la verificación formal de las especificaciones es muy difícil de realizar, por las necesidades computacionales que presenta, se hace necesario realizar validaciones lo más completas posibles.

El objetivo de EVP es la integración de diferentes formalismos, y sus técnicas asociadas, para la especificación y análisis de sistemas distribuidos, de forma que los diseñadores puedan combinarlos según preferencias y adecuación a cada problema [13]. El nuevo entorno está basado en la utilización del lenguaje declarativo DRL como núcleo del sistema, de forma que las herramientas de análisis están enfocadas hacia este lenguaje único. Este lenguaje se ha diseñado e implementado con este fin [7]. Para facilitar la integración, se construyen traductores e interfaces de usuario que presenten los resultados en la forma esperada. Dicha integración se

¹ Este trabajo ha sido subvencionado por el proyecto coordinado TEMA y el CICYT TIC-1301-PB del PLANBA

² Componentes del Grupo de Ingeniería del Software: E. Alba, J. Aldana, V. Benjumea, C. Canal, M. Diaz, L. Fuentes, M.M. Gallardo, P. Merino, A. Nebro, E. Pimentel, M. Roldan, B. Rubio y J.M. Troya

lleva a cabo utilizando DRL como lenguaje de cooperación, mediante el cual se definen los interfaces a través de los cuales interaccionan los modelos descritos con diferentes formalismos.

Un enfoque similar se ha seguido en el proyecto SPECS desarrollado simultáneamente en el marco del programa europeo RACE [21]. En este proyecto se consideran dos lenguajes internos diferentes: I-CRL y A-CRL. A-CRL se utiliza para análisis, mientras que I-CRL se utiliza para implementación. Sin embargo, los lenguajes intermedios no responden al mismo modelo computacional, ya que el primero es más cercano a LOTOS, mientras que el segundo está basado en el modelo de máquina de estados. Este entorno incluye traductores de LOTOS y SDL.

En relación a otros entornos que también utilizan diferentes TDFs la principal aportación de la nueva herramienta es la integración de TDFs con modelo de transiciones y álgebra de procesos. AUTO [9] sólo permite la integración de formalismos basados en álgebra de procesos como CCS o BASIC LOTOS. En SEDOS [17] se desarrollaron herramientas para ESTELLE y LOTOS, pero no permiten la construcción de especificaciones mixtas con ambos lenguajes.

El uso de un lenguaje declarativo para el análisis de sistemas distribuidos se ha justificado desde hace ya algunos años ([20], [6], [2]). Recientemente se han utilizado con éxito en la construcción de prototipos [5] [18], y en simulación y validación [10]. Además se ha demostrado su utilidad como lenguaje objeto para la traducción de las TDFs estándar [12], [13]. Respecto de otras herramientas, EVP presenta las siguientes ventajas:

- i) Es un entorno distribuido que puede ejecutarse en una red de estaciones de trabajo, lo que permite aumentar fácilmente los recursos computacionales dedicados a la validación de las especificaciones.
- ii) Debido a la expresividad y la gran cantidad de técnicas de programación concurrente asociadas a los LLCs es fácil modelar distintos mecanismos de interacción entre procesos. Esto permite la traducción de TDFs basadas en distintos paradigmas (máquinas de estados, álgebras de procesos, redes de Petri, ...).
- iii) Por el mismo motivo, el propio DRL puede utilizarse como lenguaje de especificación, en línea con las propuestas de utilizar lenguajes lógicos concurrentes en el desarrollo de sistemas altamente concurrentes. Asimismo, puede resultar útil como lenguaje de interconexión entre las diferentes TDFs.

El resto del artículo se estructura de la siguiente forma. En la sección 2 se discute la metodología de diseño asociada al entorno EVP. La sección 3 resume las características de los componentes actuales del entorno. Para finalizar se exponen algunas conclusiones y las líneas de trabajo futuro.

2. METODOLOGÍA ASOCIADA A EVP

La Figura 1 resume la metodología de desarrollo de software para sistemas distribuidos contemplada en EVP. Aunque actualmente no se han implementado todos sus componentes, describiremos el proceso de diseño completo. Como primer paso, se pueden crear una serie de especificaciones formales que reflejan el comportamiento del sistema, sus requerimientos y los datos que se utilizan. Para la especificación del comportamiento se pueden describir partes combinando las TDFs estándar (LOTOS, ESTELLE, SDL) y el propio lenguaje DRL. Los requisitos de corrección se establecen con uno o varios lenguajes que modelan el comportamiento deseado del sistema, fundamentalmente en lo que se refiere a su evolución temporal. La definición de los diferentes tipos de datos se realiza con un formalismo algebraico del tipo ACT-ONE [15], mientras que la estructura abstracta de los mensajes del protocolo podría hacerse con ASN.1. Ambos formalismos pueden combinarse de forma similar a como se recomienda para combinar SDL y ASN.1.

