Entorno de Desarrollo Ptolemy

Salguero, Federico *Osio,Jorge* *Gastaldi, Guillermo* *José Rapallini*
ryotech01@yahoo.com josio@gioia.ing.unlp.edu.ar josrap@ing.unlp.edu.ar

AreaCodiseño Hardware/Software-CeTaD-Dto Electrotecnia-Facultad de Ingenieria-UNLP

1. INTRODUCCIÓN: PTOLEMY

Ptolemy es un entorno de software de libre difusión desarrollado por la Universidad de Berkeley en EE.UU[1].
podemos referirnos a él como un entorno de programación visual que soporta jerarquía. También decir que se trata de un lenguaje de coordinación, aunque no seria exacto definirlo como un lenguaje. [2]
Se entiende por lenguaje al conjunto de símbolos y reglas para combinarlos (su sintaxis), y reglas para interpretar combinaciones de dichos símbolos (su semántica).
Un lenguaje podría especificar solamente la interacción entre módulos computacionales y no la computación desarrollada por dichos módulos. Ptolemy no sólo se ocupa de proveer al usuario de métodos de interacción, sino que provee una interfase a un lenguaje anfitrión que especifica la computación de dichos módulos.
Dentro del diseño digital, el Codiseño HW/SW se ocupa del diseño simultáneo de sistemas con componentes hardware y componentes software, buscando explotar las ventajas que ofrece cada alternativa. La idea es evitar el aislamiento entre los diseños hardware y software, permitiendo que estos se desarrollen en paralelo.
El papel que juega Ptolemy en la problemática del Codiseño, es que gracias a sus características jerárquicas se presenta como un marco especial para el modelado, simulación y síntesis de software de sistemas embebidos, de tiempo real y concurrentes.[3]
Su versatilidad se basa en la posibilidad de realizar descripciones heterogéneas utilizando diferentes *semánticas de descripción*. Ptolemy interpreta las semánticas de ejecución e interacción de sus componentes a través de los llamados *dominios* (ambientes de diseño especificados). Se asocia a cada dominio un Modelo de Computación (MoC) específico que se ocupa de definir formalmente la semántica del comportamiento de sus componentes.
Luego, la *semántica de descripción de un lenguaje* ó MoC, se define como el conjunto de leyes que gobiernan la interacción de los componentes del modelo ejecutable[3].
Pueden o no estar relacionados a la tecnología de implementación

2.EI KERNEL DEL PTOLEMY

Generalidades y terminología.

Ptolemy se basa en un Núcleo (Kernel) constituido por una familia de definiciones de clases C++. Los dominios son definidos a través de la creación de nuevas clases C++ derivadas de las clases base del Kernel[4]. Los dominios pueden operar en dos modos:

- Simulación - Un organizador invoca segmentos de código en un orden apropiado al modelo de computación.
- Generación de código – Los segmentos de código son conjuntamente agrupados en un lenguaje arbitrario para producir uno o más programas que implementen la función especificada.

En Ptolemy un sistema se describe en forma de diagrama de bloques interconectados, creados por el usuario. Para facilitar la creación de un sistema Ptolemy provee al usuario de librerías de bloques muy completas.
Para las interconexiones se pueden usar dos tipos de bloques, *Estrellas* y *Galaxias*. Las Estrellas son los bloques fundamentales de diseño, ya sea, creadas por el usuario o las presentes en las amplias librerías que provee Ptolemy . Una Galaxia es un bloque de más alta jerarquía que puede estar formado por Estrellas o por otras Galaxias.
De esta manera se pueden representar sistemas muy complejos mediante muchos subsistemas anidados jerárquicamente. A un sistema completo en Ptolemy se lo denomina *Universo* y describe una aplicación completa.
En los bloques las interfaces de entrada y salida son llamadas *Portholes* (puertos).
Las interconexiones entre bloques se realizan conectando los Portholes de los bloques mediante

