ESTUDIO DE LA HERRAMIENTA PTOLEMY II PARA EL MODELADO Y DISEÑO DE SISTEMAS TELEMÁTICOS Y GENERACIÓN DE UNA METODOLOGÍA PARA LA CONSTRUCCIÓN DE ACTORES

Alejandro Barragán Cerquera
Marina Tejada Abella

Director:
Ing. Mario Fernando Solarte

UNIVERSIDAD DEL CAUCA
FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICAIONES
POPAYÁN
2001

CONTENIDO

LISTA DE FIGURAS

1. INTRODUCCIÓN

En el marco de la Ingeniería del Software se han buscado mecanismos que permitan la creación de programas de manera organizada y eficiente, empleando una colección de técnicas y convenciones predefinidas. Son conocidos los diferentes enfoques de Ingeniería del Software que encierran las diversas metodologías que se han planteado a través de los años para lograr este objetivo. Entre los diferentes enfoques se encuentran[1]:

- Análisis estructurado / Diseño estructurado (SA/SD)
- Desarrollo Estructurado de Jackson (JSD)
- Técnica de Modelado de Objetos (OMT)

entre otros.

Estas metodologías difieren en el énfasis relativo que se hace en los distintos componentes del modelado; cada cual tiene acogida dentro cierto tipo de aplicaciones y tendencias. Para este trabajo se tratara el enfoque orientado a objetos por ser el énfasis del proyecto.

Una metodología orientada a objetos presta su apoyo a todo el ciclo de vida del software, desde la formulación del problema hasta el análisis de requisitos, diseño e implementación, resultando con todos estos pasos y las características propias del enfoque, en un diseño limpio y fácil de comprender, que resulta más fácil de probar, mantener y extender que los diseños no orientados a objetos porque las clases de objetos proporcionan una unidad natural de modularidad.

En cada una de estas etapas la utilización de herramientas es un factor que valida los resultados y facilita la evolución eficaz del trabajo. Estas Herramientas pueden ser llamadas “herramientas de ayuda al desarrollo” y son un software que facilita la

producción de aplicaciones a medida. Existe una amplia gama de posibles productos que se pueden incluir en esta definición, desde los lenguajes de programación (incluyendo la combinación de algunos de ellos), hasta sofisticados y complejos productos[2].

Las siguientes herramientas se consideran actualmente como herramientas de ayuda al desarrollo[3]:

- Herramientas de ingeniería de software asistida por ordenador o CASE.
- Lenguajes de cuarta generación o 4GL.
- Otras herramientas

Es importante señalar que las fronteras entre unas y otras no siempre están claramente definidas.

Herramientas que comprenden algunas fases del ciclo de vida de desarrollo de software:

Herramientas de análisis y diseño. Permiten al desarrollador crear un modelo del sistema que se va a construir y también la evaluación de la validez y consistencia de este modelo. Proporcionan un grado de confianza en la representación del análisis y ayudan a eliminar errores con anticipación. Se tienen:

- Herramientas de análisis y diseño (Modelamiento).
- Herramientas de creación de prototipos y de simulación.
- Herramientas para el diseño y desarrollo de interfaces.

De forma general se ha podido observar que todas las herramientas están tendiendo a las siguientes líneas [3]:

1. Herramientas más abiertas para poder conectarse con mayor facilidad unas con otras.
2. Soporte para diferentes plataformas físicas y lógicas.
3. Mejor aprovechamiento de los recursos físicos y los servicios de red e interconexión.
4. Consistente aplicación de lo que es la tecnología de orientación a objetos.

5. Interfaz de usuario más amigable apoyándose en tecnologías multimedia.

El siguiente documento contiene una descripción de la herramienta Ptolemy II la cual será tratada como una herramienta de ayuda al desarrollo y que dentro del presente trabajo se tratará con la siguiente estructura:

El primer capítulo está constituido por esta introducción al trabajo de grado y la presentación de la respectiva monografía.

En el capítulo dos se describen los conceptos básicos de las tecnologías orientadas a objetos, las nociones de modelamiento y diseño y su importancia en los Sistemas Telemáticos, y por último se presenta y caracteriza a Ptolemy II como una herramienta que utiliza estos conceptos.

En el capítulo tres se hace una descripción de los componentes de la herramienta y su funcionamiento, los actores de Ptolemy II, los paquetes, describiendo en forma amplia el paquete Kernel, y por último las librerías que lo componen.

El capítulo cuatro describe los dominios o modelos de computación los cuales refieren la interacción entre los componentes de Ptolemy II, estos dominios serán explicados a fondo en el Anexo 1.

El capítulo cinco contiene la metodología para la construcción de actores de Ptolemy II, y la aplicación de esta metodología en la creación de un actor concreto.

Al final se darán algunas recomendaciones y se harán las conclusiones producto del desarrollo del trabajo que se expone.

Como anexos se entregaran los siguientes documentos:

Anexo 1. Dominios de Ptolemy II. En este anexo se hace una descripción detallada de los dominios existentes en Ptolemy II.

Anexo 2. Manuales. Este anexo contiene los manuales del usuario y del administrador. El primero da las indicaciones del manejo de la interfaz de usuario y

de las librerías. El segundo proporciona las pautas para la instalación de la herramienta así como la introducción de los nuevos actores.

Anexo 3. Librería de actores. Este anexo presenta los actores existentes en las librerías de Ptolemy II, así como una descripción de estos.

Anexo 4. Código del Actor Cuantificador Logarítmico. Este anexo contiene el código del actor construido utilizando la metodología para la construcción de actores de Ptolemy II.

Anexo 5. Diagramas de flujo de la metodología para la construcción de actores de Ptolemy II.

2. MARCO DE REFERENCIA

2.1. EL PARADIGMA ORIENTADO A OBJETO

El paradigma de orientación a objetos es un paradigma centrado en conceptos, que comprende los pilares (principios fundamentales) de abstracción, encapsulación, herencia y polimorfismo. Fue concebido originalmente por tres de los más prominentes filósofos griegos, Sócrates (470-399 AC), Platón (428-348 AC), y Aristóteles (384-322 AC) en la Teoría de las Formas [1]. El paradigma ha evolucionado desde su conceptualización original hasta llegar a ser una fuerza eje en la evolución de la ciencia, tecnología y cualquier otro dominio en el cual se aplique.

El paradigma de orientación a objetos, se enfoca en las características estructurales y de comportamiento de las entidades como unidades completas. El paradigma abarca y soporta los siguientes pilares:

La abstracción involucra la formulación de representaciones enfocándose en similitudes y diferencias de entre un conjunto de entidades y así definir una única representación que tenga las características que son relevantes en la definición de cada elemento del conjunto.

La encapsulación involucra el empaquetamiento de las representaciones enfocándose en el ocultamiento de detalles para facilitar la abstracción, donde las especificaciones son usadas para describir lo que son y hacen las entidades, y las implementaciones son usadas para describir cómo es realizada una entidad.

La herencia involucra el relacionar y reutilizar las representaciones existentes para definir nuevas representaciones.

El polimorfismo involucra la habilidad de nuevas representaciones para ser definidas como variantes de las ya existentes, donde las nuevas implementaciones son introducidas pero las especificaciones prevalecen de manera que una especificación tiene varias implementaciones

Estos pilares son usados para facilitar la comunicación, incrementar la productividad y consistencia, y habilitar la administración del cambio y complejidad dentro de los esfuerzos para la solución de un problema.

2.1.1. CONCEPTOS BÁSICOS DE LAS TECNOLOGÍAS ORIENTADAS A OBJETOS

La Tecnología de Objetos o la Orientación a Objetos comprende un conjunto de metodologías y técnicas que se aplican de manera similar en distintas tareas de la informática (mejorar las estructuras de datos, trabajo en redes de computadores, sistemas de bases de datos, entre otros), y que tienen como objetivo fundamental facilitar el trabajo de las personas que desarrollan estas tareas.

Este objetivo se verifica y se hace notar de manera considerable en los proyectos de desarrollo de sistemas grandes y complejos, más aun cuando se ha planeado desarrollar los sistemas para un ambiente de interfaz amigable con el usuario, donde se ha podido observar que el enfoque estructurado no presenta mayor desempeño. El objetivo también e verificado por las siguientes características de la OO: fácil mantenimiento y actualización de sistemas en uso, la utilización de conceptos de apariencia sencilla pero consolidada, y la orientación hacia una forma de pensar de un modo abstracto acerca de un problema empleando conceptos prácticos y no del mundo de la computación que obliga a pensar más en términos de la aplicación que en términos computacionales; .

El modelado y diseño orientado a objetos constituye una nueva forma de pensar acerca de problemas empleando modelos que se han organizado tomando como base conceptos del mundo real, entendiendo éste como el dominio que abarca

problemas, soluciones y esfuerzos para resolver los problemas [1]. EL mundo real incluye:

Cosas o entidades que tienen un propósito o rol dentro del mundo. Las entidades tiene características estructurales que representan lo que las cosas "saben" y características de comportamiento que representan lo que la cosa "hace". Una entidad agregada puede contener entidades componentes que son características estructurales y de comportamiento.
Relaciones entre entidades. Las relaciones pueden ser tratadas como características estructurales compartidas entre múltiples entidades que tienen una relación con alguna otra. Una entidad agregada puede contener entidades componentes que están relacionadas vía un atributo de la entidad agregada.
Ocurrencias entre entidades. Las ocurrencias pueden ser tratadas como características de comportamiento compartidas entre múltiples entidades que interactúan con alguna otra. Una entidad agregada puede contener entidades componentes que son manipuladas vía una operación de la entidad agregada.
Estos conceptos son fundamentales para la interacción con el mundo real.

La construcción fundamental es el objeto que logra incluir la estructura de datos con su comportamiento; estos objetos conforman la estructura del software el cual se organiza como una colección de estos objetos discretos.

2.1.1.1. OBJETOS Y CLASES

Los objetos son construcciones que representan las entidades. Los objetos encapsulan las características estructurales conocidas como atributos y las características de comportamiento conocidas como operaciones. Los atributos son construcciones que representan las características estructurales de las entidades y determinan los posibles estados de un objeto. Las operaciones son construcciones que representan las características de comportamiento de las entidades y determinan los comportamientos posibles de un objeto invocado, en respuesta a la recepción de un mensaje. Los objetos tienen identidad y son instancias de clases. Fundamentalmente, los objetos son entidades abstraídas que encapsulan un estado y un comportamiento.

Las clases son descripciones de objetos con una implementación en común. Las clases están ligadas a la implementación uniforme de las características estructurales y de comportamiento. Fundamentalmente, las clases son descripciones de objetos con atributos, implementación de operaciones, semánticas, asociaciones e interacciones comunes. Dentro de este concepto de clase se debe mencionar el de superclase y subclase, en donde, una subclase es una clase que se deriva o desciende de otra y hereda el estado y el comportamiento de esta. El término superclase se refiere a la clase que es el ancestro más directo y del cual se derivan las subclases.

2.1.1.2. ENLACES Y ASOCIACIONES

Los enlaces y asociaciones abstraen relaciones entre entidades.

Los enlaces son construcciones que representan las entidades que se relacionan a otras entidades. Los enlaces son instancias de asociaciones. Fundamentalmente, los enlaces son relaciones abstraídas, entre objetos.

Las asociaciones son descripciones de enlaces con una implementación común. Las agregaciones son asociaciones que especifican una relación de todo - parte entre las partes agregada y componente. Las composiciones son agregaciones con unas restricciones de tiempo de vida coincidente y fuerte propiedad entre las partes compuesta y componente. Las generalizaciones son asociaciones que especifican unas relaciones taxonómicas que relacionan construcciones de representación más generales y otras más específicas. El polimorfismo habilita a un mismo mensaje (interface de operación) para que pueda invocar el método apropiado (implementación de operación) basado en la clase del receptor cuando una instancia más específica es substituida por una más general. Fundamentalmente, las asociaciones son descripciones de enlaces con atributos, implementaciones de operación, semánticas, asociaciones e interacciones comunes.

2.1.1.3. ESCENARIOS E INTERACCIONES

Los escenarios e interacciones abstraen ocurrencias entre entidades

Los escenarios son construcciones de representación de entidades que son conducidas por una secuencia de intercambios de mensajes con otras entidades. Los escenarios son instancias de interacciones. Fundamentalmente, los escenarios son un intercambio de mensajes entre objetos abstraídos.