Las especificaciones anteriores se traducen al lenguaje base del entorno, generando una especificación DRL que puede utilizarse como prototipo y como entrada para las herramientas de análisis.

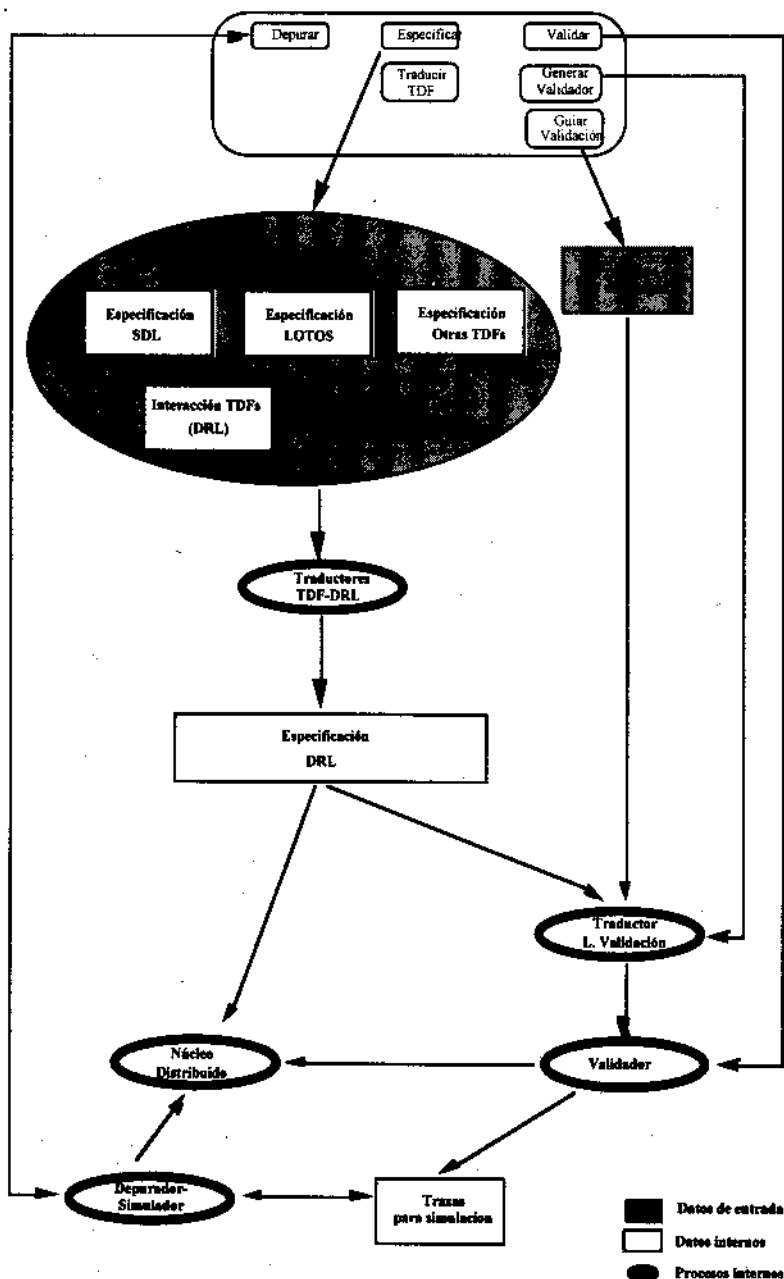


Figura.1. Esquema general del entorno

Actualmente, EVP contempla el uso de SDL, LOTOS y DRL como lenguajes de especificación del comportamiento, incluyendo un conjunto de tipos de datos predefinidos. Con respecto al análisis, en la versión actual sólo se pueden especificar invariantes del sistema. El análisis incluye la simulación interactiva y la validación de propiedades de seguridad por análisis de alcanzabilidad.