cables o arcos. Los datos que se pasan entre bloques mediante las interconexiones son llamados Particulas (*Particles)* [4].
En Ptolemy un Dominio especifica un modelo de computación. Estos Dominios, como se dijo, representan diferentes estilos de descripción de sistemas digitales.
La semántica de un dominio es definida por clases que manejan la ejecución de una especificación. Estas clases pueden invocar a un simulador, generar código, o pueden invocar a un compilador sofisticado.
Un Target (Objetivo) es la clase base de más alto nivel en la ejecución. Un Target realizará su función vía un Scheduler (Planificador). El Scheduler define la semántica operacional de un Dominio controlando el *orden de ejecución* de un modelo funcional. Puesto más sencillo, la función del Planificador es manejar el orden de invocación de cada bloque en una aplicación.
En los dominios de simulación un Scheduler invoca los métodos de ejecución de los bloques del sistema y estos métodos llevan a cabo la función asociada con el diseño. En los dominios de generación de código, el Scheduler invoca también métodos de ejecución de bloques, pero estos métodos de ejecución sintetizan código en algún lenguaje. Esto es, generan código para realizar alguna función en vez de realizar la función en sí misma. Por otro lado, el Target es responsable de generar el código de conexión entre bloques (sólo si es necesario). Este mecanismo es muy simple e independiente del lenguaje[3].
El usuario puede elegir un Target para un universo o una galaxia en un dominio especifico y así definir el mecanismo en que se ejecuta un sistema.
Ptolemy tiene una característica muy importante y es que los dominios pueden entrelazarse jerárquicamente para trabajar juntos, es decir, es posible anidar bloques pertenecientes a distintos dominios, este mecanismo se denomina Wormhole. [4]

3. ESTRUCTURA INTERNA DE PTOLEMY

El uso típico del Ptolemy involucra el inicio de dos procesos UNIX al correr la interfase gráfica interactiva pigi (Ptolemy interactive graphical interface). El primer proceso consiste en la interface del usuario llamada vem y la base de datos de diseño llamada oct; mientras que el otro proceso contiene el Kernel del Ptolemy. Una alternativa es correr el Ptolemy como un proceso único, prescindiendo de la interface gráfica del usuario. En este caso, el intérprete de texto está basado en la herramienta de lenguaje de comandos ptcl (Ptolemy Tool Command Language). Ambas posibilidades se esquematizan en la figura 1 .

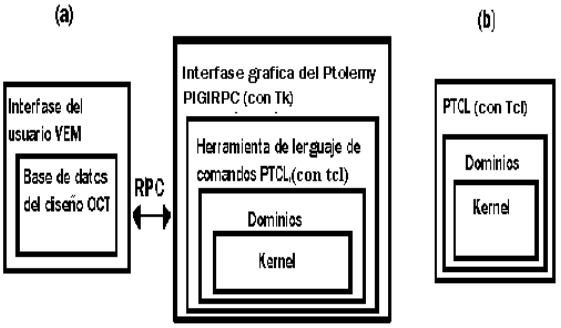

Figura 1 . Organización en Ptolemy, muestra las dos ejecuciones posibles.

Los programas ejecutables **pigirpc** ó **ptcl**, pueden ser configurados para incluir cualquier subconjunto de dominios disponibles.
La representación de los dominios desarrollados por Berkeley se muestra en la figura 2.

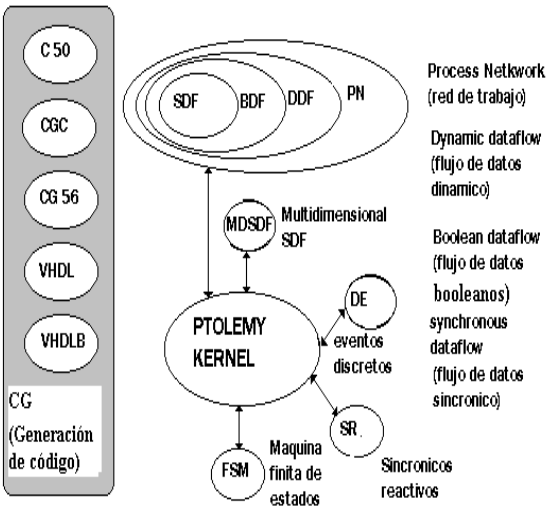

Figura 2 . Dominios en ptolemy

Estos son los dominios que representan estilos diferentes de descripción de sistemas digitales.
figura.