Las interacciones son descripciones de escenarios con una implementación común. Los intercambios de mensajes involucran a un remitente que es el que aplica una operación en un receptor enviando un mensaje (interface de operación), que invoca un método (implementación de operación) dentro del receptor. Fundamentalmente, las interacciones son descripciones de escenarios con una secuencia de intercambio de mensajes, clases y asociaciones en común.

2.1.2. OTROS CONCEPTOS ORIENTADOS A OBJETOS

Se amplían ahora los conceptos dados como principios fundamentales del Paradigma OO y se incluyen otras características y conceptos del enfoque orientado a objetos.

2.1.2.1. ABSTRACCIÓN

Consiste en centrarse en los aspectos esenciales inherentes de una entidad, e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir cómo debería ser implementado. El uso de la abstracción mantiene la libertad de tomar decisiones durante el mayor tiempo evitando comprometerse de forma prematura con ciertos detalles. La mayoría de los lenguajes modernos proporcionan abstracción de datos pero la capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional. El uso de la abstracción durante el proceso de análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar

decisiones de diseño o de implementación antes de haber comprendido el problema.

2.1.2.2. ENCAPSULACIÓN

Denominada también ocultamiento de información, consiste en separar los aspectos externos del objeto, los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás. La encapsulación evita que la implementación llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin afectar a las aplicaciones que la utilizan. La encapsulación no es exclusiva de los lenguajes orientados a objetos pero la capacidad de combinar la estructura de datos y el comportamiento en una única entidad hace que la encapsulación sea aquí más limpia y potente que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

2.1.2.3. HERENCIA

Consiste en compartir atributos y operaciones entre clases tomando como base una relación jerárquica. En términos generales se puede definir una clase que después se irá refinando sucesivamente para producir subclases. Todas las subclases poseen, o heredan, todas de las propiedades de su superclase y añaden, además, sus propiedades exclusivas. No es necesario repetir las propiedades de las superclases en cada subclase. La capacidad de sacar factor común a las propiedades de varias clases en una superclase común y de heredar las propiedades de la superclase puede reducir muchísimo la repetición en el diseño y en los programas siendo una de las ventajas principales de la orientación a objetos.

2.1.2.4. POLIMORFISMO

Significa que una misma operación puede comportarse de modos distintos en distintas clases. Un ejemplo de polimorfismo se da en las operaciones: una operación determinada se puede comportar de modo distinto para diferentes clases, a esto se le llama método, entonces, por el polimorfismo, los operadores orientados a objetos pueden ser implementados por más de un método.
En un lenguaje OO es el mismo objeto quien determina el método correcto para implementar una operación basándose en el nombre de la operación y en la clase de objeto que está siendo afectado. El usuario de una operación no necesita ser consciente del número de métodos que existen para implementar una operación polimórfica. Se pueden añadir nuevas clases sin modificar el código existente, siempre y cuando se proporcionen métodos para todas las operaciones aplicables a las nuevas clases.

2.1.2.5. IDENTIDAD

Quiere decir que los datos están cuantificados en entidades discretas y distinguibles denominados objetos. Los objetos pueden ser concretos o conceptuales. Cada objeto posee su identidad propia inherente, esto quiere decir que dos objetos serán diferentes aun cuando los valores de todos sus atributos sean idénticos; dentro de un lenguaje de programación cada objeto posee una denominación mediante la cual se puede hacer alusión a él de modo exclusivo. La denominación se puede implementar de distintas maneras que puede ser una dirección, un índice de una matriz o un valor exclusivo de un atributo. Las referencias a los objetos son uniformes e independientes del contenido de los objetos permitiendo crear colecciones mixtas de objetos.

2.1.2.6. CLASIFICACIÓN

Significa que los objetos con la misma estructura de datos (atributos) y comportamiento (operaciones) se agrupan para formar una clase. Una clase es una abstracción que describe propiedades importantes para una aplicación y que ignora el resto. La selección de clases es arbitraria y depende de la aplicación.

Toda clase describe un conjunto posiblemente infinito de objetos individuales. Se dice que cada objeto es una instancia de su clase y cada instancia posee su propio valor para cada uno de los atributos pero comparte los nombres de los atributos y las operaciones con las demás instancias de su clase; todo objeto contiene una referencia implícita a su propia clase.

2.1.2.7. COMBINACIÓN DE DATOS Y COMPORTAMIENTOS

Aquél que invoca una operación no necesita considerar cuantas implementaciones existen de una operación dada. El polimorfismo de operadores traslada la carga de decidir qué implementación hay que utilizar llevándola del código que hace la llamada a la jerarquía de clases. Por ejemplo, un código no orientado a objetos para visualizar el contenido de una ventana debe distinguir el tipo de cada figura y debe invocar el procedimiento adecuado para visualizarlo. Un programa orientado a objetos invocaría simplemente la operación dibujar para cada figura; la decisión del procedimiento que hay que utilizar es realizada implícitamente por cada objeto basándose en su clase. No es necesario repetir la selección de procedimiento cada vez que se invoque a la operación en el programa de aplicación. El mantenimiento es más sencillo por que el código que hace la llamada no necesita ser modificado cuando se añade una clase nueva. En un sistema orientado a objetos la jerarquía de estructuras de datos es idéntica a la jerarquía de herencia de operaciones.

2.2. MODELAMIENTO Y DISEÑO ORIENTADO A OBJETOS

La visión actual del desarrollo de software toma una perspectiva orientada a objetos. En este enfoque, el principal bloque de construcción de todos los sistemas software es el objeto o la clase. Para explicarlo de manera sencilla, un objeto es un elemento, extraído del vocabulario del espacio del problema o del espacio de la solución, una clase es una descripción de un conjunto de objetos similares. Todo objeto tiene identidad (puede nombrarse o distinguirse de otra manera de otros objetos), propiedades (generalmente hay algunos datos asociados a él), y comportamiento (se le pueden hacer cosas al objeto, y el a su vez puede hacer cosas a otros objetos).

Un modelo es una abstracción de algo, que se elabora para comprender ese algo antes de construirlo. El modelo omite detalles que no resultan esenciales para la comprensión del original y por lo tanto facilita dicha comprensión.

Los modelos se utilizan en muchas actividades de la vida humana: los arquitectos, los artistas, los ingenieros. Unos y otros abstraen una realidad compleja sobre unos bocetos, modelos al fin y al cabo.

Un modelo en OO (Orientación a Objeto) es una abstracción cerrada semánticamente de un sistema y un sistema es una colección de unidades conectadas que son organizadas para realizar un propósito específico. Un sistema puede ser descrito por uno o más modelos, posiblemente desde distintos puntos de vista.

La OMT (Técnicas de Modelamiento de Objetos), por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos OO: el modelo de objetos, que describe la estructura estática; el modelo dinámico, con el que describe las relaciones temporales entre objetos; y el modelo funcional que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de un modelo, se consigue una abstracción de la realidad que tiene en sí misma información sobre las principales características de ésta.

Esto lleva a la pregunta que cuestiona su utilidad, ¿para qué se modela?El objetivo definitivo del modelamiento es comprender mejor el sistema que se esta desarrollando.

A través del modelado se consiguen cuatro objetivos:

- Los modelos ayudan a visualizar cómo es o se quiere que sea un sistema.
- Los modelos permiten especificar la estructura o el comportamiento de un sistema.
- Los modelos proporcionan plantillas que guían en la construcción de un sistema.
- Los modelos documentan las decisiones que se han tomado.

El modelado es tanto para grandes como para pequeños sistemas, sin embargo, es absolutamente cierto que cuanto más grande y complejo es el sistema, el modelado se hace más importante ya que se construyen modelos de sistemas complejos por que no se puede comprender el sistema en su totalidad.

A través del modelo, se reduce el problema que se esta estudiando, centrándose sólo en un aspecto a la vez.

2.2.1. IMPORTANCIA DEL MODELAMIENTO Y EL DISEÑO EN LOS SISTEMAS TELEMÁTICOS

Los Sistemas Telemáticos son sistemas de aplicación en el ámbito de las telecomunicaciones que utilizan los sistemas informáticos con el fin de mejorar la calidad de los servicios prestados o proveer nuevos servicios; estos sistemas de software o servicios de telecomunicación son soportados en alto grado por componentes computacionales que efectúan el tratamiento y procesamiento de la información. El tratamiento de la información va ligado a las aplicaciones software que utilizando los servicios de esos componentes y que hoy en día pueden llegar a tener más peso que la infraestructura que las soporta, entre estas se encuentran, aplicaciones para sistemas distribuidos, manejo de servidores de red, sistemas de tiempo real, entre otros.

Una de las aplicaciones más representativas de los Sistemas Telemáticos son las redes de computadores, que surgen por la necesidad de conectar computadores personales para compartir y transmitir información. Dichas redes de computadores constan de una colección de computadores autónomos interconectados. Hoy en día el rápido avance que se ha tenido en el área de las telecomunicaciones y la necesidad que ha tenido la sociedad por tener información en cualquier momento y en cualquier lugar, ha hecho que los Sistemas Telemáticos se hayan popularizado notablemente, siendo su uso obligatorio para suplir las necesidades que conlleva esta nueva sociedad de la información.

El modelado en los Sistemas Telemáticos es de gran importancia, ya que la mayoría de éstos presentan mayor complejidad que por ejemplo los mecánicos, la

técnica del modelamiento se convierte en una pieza fundamental a la hora de diseñar e implementar estos sistemas. Las ventajas que ofrece el modelado de sistemas, hace que hoy en día se pueda implementar de una manera más segura cualquier tipo de sistema, sin importar su complejidad o tamaño. Los Sistemas Telemáticos como se ha dicho, presentan por lo general una gran complejidad, por lo tanto, a la hora de modelar uno de estos sistemas se pueden representar diferentes niveles de este sistema para simplificar el modelo total, y además estos modelos se pueden simular de tal forma que mientras más cercano esté el modelo a las condiciones reales, más precisa será su implementación.

2.2.2. MODELAMIENTO Y DISEÑO EN PTOLEMY II

El proyecto Ptolemy II desarrollado por el grupo de investigación del Departamento de Ingeniería Eléctrica y Electrónica de la Universidad de California en Berkeley, es un proyecto que estudia el diseño y modelado de Sistemas Heterogéneos y su énfasis es en sistemas empotrados, particularmente aquellos que mezclan tecnologías, incluyendo por ejemplo electrónica digital y analógica, hardware y software, y dispositivos electrónicos y mecánicos (incluyendo MEMS -Sistemas Mecánicos Microelectrónicos-). Una de las áreas de interés de Ptolemy está también en sistemas que son complejos, entendiéndose por esto, que ellos mezclan ampliamente diferentes operaciones, tales como procesamiento de señales, control de retroalimentación, e interfaces de usuario

El Modelamiento es el acto de representar un sistema o subsistema formalmente. Un modelo podría ser matemático, en cuyo caso puede ser visto como un grupo de aserciones acerca de las propiedades de un sistema tales como su funcionalidad o dimensiones físicas. Un modelo puede también ser constructivo, en cuyo caso define un procedimiento computacional que imita un grupo de propiedades de un sistema. Los modelos constructivos son generalmente usados para describir el comportamiento de un sistema en respuesta a estímulos provenientes fuera del sistema. Los modelos constructivos son también llamados modelos ejecutables.

El Diseño es el acto de definir detalladamente un sistema o subsistema. Usualmente esta envuelve la definición de uno o más modelos del sistema y el

refinamiento de los modelos hasta que la funcionalidad deseada es obtenida entre un grupo de restricciones.

El Diseño y Modelamiento están fuertemente relacionados. En algunas circunstancias, los modelos pueden ser inmodificables, esto quiere decir que ellos describen subsistemas, restricciones o comportamientos que son impuestos externamente en un diseño.

En Ptolemy el modelamiento y diseño de sistemas está basado en componentes, estos componentes se llaman actores y se encuentran en las diferentes librerías de Ptolemy II. Estos actores se comunican entre sí para formar el sistema que se desea modelar y/o diseñar, para luego obtener un modelo ejecutable del sistema.

Los modelos ejecutables algunas veces son llamados simulaciones, un termino apropiado cuando el modelo ejecutable es claramente distinto del sistema que modela. Sin embargo, en muchos sistemas electrónicos, un modelo que empieza como una simulación lleva a una implementación de software del sistema. La distinción entre el modelo y el sistema en sí empieza a ser menos clara en este caso. Esto es particularmente cierto para software empotrado.

El modelado en Ptolemy II cumple los objetivos del modelado, ya que con los modelos creados en Ptolemy se busca visualizar como es o se quiere que sea un sistema. Además se logra ver la estructura y el comportamiento de dicho sistema, lo cual es fundamental a la hora de empezar a realizar el diseño de un sistema.