3. EVP: FUNCIONALIDAD DE LOS COMPONENTES

En esta sección se describen las características básicas de los componentes de la versión actual del entorno EVP. Para cada uno de ellos se intenta justificar el enfoque utilizado y se presentan las principales aportaciones en el marco lógico concurrente seguido.

3.1. DRL

Los Lenguajes Lógicos Concurrentes (LLCs) han sido propuestos como herramientas en distintas fases del diseño de protocolos como por ejemplo en la especificación, construcción rápida de prototipos y en la simulación y validación de especificaciones [14][6]. Sin embargo, se han encontrado algunos problemas en la utilización de estos lenguajes, como por ejemplo la ausencia de mecanismos de control de tiempo real, implementaciones distribuidas poco eficientes o la imposibilidad de expresar algunas propiedades de viveza. Algunas de estas carencias han sido puestas de manifiesto en [11][8]. Los lenguajes lógicos concurrentes son lenguajes de muy alto nivel para sistemas paralelos para los que existen una gran cantidad de técnicas de programación concurrente [19]. Los tres más importantes lenguajes de esta familia son Parlog [4], GHC [22] y Concurrent Prolog [19]. Al ser lenguajes lógicos, mantienen numerosas ventajas del modelo de programación lógica, incluyendo la interpretación lógica de la ejecución, representación de datos como términos lógicos, manipulación de datos mediante unificación etc. Su modelo de ejecución consiste en un conjunto de procesos concurrentes que se comunican por instanciación de variables lógicas compartidas y que se sincronizan mediante la espera en variables no instanciadas. Una computación comienza con un conjunto de objetivos; cada objetivo es un átomo $p(T_1, \dots, T_n)$, que representa un proceso. El estado de dicho proceso vendrá dado por la sustitución parcial de cada uno de los términos T_i . El conjunto de objetivos puede ser considerado como una red de procesos que se comunican entre sí. El comportamiento de cada proceso se especifica mediante cláusulas de Horn con guardas, con el siguiente formato:

$$h \leftarrow g_1, g_2, \dots, g_m : b_1, b_2, \dots, b_k.$$

La cabeza de la cláusula (h) y la guarda (la conjunción de predicados g_1, g_2, \dots, g_m) especifican las condiciones para una transición de reducción. El conjunto de predicados b_1, \dots, b_k se denomina cuerpo de la cláusula. La computación de un proceso comienza con la selección de una cláusula candidata de entre las que definen el comportamiento de dicho proceso. Dicha selección se realiza en dos fases: a) unificación de entrada (matching) entre el objetivo y la cabeza de la cláusula sin instanciar ninguna variable del objetivo; y b) evaluación de las guardas. Estos lenguajes son no deterministas en un sentido "don't care", es decir, una vez que se produce una transición no existe vuelta atrás en el caso de que el proceso falle, cosa que sí ocurre en Prolog. Desde un punto de vista formal, esto permite la observación de las sustituciones parciales entre los procesos y la comunicación entre ellos.

Las características del lenguaje del núcleo de ejecución vienen determinadas por dos factores principales. El primero es que dicho lenguaje debe ser lo suficientemente expresivo para permitir su uso directamente como lenguaje de especificación de sistemas distribuidos o para realizar una traducción fácil de otras TDFs al lenguaje. El segundo factor es el tipo de entorno que se pretende construir; se trata de un entorno que se ejecutará sobre un sistema distribuido débilmente acoplado y que debe permitir el control del tiempo real. El primer factor nos llevó a la elección de un lenguaje basado en un LLC, ya que estos lenguajes tienen suficiente potencia como para permitir una traducción fácil a partir de las técnicas de descripción formal (TDFs). El segundo factor nos lleva a que nuestro lenguaje tenga una serie de características que permitan una ejecución eficiente en el tipo de sistema al que va destinado. Entre las características más

importantes cabe destacar: modularidad, ejecución eficiente en entornos distribuidos, control de procesos y control del tiempo real.

El lenguaje del núcleo de ejecución tiene un modelo computacional orientado a procesos. Los procesos se pueden agrupar en *gránulos*, que son la mínima unidad de un programa que se puede ejecutar de forma remota. El comportamiento de un proceso vendrá determinado por un conjunto de cláusulas de Horn con guardas planas, es decir la guarda de una cláusula será siempre una conjunción de predicados predefinidos. El conjunto de cláusulas que define un proceso vendrá precedido de una declaración de modo, indicando la direccionalidad de las variables.