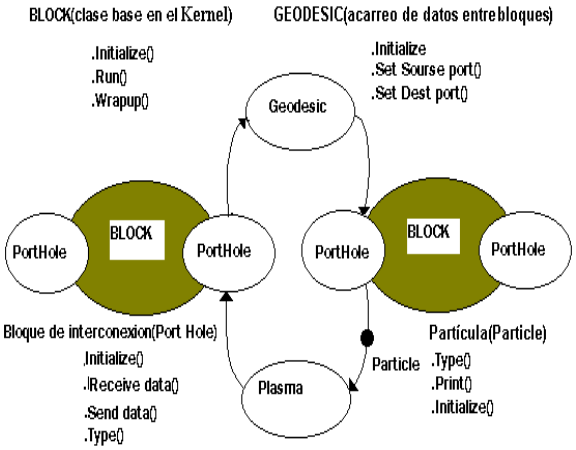

Figura 3. Objetos Block en Ptolemy, envían y reciben datos encapsulados en partículas.

Una clase base en el Kernel, llamada **Block** (Bloque), representa un *objeto* en esta red. Así, el *objeto* **Bloque** en Ptolemy puede enviar y recibir datos encapsulados en **particles** (partículas), o sea datos en algún formato, por ejemplo en punto flotante. Lo hace a través de una clase base definida en el Kernel de Ptolemy llamada **Portholes** (Puertos). El transporte de información lo maneja la clase base **Geodesic**, y la clase base, que sirve como repositorio de **partículas** de cualquier tipo para su posterior uso, se llama **Plasma**.

No todos los dominios usan estas clases, pero la mayoría de los corrientes lo hacen utilizando esta infraestructura eficientemente.

La figura 3 muestra algunos de los *métodos* [3] representativos definidos en las clases bases que se muestran. En el ejemplo, *inicialización* (initialize), *ejecución* (run) y *empaquetamiento* (wrapup) están en la clase **Bloque**. Estos proveen una interface a cualquier funcionalidad del *bloque*.

Los bloques pueden estar organizados jerárquicamente como se muestra en la figura 4.

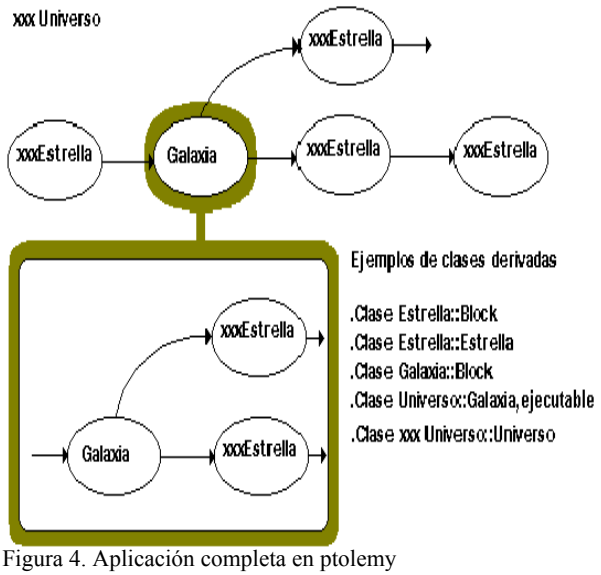

Figura 4. Aplicación completa en ptolemy

En Ptolemy, los niveles más bajos de una jerarquía corresponden a los derivados de la clase base **Star** (Estrella) del Kemel. Luego una **estrella** derivada de la clase base **Star**, es la unidad indivisible de computación en una aplicación Ptolemy. Cualquier simulación en Ptolemy finalmente consiste en ejecutar los *métodos* de las **estrellas** usados para definir la simulación.