2.2.2.1. MODELAMIENTO Y DISEÑO DE SISTEMAS TELEMÁTICOS EN PTOLEMY II

Debido a la gran importancia que han tomado los Sistemas Telemáticos, en especial las aplicaciones de redes y aplicaciones distribuidas, el proyecto Ptolemy ha centrado gran parte de sus esfuerzos en proporcionar los componentes necesarios para poder llevar el proceso de modelamiento, diseño y simulación de estos sistemas en su herramienta Ptolemy II.

Ya que los Sistemas Telemáticos son sistemas que manejan gran cantidad de datos, Ptolemy II permite dividir un sistema complejo en subsistemas más sencillos para lograr obtener un modelo de cada uno de estos de una forma más fácil, lo cual hace mucho más fácil el proceso de representación del sistema y cada uno de estos modelos pueden interactuar entre sí para llegar a tener el modelo del sistema total.

Un ejemplo de las aplicaciones telemáticas en las que esta trabajando el proyecto Ptolemy es la implementación de ambientes de computación distribuidos con la herramienta Ptolemy II. A continuación se presentan dos formas en las que se está trabajando para implementar estos ambientes con la herramienta:
Paso de mensajes a través de la red. Este da la posibilidad de que los modelos de Ptolemy II puedan obtener información de la red, y colocar información en la red.
Disponibilidad de los componentes de Ptolemy II a través de la red. La idea es que cualquier componente pueda ser colocado en la red. Esta alternativa se esta trabajando implementando el modelo de objetos de CORBA.

El trabajo en red de Ptolemy II es posible gracias a que este se encuentra implementado en el lenguaje Java, el cual por sus características es propicio para ambientes distribuidos.

Los dominios[1] DE (Eventos Discretos) y PN (Redes de Procesos) de Ptolemy II ofrecen los actores necesarios para llevar a cabo estas aplicaciones de redes, estos actores implementan funciones para el manejo de eventos que se presentan en tiempos discretos, así como el manejo de colas de eventos y simulación de servidores.

Los dominios de Ptolemy II ofrecen las facilidades para la realización de modelos de sistemas de tiempo real, los cuales pueden ser implementados bajo los dominios que trabajan con noción de tiempo como son el CT (Tiempo Continuo) y el CSP (Procesos Secuenciales de Comunicación). Con estos dominios se pueden modelar sistemas en los que se necesite representar tareas en tiempo real como por ejemplo vídeo o audio bajo demanda.

[1] Dominio: Es una implementación de un modelo de computación en Ptolemy II.

Los modelos CSP son útiles para aplicaciones donde la utilización compartida de recursos es un elemento clave en el sistema, como en el caso de las bases de datos cliente-servidor y la multiplexión o multitarea de recursos hardware. El dominio CSP, al igual que los otros dominios de Ptolemy II son explicados en el capítulo 4 del presente documento.

Aparte de las aplicaciones distribuidas Ptolemy II, también se tienen aplicaciones de red que mezclan simulaciones de eventos discretos y simulaciones de flujo de datos sincrónico, estas aplicaciones abarcan sistemas de manejo de recursos compartidos, protocolos de comunicación de red, redes de conmutación de paquetes, redes inalámbricas y sistemas multimedia.

2.3. PTOLEMY II

Ptolemy II es una herramienta software que hace parte de las tecnologías orientadas a objetos y que por su facilidad y flexibilidad se ha convertido en una de las herramientas más utilizadas para el desarrollo de sistemas de telecomunicaciones, aplicaciones de redes, sistemas optoelectrónicos y sistemas de radiocomunicaciones entre otros.

El proyecto Ptolemy surgió en la Universidad de California en Berkeley (USA), con su primera herramienta Ptolemy Clásico, un ambiente software de 3^{era} generación que se lanzo en 1990. Esta herramienta fue consecuencia de dos generaciones previas de ambientes de diseño, Blosim y Gabriel los cuales fueron enfocados hacia el procesamiento de señales digitales. Ambos ambientes utilizaban semántica de flujo de datos con una sintaxis de diagramas de bloques para la descripción de algoritmos. A diferencia de sus predecesores Ptolemy Clásico proporcionaba una infraestructura que se extendía a nuevos modelos de computación para la implementación de diferentes sistemas. La flexibilidad de Ptolemy Clásico permitió implementar una metodología de diseño bastante robusta, esto llevo a que el proyecto hiciera parte de ARPA (Agencia de Proyectos de Investigación Avanzada) y que otros grupos de investigación respaldaran el proyecto como La Fuerza Aérea de los Estados Unidos y diferentes compañías que trabajan en el campo de la Electrónica y las Telecomunicaciones.

A partir de 1998 y como consecuencia del auge de las aplicaciones soportadas en la web, el proyecto Ptolemy saco su nueva herramienta Ptolemy II, la cual esta implementada en el lenguaje de programación Java, a diferencia de su predecesora implementada en C++, lo cual permite que sea una plataforma independiente, que sea distribuida y posibilita su trabajo en la red.
Debido al gran número de personas que trabajan en el proyecto, la herramienta Ptolemy II se encuentra en permanente actualización.

2.3.1. CARACTERISTICAS DE PTOLEMY II

La estructura y conformación de Ptolemy II lo ubican dentro de las tecnologías orientadas a objetos. Ptolemy II está modelado con UML e implementado en el lenguaje Java. La arquitectura global consiste en un grupo de paquetes que proporcionan un soporte genérico para todos los modelos de computación[2] que implementa Ptolemy II y un grupo de paquetes que proporcionan un soporte más especializado para cada modelo de computación. Estos paquetes incluyen librerías matemáticas, algoritmos gráficos, e interfaces con capacidades multimedia como audio, vídeo, etc., los cuales son usados por cada uno de los dominios.

Las características que hacen de Ptolemy II una tecnología orientada a objetos son las siguientes:

MODULARIDAD
PTOLEMY II consiste de varios paquetes de software que pueden ser usados independientemente.

ARQUITECTURA DEL SOFTWARE
Ptolemy II ha sido diseñado con modelamiento de objetos.

SISTEMA TIPO POLIMÓRFICO
Ptolemy II tiene la capacidad del polimorfismo, que consiste en soportar métodos con un mismo nombre y actuar de manera diferente según sea su tipo.

[2] Modelo de Computación: Define las reglas que gobiernan la interacción, comunicación y el flujo de control de un conjunto de componentes.

INTERFAZ DE USUARIO

Ptolemy II en su versión 1.0.1 tiene una moderna interfaz de usuario llamada Vergil la cual proporciona un acercamiento más amigable del usuario a la herramienta.

Las capacidades que ofrece la herramienta Ptolemy II gracias a su enfoque orientado a objetos son las siguientes:

Mejor modularización debido al uso de paquetes. Ptolemy II está dividido en paquetes que pueden ser usados independientemente y distribuidos en la red. Esto rompe con la tradición en el diseño del software, donde las herramientas son usualmente empotradas en grandes sistemas integrados con partes interdependientes.

Completa separación de la sintaxis abstracta de la semántica. El diseño de Ptolemy II está estructurado como gráficas agrupadas. Ptolemy II define una clara y minuciosa sintaxis abstracta para las gráficas agrupadas, y separa en paquetes diferentes la infraestructura que soporta, como gráficas de mecanismos que adjuntan semántica (tales como flujo de datos, circuitos analógicos, maquinas de estados finito, etc.) a las gráficas.

Arquitectura de software basada en el modelamiento de objetos. En el diseño de Ptolemy II y versiones anteriores se ha visto el surgimiento del sofisticado modelamiento de objetos y conceptos de patrones de diseño. Todo esto ha derivado en un diseño más robusto, limpio y consistente al igual que se ha logrado simplificar el diseño del software.

Actores de dominios polimórfico. En Ptolemy clásico, las librerías de actores estaban separadas por dominios. A través de la noción de subdominios, los actores pueden operar en más de un dominio. En Ptolemy II, esta idea es tomada mucho más allá. Los actores con funcionalidad de polimorfismo pueden ser escritos para operar en un grupo mucho más grande de dominios. El mecanismo que utilizan para comunicarse con otros actores depende del dominio en el cual ellos son utilizados.

3. COMPOSICION DE PTOLEMY II

En este capítulo se describen los elementos funcionales de la herramienta, los actores de Ptolemy II, además de describir la arquitectura del software de la misma.

3.1. PTOLEMY II

Ptolemy II es una herramienta software desarrollada en la Universidad de California en Berkeley[3], la cual consiste en un conjunto de paquetes que soportan heterogéneamente, modelamiento y diseño concurrentes.

Ptolemy II es una infraestructura de aplicaciones orientadas a objetos que estudia modelamiento, simulación y diseño de sistemas empotrados, de tiempo real y concurrentes. El énfasis está en el ensamblaje de componentes concurrentes. El principio subyacente fundamental en el proyecto Ptolemy II es la utilización de modelos de computación bien definidos que gobiernan la interacción entre componentes.

En Ptolemy II se manejan ciertos conceptos que son propios de la herramienta pero que tienen correspondencia con los conceptos orientados a objetos y específicamente al lenguaje UML que es el utilizado por la herramienta.

- Paquetes: Un paquete es una colección de clases que encajan lógicamente, y que pueden interactuar entre ellas. En la herramienta los paquetes son elementos de UML. Los paquetes pueden contener subpaquetes que están adjuntos al paquete padre.

[3] Sitio web: http://ptolemy.berkely.edu

- Actores: Ptolemy II se refiere a diseño basado en componentes, estos componentes son los actores, en términos simples un actor es una entidad que procesa datos los cuales recibe a través de sus puertos o que crea y envía datos a otras entidades a través de sus puertos. Se hablará en detalle en la sección 3.3.

- Dominios: O modelos de computación, definen la semántica o preceptos que definen y rigen la interacción entre los componentes o actores y el software que soporta esta interacción. La aproximación a los dominios se encuentra en el capitulo 4 y una descripción más detallada en el Anexo 1.

3.2. PAQUETES

La arquitectura global consiste de un grupo de paquetes que proporcionan soporte genérico para todos los modelos de computación y un grupo de paquetes que proporcionan un soporte más especializado para modelos particulares de computación.

Entre estos paquetes se encuentran:

•kernel (gráficas agrupadas)
•actor (modelos ejecutables)
•data (señales, expresiones)
•schematic (API (Aplication Program Interface) para UIs (User Interfaz))
•graph (algoritmos gráficos)
•math (algoritmos matemáticos)
•plot (utilidades de despliegue gráfico)

Estos paquetes contienen clases que dan soporte a las diferentes funcionalidades de la herramienta. La figura 3.1 muestra los principales paquetes de Ptolemy II.

3.2.1. ESTRUCTURA DE LOS PAQUETES

La estructura de los paquetes es mostrada en la figura 3.1. El rol de cada paquete es explicado a continuación.

actor. Este paquete soporta entidades ejecutables que reciben y envían datos a través de puertos. Este incluye actores con tipo y actores sin tipo[4]. Para actores con tipo, implementa un sistema tipo[5] sofisticado que soporta polimorfismo. Incluye la clase base Director para clases de dominios específicos que controla la ejecución de un modelo. El concepto de sistema tipo se explica en la sección 3.5.

actor.gui. Este subpaquete es una librería de actores polimórficos con componentes de interfaz de usuario, más algunas clases básicas convenientes para applets y aplicaciones.

actor.lib. Este subpaquete es una librería de actores polimórficos, es decir, aquellos que operan en cualquier dominio.

actor.process. Este subpaquete proporciona infraestructura para dominios donde los actores son procesados (implementados).

actor.sched. Este subpaquete proporciona infraestructura para dominios donde los actores son planeados estáticamente por el director.

actor.util. Este subpaquete contiene utilidades que soportan directores en varios dominios.

data. Este paquete proporciona clases que encapsulan y manipulan datos que son transportados entre actores en los modelos de Ptolemy II.

[4] Actores con tipo y sin tipo: hace referencia a los actores implementan o no el sistema tipo.
[5] Sistema Tipo: Revisar concepto en la sección 3.5 o en el glosario.