Se pueden expresar restricciones de tiempo en la definición de un proceso mediante el uso de guardas temporales. Una guarda temporal es una guarda en la que aparece el predicado predefinido *after(T)*, donde T expresa el tiempo en milisegundos desde que la guarda se evalúa con éxito hasta que realmente se produce la reducción. Aparte de la utilización de guardas temporales existen distintas primitivas de tiempo real que permiten controlar de una forma más fácil el tiempo de ejecución de los procesos.

Una especificación estará compuesta por un conjunto de módulos en los que se define el comportamiento del sistema mediante procesos y gránulos que se comunican a través de canales lógicos. El estado inicial del sistema vendrá dado por una conjunción de procesos y llamadas remotas a gránulos. Una especificación de este tipo puede ser utilizada tanto como prototipo del sistema, como para entrada de las herramientas de análisis. Para ejecutar un prototipo basta con compilar y cargar el modelo especificado en el entorno de ejecución, dando un objetivo (estado inicial) en el intérprete de comandos del sistema. Este intérprete incluye un depurador que permite visualizar y controlar la ejecución distribuida de los procesos. El intérprete de comandos devolverá al final el resultado de la ejecución de los procesos (éxito, fallo o bloqueo), así como los contenidos de las variables y los canales de comunicación que se han monitorizado durante la ejecución.

3.2. TRADUCTORES DE TDFs

La integración de distintas TDFs en EVP se lleva a cabo mediante la utilización de diferentes traductores a DRL, que, como hemos dicho, es el modelo semántico común al conjunto de las herramientas que conforman el entorno. La misión de los traductores es reproducir la semántica original a la equivalente en DRL. Esta equivalencia se ha establecido a partir de las semánticas operacionales de los distintos formalismos y la semántica operacional de DRL [7]. A continuación se describen los esquemas de traducción de las dos TDFs standard que en la actualidad soporta el entorno.

3.2.1. Traductor de SDL

La conveniencia de utilizar los lenguajes lógicos concurrentes, concretamente DRL, como destino de la traducción de una especificación SDL, viene dada por la posibilidad de identificar los elementos básicos del lenguaje de especificación con construcciones propias del lenguaje lógico concurrente [13]. Es decir, cada constructor de SDL se traduce básicamente en una o varias cláusulas DRL correspondientes a la definición de un predicado lógico.

Siguiendo una descripción descendente, una especificación SDL consiste en un *sistema* dividido en diferentes *bloques* que interactúan entre sí. Cada uno de ellos basa su actividad en el comportamiento de diferentes *procesos* conectados entre sí y con otros bloques. Así, la componente de nivel más bajo en un sistema SDL es el proceso, que se caracteriza por la capacidad de realizar determinadas acciones, que dependen de una señal de entrada y del estado en que se encuentre; es decir, la historia de un proceso se corresponde con una secuencia de estados. De este modo, un proceso puede identificarse con un objeto que recibe señales del exterior a través de una cola fifo, y realiza una serie de acciones dependiendo del tipo de señal recibida; entre estas acciones se incluye la creación de nuevos procesos y la posibilidad de cambiar de estado. Todas estas consideraciones permiten establecer una equivalencia entre

proceso SDL y el concepto de actor [1], introducido como modelo de cómputo concurrente para sistemas distribuidos. Esta visión nos permite simular procesos SDL con lo que se denominan procesos perpetuos en los lenguajes lógicos concurrentes que constituyen una forma de implementar actores.

Lo interesante para nosotros en relación a la equivalencia de un proceso SDL con un proceso perpetuo en DRL radica en que el paso de un esquema a otro se puede automatizar, lo cual permite construir una herramienta de traducción que a partir de una especificación SDL obtenga un programa DRL cuya ejecución tendrá la funcionalidad del protocolo especificado. Por otro lado, el hecho de ser la equivalencia antes mencionada muy explícita y directa hace posible que se puedan establecer conexiones muy precisas entre el código fuente y el código objeto, de forma que los errores producidos en la ejecución de DRL se pueden identificar con segmentos de código en la especificación SDL original. En la traducción de SDL a DRL se considerarán independientemente los tres niveles en que se divide una especificación SDL: sistema, bloque y proceso. A continuación se describen las características de cada uno de ellos, junto con su correspondiente traducción a DRL.