La clase base del Kernel que representa una **galaxia** (red de *bloques*) se llama **Galaxie** (Galaxia), y pueden anidarse entre sí como en la figura 4. La representación más alta de un sistema es un **universo** (derivado de la clase base **Universo** -Universe-), y corresponde a una aplicación completa en Ptolemy como se aclarara oportunamente.

Otra de las características del Kernel del Ptolemy es que no define modelo de computación (MoC) alguno. La razón de esto ultimo se encuentra en que el espíritu del Ptolemy es el de mezclar MoCs, y no el desarrollar una técnica general. [3]

Lo que se debe resaltar es que la definición de la semántica de un modelo de computación se deja libre a cada dominio en particular.

Un **Objetivo** (Target: clase base de más alto nivel en la ejecución) es similar a un **Bloque**, tiene *métodos* llamados *setup, run, wrapup*. Para definir un dominio de simulación de nombre cualquiera, por ejemplo, uno puede definir al menos un *objeto* derivado del **Objetivo** que corra la simulación.

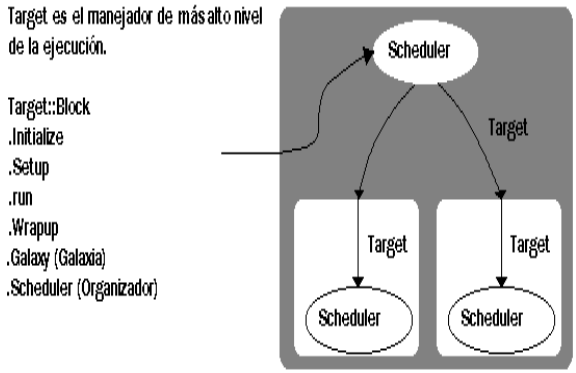

Figura 5 . Un Target, maneja la simulación

5. MOTIVACIÓN DE PTOLEMY

Heterogeneidad vs. Semántica Unificada.

Una de las aplicaciones de las soluciones heterogéneas se observa en los sistemas de procesamiento de señales. Estos son frecuentemente sistemas embebidos y su implementación mezcla tanto partes software como hardware. Incluso puede darse el caso de un sistema que presente heterogeneidad dentro de las partes hardware y software.

Para el diseño de estos sistemas se puede encarar el problema desde dos puntos de vista diferentes. Uno seria utilizar una técnica unificada, utilizando una semántica rica y consistente para la especificación del sistema completo. El otro punto de vista seria utilizar una técnica heterogénea, que combine semánticas disjuntas. Ptolemy adopta una técnica heterogénea por el hecho de que hay una gran diversidad de estilos de diseño que son comúnmente utilizados y esto haría más laboriosa una solución unificada. [6]

Todo el énfasis de Ptolemy se centra en el soporte de Heterogeneidad. Esto es, permitir al diseñador combinar un conjunto de descripciones (dominios) para modelar subsistemas, pero además sin limitar estas descripciones. En cambio, existe la posibilidad de modificar la semántica existente y crear nuevas descripciones. Ptolemy puede ser visto como un lenguaje de coordinación, pero desde ya no es lenguaje en el sentido convencional. Sería más acertado pensar en Ptolemy como un Entorno de coordinación.

El termino Generalidad madura el concepto de técnica unificada y puede verse como la contra cara de la aproximación heterogénea. Es una forma distinta de pensar un lenguaje de coordinación. En este caso los esfuerzos se concentran en combinar Modelos especializados en uno más general con semántica unificada. Estos tipos de lenguajes de coordinación poseen una extensa sintaxis y una gran riqueza de semántica. Sin embargo, el análisis de sistemas con este tipo de descripciones más generales es difícultoso. Incluso se hace muy laboriosa la implementación de compiladores eficientes para este tipo de descripción.

En definitiva, en lo que se refiere a Ptolemy, el equipo de Berkeley ha optado por la aproximación heterogénea, y entre los motivos de tal decisión se toma en cuenta que:

- La complejidad y diversidad de los diseños se incrementará con el tiempo.
- La practicidad que se logra especializando un lenguaje de especificación más general.
- Dejar abierta la posibilidad de implementar futuros lenguajes y MoC´s.