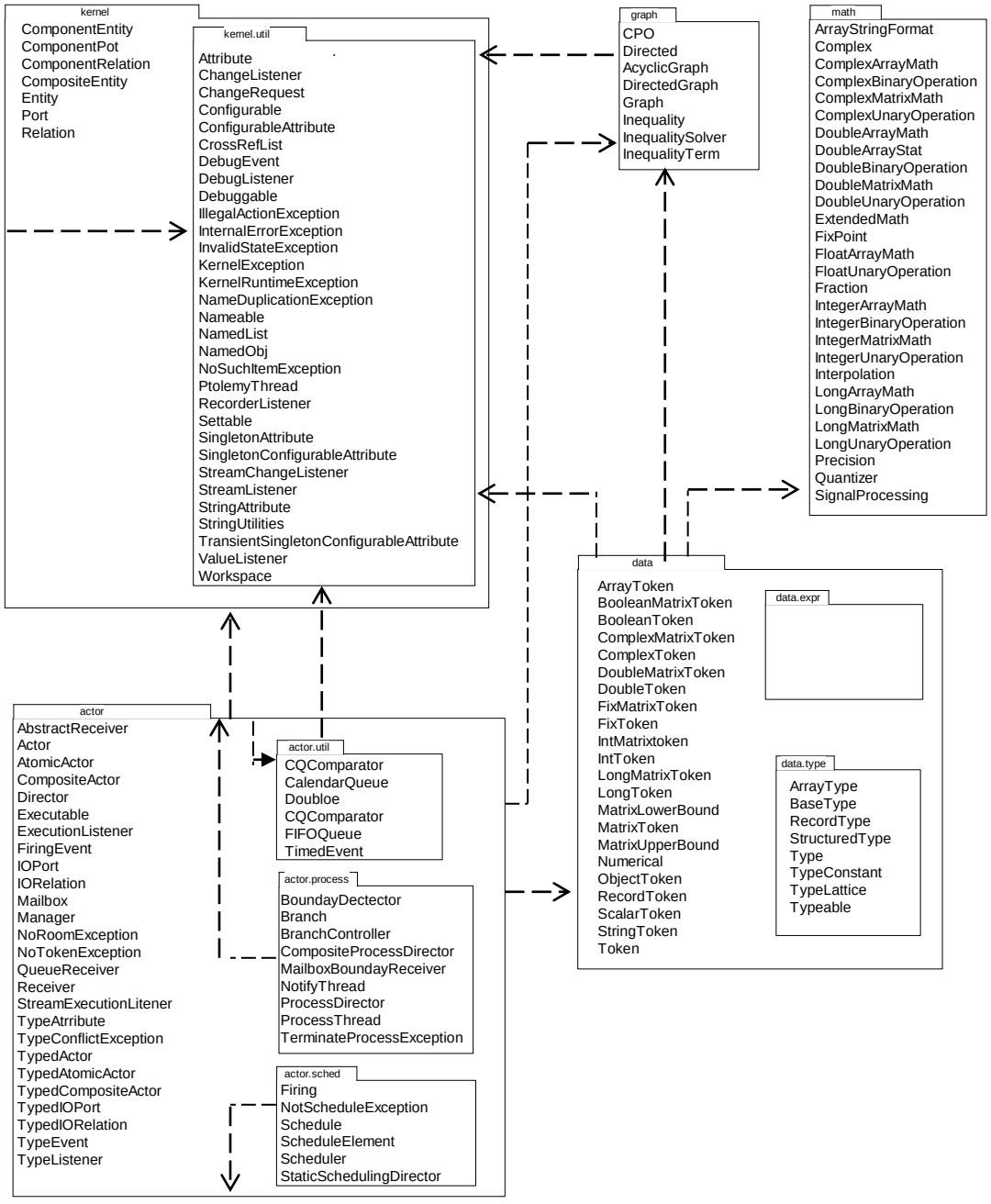


Figura 3.1. Principales paquetes que conforman Ptolemy II y sus subpaquetes.[6]

[6] Tomada de documento de Ptolemy. Pagina 1-12.

data.expr. Esta clase soporta un lenguaje de expresión extensible y un interprete para este lenguaje. Los parámetros pueden ser valores especificados por expresiones. Estas expresiones podrían referirse a otros parámetros. Las dependencias entre parámetros son manejadas transparentemente, como en una hoja de cálculo, donde la actualización del valor de uno resultará en la actualización de todos los que dependan de él.

domains. Este paquete contiene un subpaquete para cada dominio de Ptolemy II.

graph. Este paquete proporciona algoritmos para la manipulación y análisis de gráficas matemáticas. Las gráficas matemáticas son más simples que las gráficas agrupadas de Ptolemy II en las que no hay jerarquía.

gui. Este paquete contiene componentes de interfaz de usuario.

kernel. Este paquete proporciona la arquitectura del software para la sintaxis abstracta clave y gráficas agrupadas. Las clases en este paquete soportan entidades con puertos, y relaciones que conectan los puertos. El agrupamiento es donde una colección de entidades es encapsulada en una sola entidad compuesta, y un subgrupo de los puertos de las entidades internas son expuestos como puertos de la entidad agrupada.
kernel.event. Este paquete contiene clases e interfaces que soportan mutaciones controladas de gráficas agrupadas. Las mutaciones son modificaciones en la topología, y en general, se permite que ocurran durante la ejecución de un modelo.

kernel.util. Este subpaquete del paquete kernel proporciona una colección de clases que no dependen del paquete kernel. Estas clases son separadas en un subpaquete de manera que puedan ser utilizadas sin el kernel.

math. Este paquete encapsula funciones matemáticas y métodos para operar en matrices y vectores. También incluye una clase de números complejos y una clase que soporta fracciones.

media. Este paquete encapsula un grupo de clases que soportan procesamiento de audio e imágenes.

plot. Este paquete proporciona un despliegue bidimensional de señales.

schematic. Este paquete proporciona una interfaz de nivel superior para Ptolemy II. Una GUI puede usar las clases en este paquete para ganar acceso a los modelos de Ptolemy II.

3.3 ACTORES

Los actores son componentes con entradas y salidas que por lo menos conceptualmente operan concurrentemente con otros actores.
El énfasis de Ptolemy II está en el diseño basado en componentes; estos componentes, llamados actores, son agregados e interconectados entre si para construir un modelo. Algunos de estos actores son polimórficos lo que permite la reutilización de componentes y evita la duplicación de código. Algunos actores se diseñan para ser polimórficos en dominio, significando esto que pueden operar en varios dominios. Muchos actores también son polimórficos en datos, esto significa que ellos pueden operar en una gran variedad de tipos de datos de entrada o de señales.

Ptolemy II incluye librerías de actores polimórficos los cuales pueden ser usados en un amplio número de dominios, estas librerías se especifican en la sección 3.3.2.

Así mismo existen actores que sólo se usan en un dominio especifico, es decir que no presentan polimorfismo en dominio, estos actores se encuentran en ptolemy.domains.x.lib, donde x es el nombre del dominio.

3.3.1. PUERTOS

Los actores interactúan a través de puertos de entrada y de salida. Por convención, los puertos son miembros públicos de los actores. Ellos median entrada y salida.

3.3.1.1. CLASES DE PUERTOS

Un puerto puede ser un puerto único o multipuerto. Por defecto, este es puerto único y puede ser declarado como multipuerto modificando su estado en el código. Todos los Puertos tienen un ancho. Si el puerto no está conectado, el ancho es cero. Si el puerto es un puerto único, el ancho puede ser cero o uno. Si el puerto es un multipuerto, el ancho es mayor que uno. Un multipuerto media la comunicación sobre cualquier número de canales.

3.3.2. LIBRERÍA DE ACTORES

Ptolemy incluye tres librerías de actores. Dos librerías de actores polimórficos, ptolemy.actor.gui y ptolemy.actor.lib. y se encuentra una librería por cada dominio para aquellos actores de dominio especifico, por ejemplo se tienen librerías de actores para los dominios DE y CT.

Los actores polimórficos de estas librerías pueden ser usados en un amplio rango de dominios, donde el dominio proporciona el protocolo de comunicación entre actores. En general, escribir actores polimórficos en datos y dominios es considerablemente más difícil que escribir actores más especializados, ya que se debe seguir la semántica impuesta para los actores que se pretende, interactúen en cualquier dominio; esto referido a la configuración de los atributos (puertos, parámetros) los métodos, entro otros.

3.3.2.1. Actor.gui.

Todos los actores de esta librería implementan funciones de interfaz de usuario gráfica (GUI). Los actores de esta librería dan múltiples opciones de despliegue de la información procesada por los actores en forma gráfica.

3.3.2.2. Actor.lib.

El subpaquete actor.lib proporciona una clase básica para actores con puertos donde los puertos llevan datos clasificados en un tipo (encapsulados en objetos llamados tokens).

Ninguna de las clases, en este subpaquete tienen métodos, excepto aquellas heredadas de las clases básicas.
Muchos de los actores en este paquete son transformers[7]. Estos actores leen datos de entrada, modificándolos en alguna manera, y producen datos de salida.

Las interfaces TimedActor y SequenceActor juegan un papel muy importante en esta librería. Un actor que implementa SequenceActor declara que sus entradas son asumidas como secuencias de valores de datos distintos, y que las salidas que produce serán secuencias de valores de datos distintos. Así, por ejemplo, el actor Average computa un promedio de las señales de entrada. Por el contrario, el actor Sine no implementa SequenceActor, porque no le interesa si la entrada es una secuencia. En particular, el orden en el cual se presentan las entradas es irrelevante, y es irrelevante si una entrada en particular se presenta más de una vez. La implementación de la interfaz TimedActor declara que el tiempo actual en un modelo de ejecución afecta el comportamiento del actor.
El Anexo 3 contiene la lista completa de los actores presentes en las librerías de la herramienta y su funcionalidad.

3.4. ARQUITECTURA DEL SOFTWARE DE PTOLEMY II

A continuación se describirá la arquitectura del software de Ptolemy II, se hará una descripción de los paquetes que conforman la herramienta.

[7] Los actores transformers son aquellos que transforman un flujo de datos de entrada para generar un flujo de datos de salida.

3.4.1. PAQUETE KERNEL

En este numeral se define como está constituido el Kernel de la herramienta, y se hace referencia a las principales clases que lo conforman.

3.4.1.1 INTRODUCCIÓN

Típicamente, un Kernel (o cualquier centro comparable de un sistema operativo) incluye un manipulador de interrupciones que maneja todas las demandas o funcionamientos de I/O que compiten por los servicios que presta el Kernel, un scheduler que determina qué programas comparten el tiempo de procesamiento y en qué orden, y un supervisor que da uso de los recursos del computador a cada proceso cuando es programado. Un kernel también puede incluir un manejador del espacio de dirección del sistema operativo en memoria o almacenamiento, comparte éste entre todos los componentes y otros usuarios del Kernel. Los servicios del Kernel son pedidos por otras partes del sistema operativo o por otras aplicaciones a través de un conjunto específico de interfaces de programa conocido como llamadas del sistema. Algunos Kernels se han desarrollado independientemente para el uso en cualquier sistema operativo que necesite usarlo.

3.4.1.2. CONCEPTO DE KERNEL EN PTOLEMY II

El paquete Kernel está dentro de la estructura de paquetes de Ptolemy II. Este paquete soporta gráficas jerárquicas agrupadas, que son colecciones de entidades y las relaciones entre esas entidades. Se ampliará el concepto de gráficas agrupadas en la sección 3.4.1.3.
El Kernel de Ptolemy II define un pequeño conjunto de clases Java que implementan una estructura de datos que soportan una forma general de gráficas agrupadas, además de métodos para accesar y manipular dichas gráficas.

3.4.1.2.1. RESUMEN DEL PAQUETE KERNEL Y EL SUBPAQUETE KERNEL.UTIL

El paquete Kernel provee la arquitectura del software para el modelo de datos de Ptolemy II, o sintaxis abstracta. Esta sintaxis abstracta tiene la estructura de gráficas agrupadas. Las clases en este paquete soportan entidades con puertos, y relaciones que conectan los puertos. El agrupamiento se da cuando una colección de entidades es encapsulada en una simple y única entidad compuesta (composite entity), y tiene unos puertos expuestos como propios.

El subpaquete kernel.util del paquete kernel provee una colección de clases de utilidad que no dependen del paquete principal. Esta colección está separada en un subpaquete que hace posible que estas clases puedan ser usadas sin el kernel. Las utilidades incluyen una colección de excepciones, clases que soportan objetos nombrados con atributos, y listas de objetos nombrados. La figura 3.2 muestra las clases importantes en el paquete Kernel.

En las siguientes secciones se hablará de la estructura del kernel de Ptolemy II, además se explorarán las facilidades y el soporte que ofrece el kernel a las diferentes tareas de la herramienta.