Con una especificación SDL se describe el comportamiento de un sistema, suponiendo que interactúa con su entorno. Puesto que nuestro objetivo es ejecutar dicha especificación, deberemos programar en cada caso el comportamiento del entorno y su interacción con el sistema, con objeto de obtener un programa DRL completamente ejecutable que represente la especificación de nuestro sistema en su entorno particular. Aunque en la especificación de un sistema a ejecutar no se define el comportamiento del entorno, sino únicamente su interfaz con el sistema, habrá que especificar este entorno con un predicado DRL y comunicarlo adecuadamente con el predicado al que se traduce el sistema.

Un sistema SDL se compone de bloques comunicados a través de canales, y cada uno de ellos estará compuesto por procesos conectados mediante rutas de señal. La traducción de un sistema vendrá dada por una cláusula en cuyo cuerpo se invocan los procesos DRL que gestionan la actividad de cada uno de los bloques que lo constituyen (gestor de bloque). Del mismo modo, el comportamiento de cada bloque estará caracterizado por los procesos que lo definen (Fig. 2).

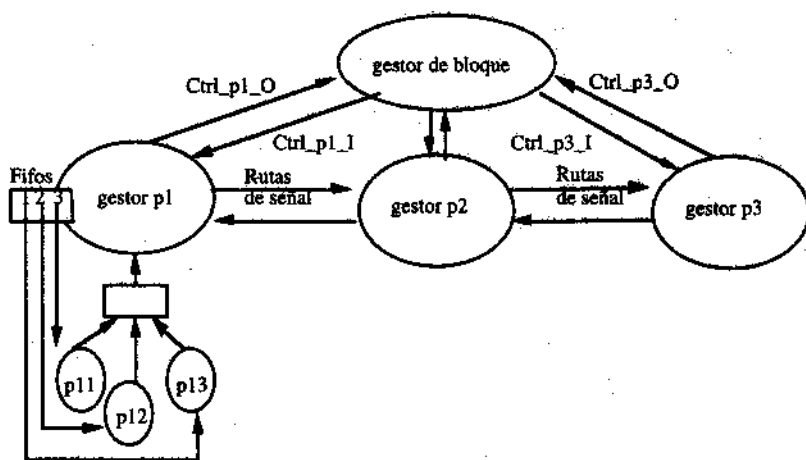


Figura 2. Esquema de la traducción a nivel de bloques y procesos

Como ya se ha mencionado, cada proceso SDL se corresponde directamente con un proceso DRL (p_i), pero para mantener un control de las diferentes instancias (p_{ij}) que se puedan crear durante la ejecución de una especificación, se necesita un proceso auxiliar (gestor p_i). Cada proceso se conecta con el bloque que lo contiene mediante enlaces bidireccionales adecuados ($Ctrl_{p_i-I}$, $Ctrl_{p_i-O}$) a través de dichos gestores. El papel que desempeña el gestor de un

proceso SDL es fundamentalmente el de mantener referencias (Fifos) a sus instancias, y determinar el destinatario de las señales que recibe.

3.2.2. Traductor de LOTOS

Una primera aproximación a la comparación entre LOTOS y DRL nos hace ver que son lenguajes totalmente distintos, tanto en la forma como en sus orígenes. A pesar de esta diferencia cabría una visión simplista en la búsqueda de equivalencia entre los constructores LOTOS y los operadores en los de DRL (Fig. 3). La idea inicial en la búsqueda de equivalencias es definir predicados que emulen el comportamiento de los operadores LOTOS. Antes de buscar esta equivalencia de operadores es necesario buscar la equivalencia entre la interacción de un proceso con su entorno en LOTOS y en DRL. En LOTOS un proceso puede ser visto como una caja negra que ofrece unas acciones a su entorno para sincronizar e intercambiar datos a través de puertas. Intentaremos mantener la equivalencia proceso LOTOS-proceso LLC. Para representar las acciones que se ofrecen a través de una puerta podemos pensar en usar una lista de comunicación (stream) por puerta, pero esto resultaría muy ineficiente. La solución por la que se ha optado es la de utilizar una sola lista de comunicación para todas las puertas. Cada proceso ofrecerá sus acciones, a través de esta lista, a su entorno superior, que puede ser bien el entorno exterior o uno intermedio, que correspondería a un operador.