El soporte de heterogeneidad se logra con tecnología de Software OO y la utilización de dicha tecnología le agrega otras características a Ptolemy. A saber:

- **Agilidad.** Al soportar distintos modelos de computación, cada subsistema podrá ser simulado e **implementado en un prototipo** de una manera apropiada y natural a ese subsistema.
- **Extensibilidad.** Se soportará la integración fronteriza de nuevos modelos computacionales, y existiría la posibilidad de permitirles interoperar con modelos existentes sin efectuar cambios en los mismos.
- **Amigabilidad.** Utiliza una interface gráfica moderna con un estilo de representación de diagramas en bloque con organización jerárquica. [6]

6. MODELOS DE COMPUTACIÓN

6.1. Tratamiento del tiempo en distintos MoC´s

Cuando se habla de concurrencia generalmente se incurre en el error de pensar que el significado de la palabra implica la ocurrencia de dos o más eventos “al mismo tiempo”. Pero la definición debe ser más abarcativa, es por eso que debemos pensar en concurrencia como “simultaneidad de ejecución”. De esta manera aunque un MoC no tenga una noción del tiempo explícita, no quiere decir que sea incapaz de describir procesos

concurrentes. Es más, los MoC´s Dataflow son concurrentes sin tener noción del tiempo.
En el caso del MoC Eventos Discretos (ED), el tiempo es una parte integral del modelo. Los eventos contendrán una marca de tiempo, siendo este un indicador del instante en el cual los eventos ocurren dentro del modelo. Un simulador de ED manejará una cola de eventos ordenados según su marca de tiempo para su ejecución. Llegados a este punto debe distinguirse entre tiempo real y tiempo de simulación. Tiempo real es el tiempo que se emplea en la ejecución del modelo, mientras que internamente un simulador ED lleva una noción precisamente del tiempo simulado de ejecución de los bloques individuales. La marca de tiempo puede ser un numero entero, un número de punto flotante, o una estructura de datos cualquiera, representando no sólo el avance del tiempo, sino quizás también la secuenciación de Micropasos dentro de cada instante. Bajo estas consideraciones, las marcas de tiempo se incrementarán monótonamente en la medida que los eventos sean vistos por los componentes particulares. Este tipo de ordenamiento temporal puede resultar costoso computacionalmente, ya que requiere una coordinación entre las partes de simulación muy ajustada.
En el caso de los lenguajes sincrónicos, el tiempo progresa en saltos discretos a los que se denomina "pulsos". Por lo que dos eventos serán concurrentes si ocurren en el mismo pulso. En un modelo de estas características, los eventos ocurren regidos por un reloj patrón. Entonces los eventos pertenecientes a pulsos de reloj diferentes son ordenados globalmente. Sin embargo, aquellos que fueran simultáneos, pueden estar totalmente ordenados, parcialmente ordenados o desordenados. Las últimas consideraciones dependen del MoC particular en uso.
Así es como todas las señales (secuencias de eventos) tendrán eventos en todos los pulsos de reloj.
Los simuladores se simplifican notablemente al no haber ordenamiento por marcas de tiempo, y se los llama comúnmente "manejados por ciclos". Un ciclo constituye el procesamiento de todos los eventos para un dado pulso de reloj. El orden en que los eventos son procesados está determinado solo por la precedencia de datos.
Un modelo manejado por ciclos es ineficiente en el caso que se desee describir un sistema en el cual todos los eventos no ocurran en la misma razón de tiempo. Para sistemas multitasa basados en ciclos todos los eventos n-ésimos en una señal se alinean con los eventos en otra. Entonces, nuevamente se presentan problemas cuando las tasas de muestreo entre las distintas señales sean irregulares.
En algunos lenguajes sincrónicos como Esterel, todas las señales están explícitamente acompañadas por una señal de reloj asociada, y esta tiene un significado relativo a las otras señales de reloj. Cuando se comparan dos señales, las señales de reloj asociadas indican qué eventos son simultáneos y cuáles preceden o siguen a otros.
Por último, un ordenamiento parcial de eventos, como los que llevan a cabo los MoC´s Dataflow, ayudan a mantener independiente el modelo de la implementación, ya que se evita la sobre especificación del mismo. [6]