3.4.1.3. GRÁFICAS AGRUPADAS

Las gráficas agrupadas representan colecciones de entidades y las relaciones existentes entre esas entidades. Estas gráficas proveen una sintaxis abstracta (sección 3.4.1.4) de los diagramas de transición de estado, diagramas de bloque, etc. Esto quiere decir que no se impone alguna semántica sobre los modelos.

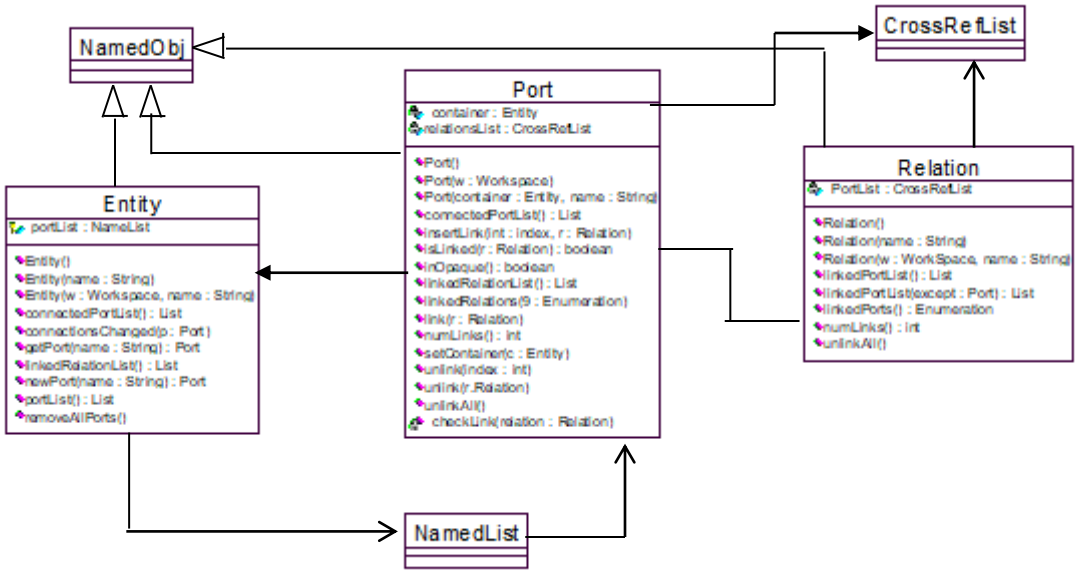


Figura 3.2. Principales clases en el paquete kernel que soportan topologías básicas. Tomada del documento de Ptolemy II página 7-3

3.4.1.4. SINTAXIS ABSTRACTA

Las gráficas agrupadas proveen una sintaxis abstracta. Estas gráficas a su vez conforman un modelo, por lo tanto el uso de la sintaxis abstracta en Ptolemy permite la manipulación de modelos ejecutables y no se limita a modelos basados en paquetes o dominios específicos sino que permite el trabajo con cualquier modelo de Ptolemy. Una sintaxis abstracta es una organización de datos conceptual.

Una gráfica agrupada particular se denomina topología y es una colección de entidades y relaciones. En la figura 3.3 se muestra el ejemplo de una topología donde las entidades son descritas como cajas redondeadas y las relaciones como diamantes. Las entidades tienen puertos, mostrados como círculos, y relaciones que conectan los puertos.

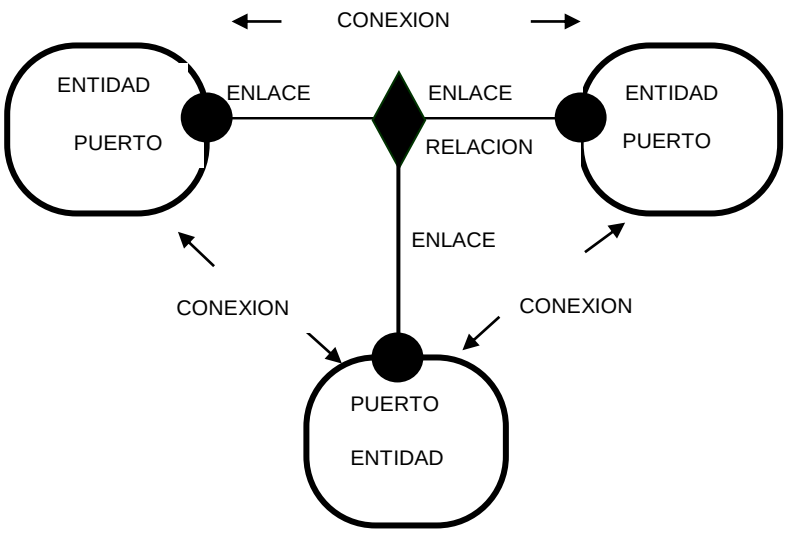


Figura 3.3. Topología Básica. Notación visual y terminología. Tomada del documento de Ptolemy II Pagina 7-2

3.4.1.5. CLASES SOPORTE

Como se mencionó el paquete Kernel tiene un subpaquete llamado kernel.util que provee algunas clases soporte fundamentales. Estas clases definen las nociones básicas de Ptolemy II de contención, nombramiento y parametrización; y además proveen soporte genérico para estructuras de datos relevantes. A continuación se mencionarán algunas de las clases soporte.

3.4.1.5.1. CONTAINERS

Aunque estas clases no proveen soporte para construir gráficas agrupadas, si proveen un soporte rudimentario para asociaciones container. Una instancia de estas clases puede tener como máximo un contenedor. Un contenedor es un objeto que lógicamente contiene a otro y es visto como el propietario del objeto.

3.4.1.5.2. NAME Y FULL NAME

La interfaz Nameable soporta jerarquía en el nombramiento de manera que los objetos nombrados individualmente en una jerarquía pueden ser identificados. Por convención, el nombre completo de un objeto es una concatenación del nombre completo de su contenedor, si hay uno, un campo (“.”), y el nombre del objeto. El nombre completo es usado frecuentemente para reportar errores. Un objeto de nivel superior siempre tiene un campo como el primer carácter de su nombre completo.

3.4.1.5.3. WORKSPACE

Workspace es una clase concreta que implementa la interfase Nameable. Todos los objetos en una topología están asociados con un espacio de trabajo y casi todas las operaciones que incluyen múltiples objetos están soportadas únicamente por objetos en el mismo espacio de trabajo.

3.4.1.5.4. ATTRIBUTE

En casi todas las aplicaciones de Ptolemy II, las entidades, los puertos y las relaciones necesitan ser parametrizados.

Los atributos pueden ser adicionados o removidos utilizando métodos y pasando a referenciarlos al contenedor. A la vez se proveen métodos para tomar un nombre de atributo como un argumento y retornar al atributo, y un método que retorna una enumeración de todos los atributos del objeto.

3.4.1.5.5. LISTA DE CLASES

Existen dos listas de clases que son utilizadas extensamente en Ptolemy II. La clase NamedList implementa una lista ordenada de objetos con la interfase Nameable. Esta lista puede ser usada para mantener un listado de atributos o para mantener la lista de puertos contenidos por una entidad.

La clase CrossRefList media enlaces bidireccionales entre objetos que contienen CrossRefLists, que pueden ser puertos y relaciones. Esta provee un mecanismo simple y eficiente para construir una trama de objetos, donde cada objeto mantiene una lista de los objetos a los que esta enlazado. Esta lista es una instancia de CrossRefList. La clase asegura consistencia; es decir, si un objeto en la red se enlaza a otro, entonces este es enlazado al primero.

3.4.1.6 CONCEPTOS RELACIONADOS CON LAS GRÁFICAS AGRUPADAS

3.4.1.6.1. ABSTRACCIÓN

Las entidades compuestas (CompositeEntity) no son atómicas, estas pueden contener una gráfica (entidades y relaciones). Por defecto, una CompositeEntity es transparente, esto conceptualmente significa que su contenido es visible desde afuera. La jerarquía puede ser ignorada por la operación de algoritmos sobre la topología. Algunas subclases de CompositeEntity son opacas. Esto fuerza a los algoritmos a respetar la jerarquía, ocultando con eficacia el contenido de un Composite y haciéndolo aparecer indistinguible de las entidades atómicas.

Un ComponentPort contenido por un CompositeEntity tiene enlaces tanto por dentro como por fuera. Mantiene dos listas de enlaces, las de relaciones internas y las de relaciones exteriores. Este puerto sirve para exponer puertos en las entidades contenidas como puertos del Composite. Los puertos dentro de una entidad están ocultos por defecto, y podrían ser explícitamente expuestos para ser visibles desde afuera de la entidad. La entidad compuesta con puertos provee una abstracción del contenido del Composite.

El mecanismo de puerto transparente es ilustrado por el ejemplo de la figura 3.4 Algunos de los puertos de la figura 3.4 no están rellenos, estos puertos son llamados transparentes. Los puertos transparentes (P3 y P4) están enlazados por las relaciones (R1 y R2) bajo su contenedor (E1) en la jerarquía. Ellos también pueden estar enlazados por relaciones en el mismo nivel (R3 y R4).

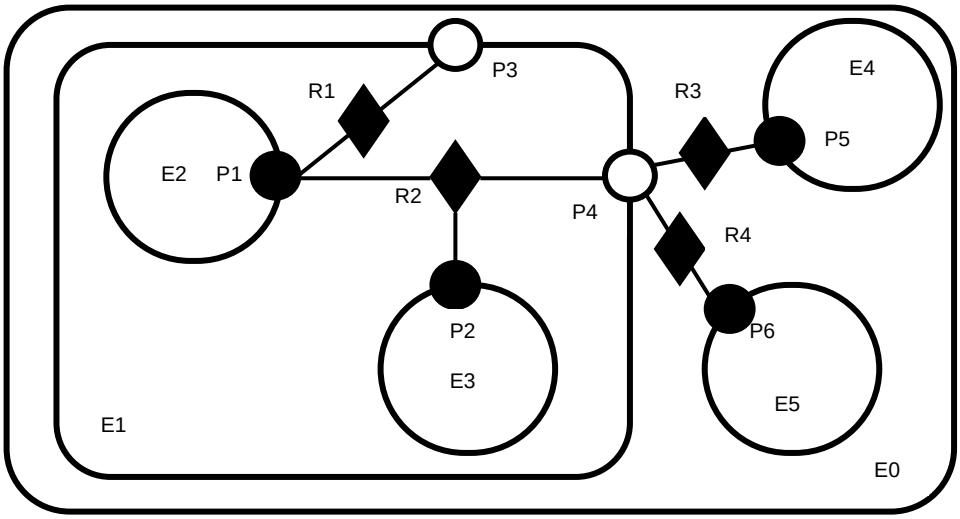


Figura 3.4. Gráfica que muestra puertos, enlaces y relaciones. Tomada del documento de Ptolemy página 7-9

ComponentPort, ComponentRelation y CompositeEntity tienen un conjunto de métodos con el prefijo “deep”. Estos métodos omiten la jerarquía atravesándola. Así, por ejemplo, los puertos que están “deeply” conectados al puerto P1 en la figura 3.4 son P2, P5 y P6.

3.4.1.6.2. ENTIDADES TUNNELING

Una entidad tunneling es la que contiene una relación con enlaces internos de más de un puerto. Este puede contener también más enlaces estándar, pero el término “tunneling” sugiere que por lo menos algunas transversales profundas se verán a través de la entidad.
Para revisar estas entidades se recomienda ver el documento de Ptolemy II en la seccion 7.4.3 que da una descripción detallada y con ejemplos didácticos extensos y comprensibles.

3.4.1.6.3. CLONACION

Las clases kernel son todas capaces de ser clonadas con algunas restricciones. La clonación significa que se puede crear un objeto idéntico pero totalmente independiente a partir de otro objeto. Así, si el objeto clonado contiene otros

objetos, entonces estos objetos son también clonados. Si, dichos objetos están enlazados, entonces los enlaces son reproducidos en los nuevos objetos. La clonación es una clase especial de Java, por tanto, cualquier clase que desee producir clones debe implementar la interfaz clonable

3.4.1.7 ENTIDADES COMPUESTAS OPACAS

Uno de los mayores principios del proyecto Ptolemy es modelar sistemas heterogéneos por medio del uso de heterogeneidad jerárquica. El ocultamiento de información es una parte central de esto. En particular, los puertos y entidades transparentes comprometen la información oculta exponiendo la topología interna de una entidad. En algunas circunstancias esto es inapropiado, por ejemplo, cuando una entidad internamente opera bajo un modelo de computación diferente del de su entorno. La entidad debería ser opaca en ese caso.