LOTOS	DRL
; operador prefijo	& And
secuencial	
>> habilitación	& And
secuencial	
operador paralelo	, And paralelo
{[]}	, And paralelo
	And paralelo
[]selección	Or-paralelo
[...]-> expresión guardada	: Guarda
Abstracción de proceso	Declaración gránulo
Instanciación de proceso	Llamada predicado

Figura 3. Primera equivalencia LOTOS--DRL

Una vez elegida la representación de las puertas, cada operador n-ario se traducirá a un predicado que se aplicará a n listas de comunicación. Como el resultado de aplicar un constructor n-ario a n expresiones de comportamiento LOTOS es otra expresión de comportamiento, el predicado emulador tendrá una lista de salida que devolverá la expresión de comportamiento resultante. En general, un predicado que emule un constructor n-ario tendrá n listas de comunicación de entrada y una de salida.

El siguiente paso es representar los eventos o acciones, es decir, lo que circulará por las listas de comunicación. En LOTOS una acción la representábamos por:

$g \ d_1 \ d_2 \ \dots \ d_n$

Donde g es el nombre de la puerta y los d_i son las declaraciones de valor y variable. Vamos a representar cada acción por un término lógico que circulará por las listas de comunicación. Estos términos deben contener toda la información referente a la acción. Veamos una primera representación:

```
a!3 ?x:nat [x<3]
t(a, [snd(3), rcv(x, nat), [x<3]])
```

Cada término de este tipo pasará por las listas de comunicación que representan las puertas y por los distintos operadores que combinan estas listas, pudiendo sufrir transformaciones por el paso de los operadores, hasta llegar al entorno más exterior o hasta un operador de

ocultación. Para la sincronización utilizaremos la técnica de mensajes incompletos, incorporando una variable de sincronización sin instanciar al término, que será instanciada en el momento en que se produzca la sincronización. Con esta modificación el término general que representará una acción tendrá la siguiente estructura:

```
t(Puerta, Sinc, Lista_oper, Lista_tipos, Lista_vars, Guarda)
```

La implementación completa de una acción consistirá en enviar este término por la lista de comunicación y esperar a que la variable Sinc sea instanciada. Tenemos ya un método para implementar el proceso básico LOTOS, que es la acción. El siguiente paso, que además es inmediato, es implementar la acción prefija, que es el único operador al que hemos encontrado equivalencia directa a través del And secuencial de DRL (&).

Sea la expresión $a:1 \ ?x:nat \ [x<3]; B$ y S la lista de comunicación que representa la expresión de comportamiento de B . Si queremos que st represente el comportamiento de la expresión en conjunto tendremos:

```
St=[Cab|Cola], evento(a, [snd,rcv], [nat,nat], [3,x], [x<3], Cab) & Cola=S.
```

Donde evento se define:

```
mode evento(?, ?, ?, ?, ^).
```

```
evento(G, Loper, Lsort, Lvar, Guarda, Sal) <- Sal=t(G, Sincro, Loper, Lsort, Lvar, G),  
data(Sincro).
```

A continuación describiremos la implementación del constructor paralelo.

```
mode par(s1?, s2?, puertas?, stream_sal^).  
par([], [], S, []):  
par([t(Gate, Sinc, Lop, Lso, Lva, G|R1), S2, Sel, S] <- not( in(Gate, Sel) ):  
S=[t(Gate, Sinc, Lop, Lso, Lva, G) |R], par(R1, S2, Sel, R).  
par(S1, [t(Gate, Sinc, Lop, Lso, Lva, G|R2), Sel, S] <- not( in(Gate, Sel) ):  
S=[t(Gate, Sinc, Lop, Lso, Lva, G) |R], par(S1, R2, Sel, R).  
par([t(Gate, ...) |R1], [t(Gate, ...) |R2], Sel, S] <- in(Gate, Sel):  
S=[t(Gate, ...) |R], par(R1, R2, Sel, R).
```

Cada constructor se traduce a un predicado que implemente la semántica operacional del constructor LOTOS. Como el constructor es binario deberá tener dos listas de comunicación de entrada y una de salida. Además llevará un argumento que indicará las puertas en las que se realiza sincronización. Existe una cláusula para cada uno de los casos que existían en la definición de la semántica operacional del operador paralelo. Las dos primeras cláusulas se refieren a cuando los procesos evolucionan sin sincronización y la tercera a cuando sincronizan.