Un Modelo de Computación (MoC) se define como la semántica de interacción definida entre los módulos y componentes. Una gran variedad de MoC´s se utilizan tanto en programación de computadoras como en diseños de sistemas electrónicos.
Algunas clases de MoC´s son:

- **Imperativos:** En un Modelo de Computación imperativo, para llevar a cabo una tarea, los módulos se ejecutan secuencialmente.
- **Finite state machine(FSM):** En un Moc FSM , un especificación enumera un conjunto de estados en que puede estar un sistema, junto con los requisitos que se tienen que cumplir para la transición de un estado a otro.
- **Dataflow:** En un MoC Dataflow , los bloques reaccionan cuando tienen datos disponibles en sus entradas llevando a cabo alguna operación y produciendo datos en las salidas respectivas. La comunicación entre módulos se da vía flujos de *datos* llamados *Tokens.* Cada Token es una estructura de datos arbitraria.[6] Dentro del modelo Dataflow, un programa es especificado por un grafo (planteo visual del sistema), donde los nodos representan las computaciones (bloques funcionales o *actors*) y los arcos representan flujos de *tokens* de datos. Los grafos son generalmente jerárquicos, donde un actor en un grafo puede ser el representante de otro grafo mas específico. Una propiedad clave en procesos Dataflow, es que la computación consiste en disparos atómicos de los nodos del grafo. Dentro de un disparo, puede pasar cualquier cosa.

- **Eventos discretos:** Los bloques reaccionan a eventos que ocurren luego de un dado

instante de tiempo, y producen otros eventos ya sea en el mismo instante o en algún instante de tiempo futuro. La ejecución es cronológica.

- **Lenguajes sincrónicos:** En los lenguajes sincrónicos los bloques reaccionan simultáneamente a un set de eventos de entrada e instantáneamente producen eventos de salida.

Es importante conocer la diferencia entre un MoC y la manera en que un MoC pueda ser implementado. Por ejemplo, mientras los primeros dos MoC´s son fundamentalmente secuenciales, los últimos tres son fundamentalmente concurrentes, y es posible usar los dos primeros en maquinas paralelas y los tres últimos en maquinas secuenciales.

También se debe entender la diferencia entre un MoC y un lenguaje, ya que la sintaxis es una parte importante de un lenguaje pero no de un Moc. Por ejemplo VHDL puede ser usado en un estilo Imperativo o de Eventos Discretos.

Los lenguajes típicamente basados en un solo MoC pueden también ser usados para implementar otros. Por ejemplo C que es típicamente Imperativo, puede ser usado para implementar un MoC Dataflow. [6]

6. 3. Redes de procesos Dataflow

6.3.1. Dominio SDF en Ptolemy

Es el dominio más antiguo y maduro de Ptolemy, siendo un caso especial del modelo de computación Dataflow desarrollado por Dennis. La especialización del modelo de computación corresponde al caso de los grafos Dataflow en donde el flujo de control es completamente predecible en tiempo de compilación. Es además un subdominio del dominio DDF, y debido a esto, cualquier estrella o Target que funcione bajo SDF debe funcionar en DDF. Las estrellas consumen y generan un número conocido y fijo de tokens de datos en cada invocación. Luego, teniendo en cuenta que la información es estática, un Scheduler de SDF puede construir un organizador que pueda ser usado repetidas veces, aunque este tipo de organizadores no podrá ser usado por las estrellas DDF. El modelo SDF es apropiado para el diseño de sistemas de procesamiento de señales multitasa con frecuencias de muestreo relacionadas por múltiplos enteros Su extensa y rica librería de estrellas incluye filtros FIR polifásicos reales y complejos Dentro de la librería de Demos se destacan ejemplos de codificación del habla, diseño de filtros, estimación espectral, síntesis de sonido, procesamiento de imágenes, codificación de video y análisis y síntesis de bancos de filtros. Una ventaja que ofrece el dominio SDF es que es fácil de programar, y se debe a que la disponibilidad de tokens de datos es estática y no necesita de un chequeo previo. Pero la característica más importante es su eficiencia en tiempo de ejecución, ya que el ordenamiento de las invocaciones de los bloques se determina estáticamente. Además soporta organización paralela automática.