Una entidad puede ser compuesta y opaca al mismo tiempo. Los puertos son definidos como opacos si la entidad que los contiene es opaca. El paquete de actores hace un uso extenso de estas entidades al soportar modelamiento mixto.

3.4.1.8. MUTACIONES

Con frecuencia es necesario restringir los momentos cuando pueden ser hechos cambios en una topología. Por ejemplo, una aplicación que usa el paquete actor para ejecutar un modelo definido por una topología puede requerir que la topología permanezca fija durante segmentos de la ejecución. Durante estos segmentos, el espacio de trabajo puede ser de lectura únicamente. El subpaquete kernel.util del paquete kernel, provee soporte para controlar cuidadosamente las mutaciones que pueden ocurrir durante la ejecución de un modelo.

3.4.1.9. EXCEPCIONES

Ptolemy II incluye un conjunto de clases excepción que proveen un mecanismo uniforme para reportar errores que toman ventaja de la identificación de los objetos nombrados por el nombre completo.

3.4.1.9.1. CLASE BASE

KernelException. No se usa directamente. Provee una funcionalidad común para las excepciones Kernel. En particular, provee métodos que toman cero, uno o dos objetos Nameable más un mensaje de detalle opcional (una cadena).

3.4.1.9.2. EXCEPCIONES MENOS SEVERAS

Estas excepciones generalmente indican una operación que fracasa. Esto puede resultar en una topología que no es la esperada, ya que las modificaciones sobre la topología no tuvieron éxito.

IllegalActionException. Arrojada en un intento de desarrollar una acción que está denegada. Por ejemplo, la acción podría resultar en una estructura de datos inconsistente o contradictoria si fuese permitido completarla. Es decir, permitir fijar el contenedor de un objeto a otro objeto que no puede contenerlo porque es de la clase incorrecta.

NameDuplicationException. Arrojada en un intento de adicionar un objeto nombrado a una colección que requiere nombres únicos.
NoSuchItemException. Arrojada sobre el acceso a un ítem que no existe. Por ejemplo que se intenta remover un puerto por el nombre y dicho puerto no existe.

3.4.1.9.3. EXCEPCIONES MÁS SEVERAS

Las siguientes excepciones nunca deberían ser activadas. Si esto sucede, indica que hay una seria inconsistencia en la topología y/o virus en el código. Estas son

excepciones en el tiempo de ejecución, por lo tanto no se necesita declararlas explícitamente para ser arrojadas.

InvalidStateException. Algunos objetos o conjuntos de objetos tienen un estado que en teoría no es permitido. Por ejemplo un NamedObj tiene un nombre nulo. O una topología tiene inconsistencias o información contradictoria, por ejemplo, una entidad contiene un puerto que tiene una entidad diferente a su contenedor. Esta excepción es derivada de la RuntimeException de Java.

InternalErrorException. Un error no esperado o un estado inconsistente ha sido encontrado. Esta excepción es derivada de la RuntimeException de Java.

3.4.2. PAQUETE DE DATOS

El paquete de datos provee encapsulamiento de datos, polimorfismo, manipulación de parámetros, un lenguaje de expresión, y un sistema tipo.

3.4.2.1. ENCAPSULAMIENTO DE DATOS

La clase Token[8] y sus clases derivadas encapsulan los datos de la aplicación. Los datos encapsulados pueden ser transportados por medio del paso de mensajes entre los objetos de Ptolemy II. Alternativamente, pueden ser utilizados para parametrizar los objetos de Ptolemy II. Encapsulando los datos de esta manera se proporciona una interfase estándar para que puedan manipularse tales datos uniformemente sin tener en cuenta su estructura detallada. Es decir el usuario no se preocupa por que al entrar una cadena (string) o un número entero, o un número real, la herramienta no pueda interpretar la estructura de esos datos.

3.4.2.2. POLIMORFISMO

Operadores aritméticos polimórficos.

[8] La clase Token permite el polimorfismo de datos. Define una interfaz polimórfica que incluye operaciones aritméticas básicas.

Uno de los objetivos del paquete de datos es soportar las operaciones polimórficas entre señales. Para esto, la clase básica Token define los métodos para cargar las operaciones básicas primitivas. Las clases derivadas cargan estos métodos al proporcionar la operación especifica de clase dónde es apropiado.
Por ejemplo, dentro de los actores que tiene la herramienta, el actor AddSubtract toma cualquier tipo de datos de la entrada que solo son identificados por su presentación (entre comillas, corchetes, etc) y el usuario no debe introducir ningún otro dato adicional, a menos que la herramienta lo solicite.

3.4.2.2.1 Conversión tipo lossless.

Para las operaciones aritméticas, si dos señales son operadas y son de tipos diferentes, la conversión de tipo es necesaria, sólo las conversiones que no pierdan información son implícitamente llevadas a cabo.
Las conversiones de tipo de datos están regidas por una jerarquía. Esta jerarquía está relacionada a la jerarquía de herencia de clases de señales en que una subclase es siempre menor que su superclase.

3.4.2.3. TIPOS PARÁMETROS EN PTOLEMY II

Ptolemy II no tiene una gran cantidad de clases de parámetros de tipo específico. En cambio, un parámetro tiene un tipo que refleja la señal que contiene. Se puede restringir el tipo permitido de un parámetro o variable.

El tipo de la variable puede especificarse de varias maneras, todas ellas exigen que el tipo sea consistente con las restricciones especificadas (o se arrojará una excepción):

- Puede ser puesto directamente por una llamada al setTypeEquals (). Si esta llamada ocurre después de que la variable tiene un valor, entonces el tipo especificado debe ser compatible con el valor. De otra manera, una excepción se arrojará.
- Poniendo el valor de la variable a una señal no-nula restringe el tipo Variable para no ser menos del tipo de la señal.

- El tipo también es restringido cuando una expresión se evalúa. El tipo Variable no debe ser menor que el tipo de la señal al que la expresión evalúa.
- Si la variable no tiene un valor todavía, entonces el tipo de una variable puede ser determinado por resolución de tipo.

Sujeto a restricciones especificadas, el tipo de una variable puede cambiarse en cualquier momento. Algunas de las restricciones del tipo, sin embargo, no se verifican hasta que la resolución del tipo se hace. La resolución del tipo normalmente es hecha por el Manager que ejecuta un modelo.

3.4.2.4. EXPRESIONES

Ptolemy II incluye un lenguaje de expresión simple pero extensivo. Este lenguaje permite operaciones en señales para ser especificadas en un modo de escritura, sin requerir compilaciones de código Java. El lenguaje de expresión puede ser usado, por ejemplo, para definir parámetros en términos de otros parámetros.

3.4.2.4.1. El lenguaje de expresión de Ptolemy II

El lenguaje de expresión de Ptolemy II utiliza un operador sobrecargado, a diferencia de Java. Este lenguaje de expresión es extensible.

A continuación se presenta una lista con los tipos de datos que maneja el lenguaje de expresión de Ptolemy II, así como los operadores, funciones y métodos que actúan sobre estos.

a)Tipos
Los tipos corrientemente soportados en el lenguaje son booleanos, complejos, decimal, entero, longitud, cadenas y matrices. Note que no hay ningún flotante o byte. En su lugar se usan decimales o enteros. El lenguaje no maneja matrices ni vectores todavía. El lenguaje de expresión soporta el mismo tipo de conversión lossless proporcionado por las clases Token.

b) Operadores aritméticos

Los operadores aritméticos son +,-,*,/ y %. Estos operadores junto con == pueden operar sobre diferentes tipos de datos.

c) Manipulación de bit

Los operadores bitwise son &, |, ^, y ä. Ellos operan en enteros.

d) Operadores relacionales

Los operadores de relación son <,<=, >, >=, == y !=. Ellos retornan booleanos.

e) Operadores lógicos

Los operadores lógicos son &&, ||, ! y |.

f) Comentarios

Cualquiera dentro de /*...*/ es ignorado, como el resto de la línea siguiente //. (las expresiones pueden estar divididas en múltiples líneas).

g) Variables

Las expresiones pueden contener referencias por nombre a parámetros dentro del alcance de la expresión.

h) Constantes

Si un identificador es encontrado en una expresión que no hace juego con un parámetro en el alcance, entonces podría ser una constante que ha sido registrada como parte del lenguaje, de lo contrario debe hacerse una revisión de las constantes declaradas y hacer las correcciones a las que haya lugar.

i) Funciones

El lenguaje incluye una colección de funciones extensible, tales como seno, coseno, etc.

j)Métodos

Cada elemento y sub-expresión en una expresión representan una instancia de Token (o una clase derivada de Token). El lenguaje de expresión soporta invocaciones de cualquier método de una señal dada, con tal de que los argumentos del método sean del tipo Token y el tipo que retorna sea Token (o una

clase derivada de Token). Los métodos a diferencia de las funciones deben tomar argumentos que son del tipo Token.

3.4.3. PAQUETES DE ACTORES

El paquete de actores proporciona un soporte básico para las entidades ejecutables o actores, además de proveer la semántica para la ejecución y mecanismos de comunicación entre estos actores.

Las entidades ejecutables se ejecutan concurrentemente (en la mayoría de sus usos); el paquete de actores proporciona la infraestructura para esa ejecución concurrente.

Este paquete proporciona la infraestructura para llevar a cabo las dos funciones clave de soporte de los actores, las cuales son el paso de mensajes y la ejecución.

El paso de mensajes es la función utilizada para la comunicación entre actores. Los mensajes se encapsulan en señales (tokens) y se envían a través de los puertos de los actores.

El paquete de actores contiene las clases necesarias para el transporte de mensajes, entre las cuales se encuentra la clase IOPort, esta es la encargada directamente del transporte de mensajes.

El paquete de actores soporta dos tipos de receptores, los cuales son el Mailbox (Buzón de mensajes) y el QueueReceiver (cola receptora), debido a que cada modelo de computación proporciona un protocolo de comunicación diferente y por lo tanto hacen uso de diferentes receptores para realizar el paso de mensajes.
La transferencia de datos entre los actores se hace a través de los puertos de estos, utilizando los diferentes métodos para capturar y enviar datos que soportan estos puertos. Los puertos pueden ser sencillos, se comunican por un solo canal, o pueden ser multipuertos, se comunican por varios canales según las relaciones entre los actores.

El receptor usado por un puerto de entrada, determina el protocolo de comunicación usado, y este está ligado directamente con el dominio o modelo de computación.

A continuación se mencionaran los tipos de receptores que están habilitados en el paquete de actores

3.4.3.1. RECEPTORES

Como se mencionó el protocolo de comunicación entre actores está determinado por el tipo de receptor usado y que a la vez depende del domino escogido, a continuación se mencionan los receptores más importantes y útiles.

3.4.3.1.1. Comunicación Mailbox

El Mailbox es un receptor que tiene capacidad para una sola señal, y cuando tiene una señal o tiene espacio para poner una señal avisa mediante llamadas a los respectivos métodos.

3.4.3.1.2. Paso de mensajes asincrónico

El receptor utilizado por este tipo de comunicación es el QueueReceiver, el cual implementa una cola FIFO, que es apropiada para todos los modelos que utilizan flujos de datos. Este tipo de comunicación es usado en el modelo de computación redes de procesos (PN).

3.4.3.1.3. Comunicaciones Rendezvous

También llamada comunicación sincrónica, este tipo de comunicación requiere que tanto el emisor como el receptor estén simultáneamente listos para la

transferencia de datos. Este estilo de comunicación es implementado en el modelo de computación procesos secuenciales de comunicaciones (CSP).

3.4.3.1.4. Comunicación de eventos discretos

En el modelo de eventos discretos, las señales que son transferidas entre actores tienen una huella de tiempo, la cual especifica el orden en el cual los receptores deben procesarlas. Este tipo de comunicación se implementa con colas FIFO.

3.4.3.2. EJECUCIÓN

El paquete de actores contiene la interfaz Executable, la cual define como se invoca un actor y todo su proceso de ejecución. Hay dos clases de actores las cuales se encuentran en el paquete de actores y son el AtomicActor, que consiste en un actor simple, y el CompositeActor que es una agregación de actores.

La interfaz Executable define los siete métodos que se deben seguir para la invocación, ejecución y terminación de un actor, estos métodos son:

- initialize(): Es el método utilizado para invocar la ejecución de un actor.
- prefire(), fire() y postfire(), son los métodos encargados del proceso de activación de un actor.
- stoptfire() es invocado para pedir la suspensión de una activación de un actor.
- wrapup() es el responsable de limpiar todo después de que la ejecución de un actor ha sido completada, como por ejemplo vaciar los buffers y eliminar hilos activos.
- terminate(), este método podría ser llamado en cualquier momento de la ejecución para detenerla enseguida. Este método es usado como último recurso para interrumpir una ejecución.