3.3. HERRAMIENTAS DE ANÁLISIS

Las herramientas de simulación y validación permiten tener una visión del comportamiento del sistema conforme se va construyendo. Actualmente están orientadas a especificaciones DRL y SDL. El simulador facilita la construcción incremental de la especificación SDL, permitiendo que el diseñador pruebe diferentes evoluciones del sistema de forma interactiva. El validador intenta realizar un análisis de todas las posibles ejecuciones, tratando de detectar situaciones problemáticas. Ambas herramientas se construyen sobre la implementación distribuida de DRL para aprovechar los recursos disponibles en una red local.

El simulador es una herramienta interactiva, que al mismo tiempo que reproduce algunos posibles comportamientos de sistema efectúa un análisis de corrección para propiedades tales como el bloqueo, los comportamientos sin definir o la violación de asertos. Ofrece tres formas de evolución diferentes:

- selección aleatoria de una transición entre las disparables
- ejecución paso a paso guiada por el usuario
- ejecución paso a paso reproduciendo una traza generada por el validador.

Una de las principales características que presenta SDL es que proporciona conceptos estructurales que facilitan la especificación de un sistema mediante refinamiento sucesivo. El simulador se adapta perfectamente a esta estrategia de diseño, permitiendo probar de forma interactiva las especificaciones incompletas que se van desarrollando. Por otra parte, el simulador sirve de complemento al validador pues permite reproducir los escenarios conflictivos detectados durante la validación del sistema. De esta forma se pueden detectar y corregir las causas que produjeron estos escenarios.

La validación automática permite encontrar secuencias de ejecución del modelo que violan algunas propiedades deseables en el mismo y produce trazas de ejecución que puedan ser utilizadas en el proceso de simulación interactiva. Los requisitos de corrección particulares se espresan con referencia a la especificación SDL. Además, este lenguaje se utiliza para indicar la forma de optimizar la validación para la especificación concreta que se analiza.

La validación se realiza por análisis de alcanzabilidad [23], y se refiere a las llamadas propiedades de seguridad. La implementación actual abarca las siguientes propiedades:

- Comportamiento indefinido
- Inconsistencia en variable
- Bloqueo
- Código no alcanzable
- Propiedades particulares

Respecto a estas propiedades, las que necesitan mayor referencia a la especificación SDL son el bloqueo y las propiedades particulares. La detección de bloqueo supone la definición de situaciones de terminación aceptables para el sistema, tales como situaciones de inactividad en espera de que se soliciten servicios. Para ello se incorpora una notación de definición de estados finales válidos en el lenguaje de validación.

El esquema de traducción de SDL a DRL proporciona suficiente información para comprobar las condiciones particulares. Para establecer asertos sobre procesos concretos se puede acceder a la cola de señales del proceso, el estado respecto a la especificación SDL, las variables locales (en forma de lista), los PIDs Self, Parent, Offspring y Sender (en forma lista). Para establecer invariantes del sistema y condiciones sobre alcanzabilidad de estados, se puede acceder además al número de procesos de cada tipo, el estado de cada bloque (próximo PID de los procesos según su tipo, número de instancias de cada tipo de proceso en el bloque, lista de pares (PID, FIFO) y las rutas de señales de los procesos).

El validador se ha construido como un programa distribuido, que permite explotar los recursos disponibles en cada ejecución de EVP. De esta forma se puede repartir tanto el trabajo de análisis de estados como el espacio de almacenamiento de estados visitados. La tarea de análisis en paralelo se ha mostrado como una técnica de análisis parcial que proporciona mejor calidad de validación que el análisis exhaustivo cuando se trata de sistemas con un gran número de estados. El uso del almacenamiento distribuido permite atacar problemas de mayor tamaño que los analizables en una única estación de trabajo.

5. CONCLUSIONES Y TRABAJO FUTURO

Hemos presentado las principales características de un entorno para la integración de diferentes técnicas de descripción formal para el desarrollo de software para sistemas distribuidos. La implementación actual del mismo se concreta en EVP, un entorno de especificación y validación de protocolos para el proyecto TEMA. El entorno actual consta de un Núcleo Lógico Distribuido, Traductores de SDL y LOTOS, un Simulador y un Validador de especificaciones. Todas estas herramientas están integradas bajo una interfaz gráfica de usuario. Además se está definiendo un lenguaje de validación que permita expresar las propiedades de cada sistema que se pretendan validar.