En varios de los entornos existentes, lo que ocurre sólo puede ser especificado en el lenguaje anfitrión (lenguaje madre) con semántica imperativa, como c y c++. En el sistema Ptolemy, puede consistir en un cuanto de computación especificado con cualquiera de los varios modelos de computación con que se cuenta.

La mayoría sin embargo, puede ser descripta como casos especiales de redes de procesos dataflow, quienes son a su vez casos especiales de las redes de procesos de Khan.[4]

En las redes de procesamiento Dataflow, cada proceso consiste en una serie repetida de disparos de un actor de dataflow. De hecho, en la mayoría de los entornos de procesamiento de señales, uno de los principales objetivos es determinar estáticamente (en tiempo de compilación) la organización de los disparos de los actores. Los disparos se organizan en una lista (para un procesador) o un set de listas(más de un procesador).

En la figura 6 se muestra un organizador para un procesador simple de un grafo Dataflow.

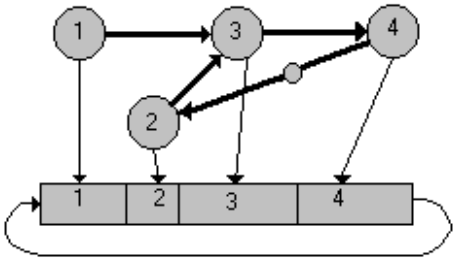

Figura 6 . Organizador de un grafo Dataflow.

La parte inferior de la figura, representa la lista de disparos que puede ser repetida indefinidamente. Un requerimiento básico para dicho organizador es que un ciclo a través del organizador debe devolver el grafo en su *estado* original (se define *estado* como el número de

tokens en cada arco). Esto no siempre es posible, pero cuando lo es, resulta en una simplificación considerable.
Synchronus dataflow (SDF) y ciclo-static dataflow[6] tienen particularmente la útil propiedad de poder encontrar un organizador estático finito, de manera de retornar al grafo a su estado original. Esto permite implementaciones extremadamente eficientes. Para modelos dataflow más generales es imposible estimar a priori si dicho organizador existe.

6.3.2. Dominio Dynamic Dataflow (DDF)

Es un modelo computacional manejado por datos, originalmente propuesto por Dennis. Su uso principal radica en el diseño de arquitecturas paralelas, sin embargo, también es apropiado como modelo de programación. Es también especialmente útil para el modelado de sistemas de procesamiento de señales que incluyan operaciones asincrónicas. El flujo de control predecible de SDF permite una organización eficiente, pero limita el rango de aplicaciones. En particular el flujo de control dependiente de datos es solo permitido dentro de los confines de una estrella. Puede decirse que DDF es un modelo cuya semántica permite el diseño de sistemas con dinámica en tiempo de ejecución. O sea, permite una organización dinámica en tiempo de ejecución, que, para ejecuciones largas que incluyan muchas iteraciones, podría decirse que es más pesado que la organización estática posible en SDF. La ventaja radica en que se obtiene un modelo de computación versátil como el de los lenguajes de programación convencionales. Soporta estructuras condicionales, iteración dependiente de datos y recursión verdadera.
Las estrellas en DDF son activadas por datos presentes a las entradas de los PortHoles

6.3.3. Dominio Boolean Dataflow (BDF)

Fue desarrollado por Joe Buck como parte de su tesis de Doctorado. Es un modelo intermedio entre SDF y DDF, en el cual se soporta un conjunto limitado pero importante de operaciones asincrónicas (análogas a las sentencias “if-then-else” y “case” en Lenguaje C), conservando casi todas las excelentes propiedades del dominio SDF.