Ptolemy II presenta dos clases llamadas Director y Manager, estas clases son las encargadas de gobernar las ejecuciones de las entidades ejecutables (actores) y de los modelos respectivamente.

La clase Director es la encargada de gobernar la ejecución de una entidad ejecutable, el director proporciona por defecto una implementación de una ejecución, y aunque algunos dominios no utilizan esta implementación de una manera exacta, todos los dominios deben seguir su secuencia para así asegurar la interoperabilidad.

La clase Manager es la encargada de controlar la ejecución global de un modelo y proporciona los métodos necesarios para que un modelo se ejecute.

3.4.4. PAQUETES DE GRÁFICAS

Ptolemy II proporciona un paquete de gráficos el cual es el encargado de entregar una extensa infraestructura para la creación y manipulación de gráficos.

Este paquete soporta un gran número de algoritmos para operar en gráficos matemáticos, los cuales son útiles en muchas de las funciones de Ptolemy II, como en el soporte de la planeación, la resolución del sistema tipo, y otras operaciones.

El paquete de gráficos contiene tres clases claves que soportan la construcción de los gráficos utilizados por Ptolemy II, y proporcionan algoritmos para manipular estos gráficos, estas clases son Graph, DirectedGraph y DirectedAcyclicGraph.

La clase Graph modela gráficos no dirigidos, la clase DirectedGraph la cual es derivada de la clase Graph, adiciona un borde dirigido al gráfico proporcionando gráficos dirigidos y la clase DirectedAcyclicGraph proporciona gráficos dirigidos sin ciclos, debido a que hay operaciones en las que por razones de desempeño los ciclos no son recomendados.

3.4.5. PAQUETE PLOT

El paquete Plot proporciona clases, applets, y aplicaciones para el despliegue gráfico bidimensional de datos.

El despliegue de datos puede ser especificado en cualquiera de estos dos formatos:

- PlotML es una extensión XML para despliegue de datos. Su sintaxis es similar a la de HTML.
- Se provee también una sintaxis textual simple para despliegue de datos, aunque a largo plazo no es probable que la herramienta la mantenga, sin embargo es adecuada para un despliegue simple de datos.

Entre las funcionalidades de este paquete se encuentran las de zoom y relleno. Cuando se desea ver en detalle alguna sección en particular de la gráfica, ésta se pude seleccionar mediante un cuadro arrastrando el puntero del mouse alrededor de la misma.

También se puede imprimir y exportar las gráficas a un procesador de texto, por ejemplo.

Las principales clases del paquete plot son:

- PlotBox: Un panel que dibuja una caja con ejes a lo largo de los bordes, comillas a lo largo de los ejes, nombres de los ejes, un título, y una leyenda.
- Plot: Una extensión de PlotBox que soporta una serie de plots de dos dimensiones de conjuntos de datos, incluyendo plots x-y, plots dispersos, y barra de gráficos.
- PlotLive: Una extensión de Plot diseñado para correr continuamente en su propia hilo de ejecución, actualizándose continuamente sobre plot en pantalla.
- PlotApplet: Un instrumento que contiene una instancia singular de Plot y que puede leer los datos a ser ploteados desde una URL.

- PlotLiveApplet: Una extensión de PlotApplet que contiene una instancia de PlotLive en lugar de Plot. Este es usado para instrumentos con plots animados que son actualizados continuamente.
- PlotFrame: Una ventana que contiene un plot singular y un menú de barra con comandos para abrir y desplegar nuevos archivos de datos.
- PlotApplication: Una extensión de PlotFrame que es una aplicación (un programa Java autosuficiente).

3.4.5.1. LIMITACIONES

El paquete plot tiene un gran número de limitaciones entre las que se encuentran:

- Se requiere un formato de archivo binario que incluya información del formato plot.
- Si su acercamiento es muy grande, el plot llega a ser irrealizable. En particular, si la extensión total del plot es mayor que 2^{32} veces la extensión del área visible, Pueden resultar errores de cuantificación en el despliegue de puntos o líneas. Notar que 2^{32} está sobre 4 billones.
- Actualmente las gráficas no pueden ser copiadas por medio de portapapeles.
- No hay un mecanismo para personalizar los colores usados en el plot.

3.5. EL SISTEMA TIPO

La estructura de computación provista por las clases básicas de actor no tienen una asignación de tipo estático, es decir, las clases no especifican los tipos de señales que pueden pasar a través de ellos. Dando un ejemplo concreto, la clase IOPort no especifica la clase de señales que manejarán los puertos y qué pasará a través de ellos. Esto puede cambiarse dando a cada instancia de IOPort un tipo. Una de las razones para hacer esto es incrementar el nivel de seguridad, lo cual significa reducir el número de errores no detectados.

En un ambiente de computación, pueden ocurrir dos tipos de errores de ejecución, los errores detectados y los errores no detectados. Los errores detectados provocan que la ejecución se detenga inmediatamente, pero los errores no detectados pueden pasar inadvertidos y más adelante causar comportamientos arbitrarios o no deseados. En Ptolemy II, el lenguaje Java fundamental es bastante seguro, así raramente hay errores, y si los hay, no causan comportamientos arbitrarios.

En lenguajes sin asignación estática de tipo, tales como Lisp y el lenguaje escrito Tcl, la seguridad se logra mediante un chequeo completo en el tiempo de ejecución. Si se imitara esta aproximación en Ptolemy II, se requerirían actores para chequear el tipo de las señales recibidas antes de usarlas, imponiendo una mayor atención sobre el chequeo del tipo a los desarrolladores de actores, desviando el esfuerzo del desarrollo de los mismos.

Para manejar este y otros problemas, se ha agregado asignación estática de tipo a Ptolemy II.

En Ptolemy II, la ejecución de un modelo no incluye compilación. No obstante, un chequeo de tipo estático puede detectar problemas antes de activar cualquiera de los actores. Por ejemplo, si un actor Source declara que su puerto de salida es String, significa que enviará StringTokens de salida en la activación, el chequeador de tipo estático identificará algún conflicto de tipo en la topología en la que este se encuentre.

En el paquete de datos, se define una jerarquía de conversión tipo de datos, llamada de tipo lattice. En esta jerarquía, la conversión de tipo más bajo a un tipo más alto es soportada por las clases Token. El principio de conversión jerárquica también se aplica para transferir datos. Esto significa que a través de cada conexión desde un puerto de salida a una entrada, el tipo de la salida podría ser igual o menor que el tipo de la entrada.

Cuando un sistema corre, el tipo de una señal enviada desde un puerto de salida puede no ser del mismo tipo del puerto de entrada al que es enviada. Si esto sucede, la señal podría ser convertida al tipo del puerto de entrada antes de ser usada por el actor receptor. Esta clase de conversión de tipo en tiempo de

ejecución es transparente al sistema Ptolemy II. Así los actores pueden lanzar seguramente las señales recibidas al tipo del puerto de entrada. Esto hace que el desarrollo del actor sea más fácil.

En Ptolemy II, la asignación de tipo aplica algunas restricciones sobre la interacción de actores. Particularmente, los actores pueden no estar interconectados arbitrariamente si la regla de compatibilidad de tipo es violada. La conversión de tipo en el tiempo de ejecución provista por el sistema permite que puertos de diferentes tipos sean conectados (bajo la regla de compatibilidad de tipo), lo cual parcialmente disminuye la restricción causada por la asignación estático.

4. DOMINIOS DE PTOLEMY II.

En este capítulo se especifican los modelos de computación los cuales describen la interacción entre los componentes de Ptolemy II, serán descritos brevemente en el presente documento y se explicarán a fondo en los anexos.

Los dominios en Ptolemy II implementan modelos de computación, la mayoría de estos modelos proporcionan la estructura necesaria para el diseño basado en componentes, donde la estructura proporciona el mecanismo de interacción entre componentes. Algunos de los dominios en Ptolemy II (CSP, DDE y PN) están basados en los procesos de hilos de Java. Los otros dominios (CT, DE, SDF) realizan su propia planificación para la interacción entre actores, en lugar de utilizar el modelo de hilos de Java. Esto hace que la ejecución de estos modelos sea mucho más eficaz.

Los modelos que implementan los dominios de Ptolemy II son gráficos que tienen la forma mostrada por la figura 4.1, donde los nodos son entidades y los arcos son relaciones. En la mayoría de los dominios, las entidades son actores (entidades con funcionalidad) y las relaciones que los conectan representan la comunicación entre actores.

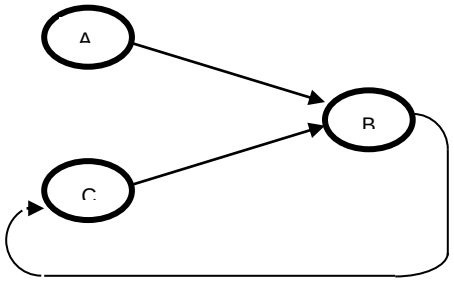


Figura 4.1. Sintaxis de los modelos en Ptolemy II.

4.1. DOMINIO DE EVENTOS DISCRETOS

El dominio de eventos discretos (Dominio DE) soporta modelos orientados a tiempo, como sistemas de colas, redes de telecomunicaciones y hardware digital, por tal motivo y según los propios creadores de la herramienta, es el dominio más importante en el área del diseño y modelado de Sistemas Telemáticos. En este dominio los actores se comunican por medio del envío de eventos, donde un evento es el valor de un dato (una señal) y una huella de tiempo.

4.1.1. PROPIEDADES

Las propiedades del dominio DE se describen a continuación.

4.1.1.1. TIEMPO MODELO

En el dominio DE, el tiempo es global, en el sentido de que todos los actores comparten el mismo tiempo global. A las señales que son enviadas desde el puerto de un actor hacia otro actor se les empaqueta una huella de tiempo, la cual lleva el tiempo modelo, o en algunos casos un tiempo futuro especificado por el actor. Esta huella de tiempo se utiliza para ordenar las señales, en este dominio, en la cola global de eventos. Un evento es removido de la cola global de eventos cuando el tiempo modelo es igual a su huella de tiempo.

4.1.1.2. EVENTOS SIMULTÁNEOS

Un aspecto importante del dominio DE es la priorización de los eventos simultáneos. Esto da al dominio un comportamiento semejante al dominio de flujo de datos para eventos que tienen huellas de tiempo idénticas. Esto se logra asignando un rango a cada actor. Cada rango es un entero no negativo, y es único para cada actor; es decir dos actores distintos no tienen asignado el mismo rango.
El rango de un actor determina la prioridad de eventos destinados a ese actor. Los eventos con la prioridad más alta son destinados a los actores con el rango más bajo.

4.1.1.3. ITERACIÓN

En cada iteración, el director escoge todos los eventos en la cola de eventos globales que tienen la huella de tiempo más pequeña. Los eventos escogidos son removidos de la cola de eventos globales y sus señales de datos son insertadas en los puertos de entrada apropiados del actor de destino. Luego el director itera el actor de destino, es decir llama a sus métodos para ser activado. Este proceso se repite hasta que no haya eventos con su huella de tiempo igual al tiempo actual del modelo. Esto concluye la iteración de un modelo. Una iteración, por consiguiente, procesa todos los eventos en la cola de eventos con la huella de tiempo más pequeña.

4.1.1.4. DETENCIÓN DE UNA EJECUCIÓN

Una ejecución se detiene cuando una de las siguientes condiciones se vuelve verdadera:

- El tiempo actual alcanza el tiempo de parada, este tiempo de parada es especificado en uno de los métodos del director DE.
- La cola de eventos globales queda vacía.

Los eventos en el tiempo de parada son procesados antes de parar la ejecución del modelo.

4.1.1.5. MUTACIONES

El director DE soporta cambios en el modelo durante la ejecución. Los cambios deben entrar en la cola donde el director o el manager definen las nuevas prioridades de los actores para realizar la ejecución de nuevo.

4.1.2. ARQUITECTURA

Las clases más importantes que se implementan en el kernel del dominio DE son DEDirector, DEActor y DEIOPort, las cuales se explicarán a continuación:

4.1.2.1. DEDirector

En el corazón de la clase DEDirector está la cola de eventos globales, la cual organiza eventos de acuerdo a su huella de tiempo y prioridades. Esta clase también se encarga de las operaciones de introducir y sacar en las colas los eventos de los actores.