El entorno se ha aplicado para el análisis de protocolos de nuevo diseño, como los protocolos AVP y MCS para aplicaciones multimedia. En ambos casos se utilizan tanto SDL como DRL en la construcción de las especificaciones formales.

Actualmente se trabaja en una serie de ampliaciones de la herramienta, entre las que cabe destacar las siguientes: uso de DRL como lenguaje de cooperación para unir especificaciones parciales realizadas con diferentes TDFs, incorporación de definiciones de datos con ACT-ONE y ASN.1, análisis de propiedades de viveza y nuevos traductores.

6.- REFERENCIAS

- [1] Agha, G. 1986. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press.
- [2] Azema P., Lloret J. C., Papapanagiotakis G., Vernadat F. "Estelle Validation and Prolog Interpreted Petri Nets". The Formal Description Technique Estelle. North-Holland, 1989.
- [3] CCITT. Specification and Description Language (SDL). Recommendation Z.100. 1992.
- [4] Clark K.L., Gregory S. "Parlog: parallel programming in logic". ACM Transactions on Programming Languages and Systems, 1986.
- [5] Danne A., Bauner J., Ahlberg I. "Prototyping Cordless Using Declarative Programming". XIV International Switching Symposium (IEICE). Yokohama, Japan. October 1992.
- [6] Davison A.. "Representing Petri nets in Parlog". The Parlog Group. Imperial College. 1987.
- [7] Díaz M., Troya J.M.. "A Parlog Based Real-Time Distributed Environment". Future Generation Computer Systems 9, pages 201-218. North-Holland. 1993.
- [8] Díaz M., Merino P. y Troya J.M. "Desarrollo de protocolos basado en Lenguajes Lógicos Concurrentes". Escuela de verano AEIA 1993.
- [9] Díaz Michel, Vissers C., Ansart J. "Sedos Software Environment For The Design Of Open Distributed Systems". The Formal Description Technique Lotos. North-Holland. 1989.
- [10] Dotan Y., Arazi B. Using Flat Concurrent Prolog in System Modeling. IEEE Transactions on Software Engineering, Vol. 17, Nº 6, June 1991.
- [11] Elshewy, M.A. "Robust Coordinated Reactive Computing in Sandra". PhD. SICS, 1990.
- [12] Gilbet D.. "A LOTOS to PARLOG Translator". Formal Description Technique. K. J. Turner (Ed). North-Holland. 1989.
- [13] GISUM(Grupo de Ingeniería del Software). J. Aldana, E. Alba, V. Benjumea, C. Canal, M. Diaz, L. Fuentes, M.M. Gallardo, P. Merino, A. Nebro, M. Roldan, B. Rubio, E. Pimentel y J.M. Troya. Dpto. Lenguajes y C. Computación. Universidad de Málaga. "EVP: Un Entorno Declarativo Distribuido para Especificacion y Validacion de Protocolos. Manual de Usuario". Proyecto Tema. December 1994.
- [14] Gregory S. , Neely R., Ringwood G. "Parlog for Specification Verification and Simulation". 7th International Symposium on Computer Hardware Description Languages and their Applications". Tokyo. August 1985.
- [15] ISO. LOTOS-A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. ISO IS 8807. 1989.
- [16] ISO. ESTELLE- A Formal Description Technique Based on State Transition Model. ISO 9074. 1989.
- [17] Madelaine E., Vergamini D. "AUTO: Averification Tool for Distributed Systems Using Reduction of Finite Automata Networks". Formal Description Techniques II. North-Holland. 1990.
- [18] Persson M., Ödling K., Eriksson D. "A Switching Software Architecture Prototype Using Real Time Declarative Language". XIV International Switching Symposium (IEICE). Yokohama, Japan. October 1992.
- [19] Shapiro E. "Concurrent Prolog: A Progress Report". IEEE Computer, 1986.
- [20] Sidhu D.P., "Protocol Verification via Executable Logic Specifications" Proc. III IFIP Symposium on Protocol Specification, Testing, and Verification, North-Holland, 1983.
- [21] The SPECS Consortium. R. Reed, W. Bouma, J.D. Evans, M. Dauphin, M. Michel eds. "Specification and Programming Environment for Communication Software". North-Holland. 1993.
- [22] Ueda K. "Guarded Horn Clauses". Ph.D. Thesis, Graduate School, University of Tokyo, 1986
- [23] West C. H.. "General Technique for Communication Protocol Validation". IBM J. Res. Develop. 22. July 1978.