6.4. Eventos Discretos

6.4.1. Dominio Eventos Discretos (DE)

Es un modelo relativamente maduro que utiliza un modelo de computación dominado por eventos. Es un modelo asincrónico como DDF, pero incorpora el concepto de tiempo global del sistema y ordena las invocaciones de los bloques de acuerdo a marcas de tiempo asociadas a las partículas. En el dominio DE, puede desarrollarse un sistema de simulación completamente general, por supuesto a expensas de pérdida en la eficiencia en tiempo de ejecución y en la facilidad y naturalidad en la programación para muchas aplicaciones. El dominio DE es apropiado por ejemplo para el modelado de redes de comunicación entre otras aplicaciones.

6.4.2. Modelo de computación de Eventos Discretos

El modelo de computación de eventos discretos se destaca por manejar eventos con marcas de tiempo. El rol del organizador es mantener una lista de eventos sorteados por marca de tiempo y procesar los eventos en orden cronológico. Las mayores dificultades conciernen:

1) Cómo se manejan eventos simultáneos (aquellos que tienen la misma marca de tiempo)
2) Cómo se manejan los lazos de realimentación de retraso nulo.

Una solución parcial, en algunos simuladores, es el retraso infinitesimal (todos los bloques de retraso nulo serán vistos como si aportaran al sistema un retraso infinitesimal). Los retrasos infinitesimales no son una solución completamente satisfactoria, por ejemplo en el caso en que el diseñador desee que C vea los dos eventos al mismo tiempo.
Por esto el dominio de eventos discretos en Ptolemy usa una solución diferente.
Los grafos que especifican programas de eventos discretos son ordenados topológicamente, y se asigna una prioridad a cada arco. El orden topológico está basado en la anotación de los nodos en el gráfico indicando si el nodo puede tener delay de valor cero para una entrada particular. Ignorando el lazo de realimentación, en la figura anterior se resuelven todas las ambigüedades. La organización topológica mostrará que B debe ser siempre invocado antes que C cuando ellos tengan eventos en sus entradas con idéntica marca de tiempo. [6]

7.CONCLUSIONES:

En este trabajo estudiamos y utilizamos el Entorno de Desarrollo Ptolemy. Del estudio podemos concluir que el Ptolemy es una herramienta que posee cualidades como la agilidad dada por los distintos modelos computacionales disponibles, la extensibilidad que puede realizarse mediante el agregado de código del usuario y la facilidad de uso que nos provee la interface grafica del entorno. Mediante el Ejemplo desarrollado, comprobamos lo estudiado. La implementación de la Transformada rápida de fourier nos permitió evaluar el desempeño de Ptolemy en el campo del procesamiento digital de señales y también obtuvimos resultados de simulación y código objeto en lenguaje C.

8. REFERENCIAS BIBLIOGRÁFICAS:

[1] UC Berkeley, Department of EECS, "The Ptolemy Project", http://ptolemy.eecs.berkeley.edu
[2] Guillermo Gastaldi, "Diseño de un medidor de impedancia en tiempo
real", UNLP, Departamento de Electrotecnia, Laboratorio CeTAD, marzo, 2002.

[3] User´s Manual: *Ptolemy User´s Manual*, Version 0.7, Linux, University of California at Berkeley, College of Engineering, Department of Electrical Engineering and Computer Sciences, 1997.

[4] Bilung Lee, "Fusing Dataflow with Finite State Machine", Ms Report, Department of electrical Engineering and computer Science, University of California, Berkeley, CA 94720,.

[5] Joseph Buck, Soonhoi Ha, Edward A. Lee, David G. Messerschmitt, "Ptolemy: A Framework for simulating and prototyping Heterogeneous System", Ms Report, Departament of electrical Engineering and computer Science, University of California , Berkeley, CA 94720, Agosto 31,1992.

[6] W.-T.Chang, S. HA, and E. A. Lee,"Heterogeneus Simulation", Ms Report, Departament of electrical Engineering and computer Science, University of California , Berkeley, CA 94720, Abril 2,1996.