4.1.2.2. DEActor

La clase DEActor proporciona los métodos necesarios para hacer uso del tiempo, el tiempo es parte esencial en los dominios temporizados como DE. También proporciona los actores específicos del dominio DE.

4.1.2.3. DEIOPort

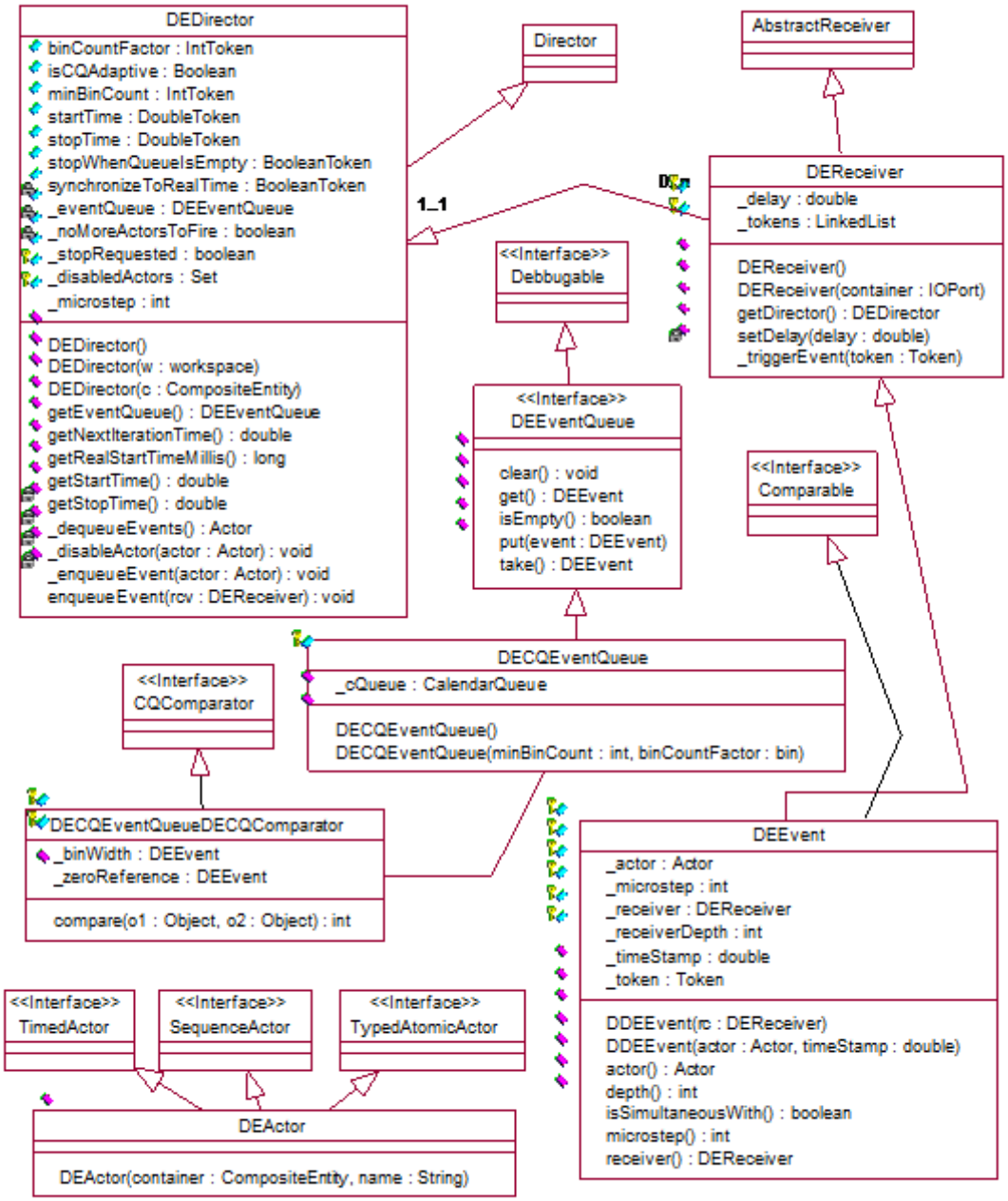
La clase DEIOPort es usada por los actores específicos del dominio DE. Soporta anotaciones que informan al planeador acerca de los retardos de los actores. También proporciona métodos para que los actores puedan enviar señales con un retraso de tiempo.

4.2. DOMINIO DE EVENTOS DISCRETOS DISTRIBUIDOS

El dominio de eventos discretos distribuidos (Dominio DDE) incorpora una noción de tiempo distribuido dentro de un estilo de comunicación de flujo de datos. En este modelo de computación el tiempo corre cuando los actores en el modelo se ejecutan y se comunican. Los actores en el dominio DDE se comunican enviando mensajes a través de canales FIFO (First in First out – Primero en entrar Primero en salir) limitados, estos actores tienen su propia noción de tiempo actual local. La

información de tiempo local es compartida entre dos actores solo si dichos actores se están comunicando.

Los modelos de eventos discretos distribuidos son bastante utilizados en aplicaciones donde se requiera conservar el ancho de banda de la red, o para explotar los recursos de computación paralelos.



F

Figura 4.2. Arquitectura del software del dominio DE.

4.2.1. PROPIEDADES

Las propiedades del dominio DDE son las siguientes:

4.2.1.1. HABILITACIÓN DE COMUNICACIÓN

4.2.1.1.1. Comunicando Señales

Un modelo DDE consiste de una red de actores que están conectados por medio de una cola FIFO unidireccional y limitada. Las señales son enviadas desde un actor transmisor a un actor receptor, la señal se coloca en la cola apropiada donde es almacenada hasta que el actor receptor la consume.

4.2.1.1.2. Comunicando Tiempo

Cada actor en un modelo DDE mantiene una noción de tiempo local. Como las señales son comunicadas entre actores, las huellas de tiempo están asociadas a cada señal, así que siempre que un actor consume una señal, el tiempo actual del actor se pone igual a la huella de tiempo de la señal consumida.

4.2.1.2. MANTENIENDO LA COMUNICACIÓN

En un modelo DDE pueden ocurrir puntos muertos temporizados (timed deadlock), debido a que en un modelo DDE se puede presentar el caso en que los actores estén bloqueados en lectura esperando mutuamente por señales con huella de tiempo que nunca aparecerán. Hay dos tipos de puntos muertos temporizados: punto muerto temporizado feedback y feedforward.

Un punto muerto feedforward ocurre cuando un grupo de actores conectados están en un punto muerto tal que todos los actores están bloqueados en lectura y al menos uno de los actores en el grupo está bloqueado en lectura en una cola de

entrada que tiene un tiempo de recepción que es menor que el reloj local del actor fuente de la ola de entrada.

Un punto muerto feedback ocurre cuando un grupo de actores conectados cíclicamente están en un punto muerto tal que todos los actores en el grupo están bloqueados en lectura y al menos un actor en el grupo, por ejemplo un actor X, está bloqueado en lectura en una cola de entrada que puede leer señales las cuales son directa o indirectamente un resultado de salida del mismo actor.

4.2.2. ARQUITECTURA

Para que un modelo tenga semántica DDE, debe tener un DDEDirector controlándolo. Esto asegura que los receptores de los puertos son DDEReceivers. Cada actor en un modelo DDE está bajo el control de un DDEThread (hilo DDE), este DDEThread contiene un TimeKeeper que maneja la noción de tiempo del actor.

4.2.2.1. MANEJO DEL TIEMPO LOCAL

Cuando un modelo DDE es inicializado, el DDEdirector asigna un hilo (DDEThread) a cada actor para llevar a cabo su proceso, este hilo crea un objeto llamado TimeKeeper y lo asigna al actor que controla, este objeto crea una relación con los receptores DDE para que estos puedan acceder a el y así determinar su tiempo.

4.2.2.2. DETECCIÓN DE PUNTOS MUERTOS

La función de detectar y (si es posible) resolver puntos muertos ya sea temporizados o no, la realiza el DDEDirector. Siempre que un receptor se bloquea informa al director, este sigue los procesos que están bloqueados y si tienen solución los soluciona. Algunos puntos muertos pueden ser resueltos incrementando la capacidad de la cola de los receptores que se encuentran bloqueados en escritura.

4.2.2.3. TERMINACIÓN DE LA EJECUCIÓN

La ejecución de un modelo termina si ocurre un punto muerto que no tiene solución, o si el tiempo de terminación del director es excedido por todos los actores que maneja o si se requiere una terminación temprana (puede ser por un botón de interfaz de usuario).

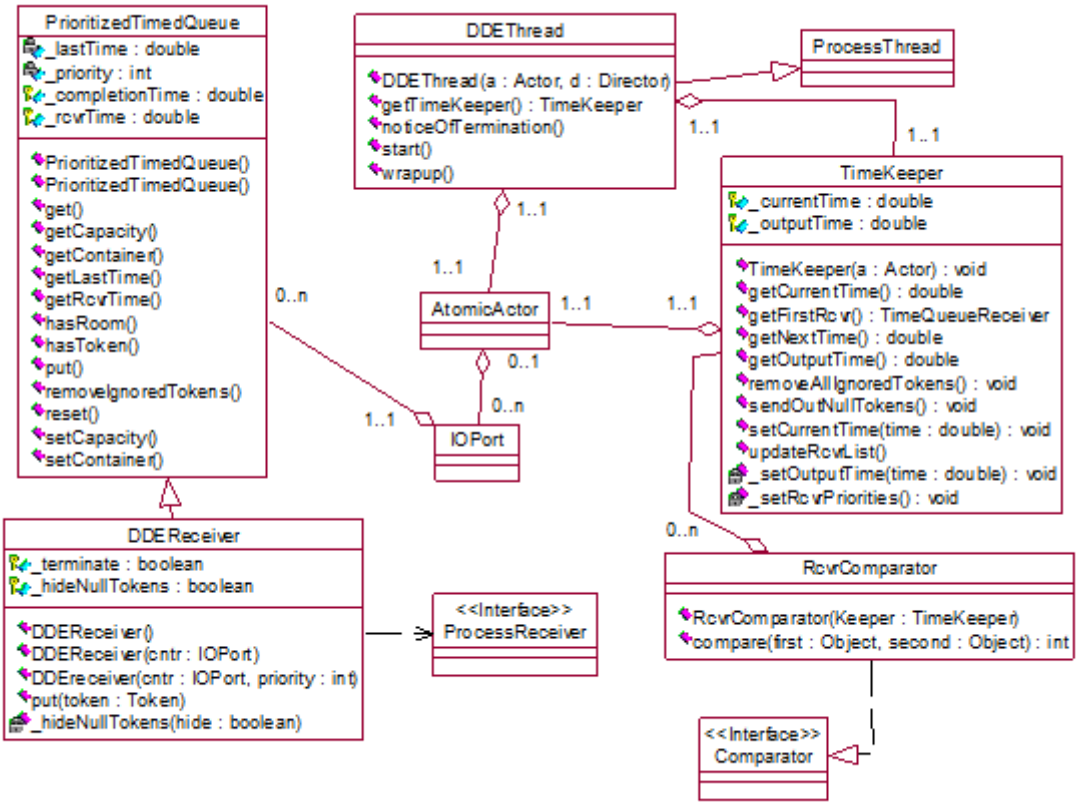


Figura 4.3. Arquitectura del Software del dominio DDE.

4.3. DOMINIO DE FLUJO DE DATOS SINCRÓNICO

El dominio de Flujo de Datos Sincrónico (Dominio SDF) se utiliza para modelar sistemas de flujos de datos simples que no necesitan ser controlados, como Sistemas de Procesamiento de Señales. Bajo el dominio SDF, el orden de la ejecución de los actores se determina de forma estática y antes de la ejecución.

Con esto se logra tener una ejecución con un mínimo de encabezado, así como un uso de memoria limitado y la garantía de no tener un punto muerto (deadlock).

Este modelo de computación es utilizado ampliamente en el área del procesamiento de señales, software empotrado de tiempo real y diseño de hardware.

4.3.1. PROPIEDADES

El dominio SDF es un modelo de computación sin noción de tiempo. Todos los actores bajo el dominio SDF consumen señales de entrada, realizan su computación y producen salidas en una operación atómica.

SDF es un dominio planeado de forma estática. La activación de un actor compuesto corresponde a una simple iteración del modelo contenido. Una iteración SDF consiste de una ejecución de la planeación SDF precalculada.

4.3.1.1. PLANEACIÓN

El primer paso en la construcción de la planeación es resolver las ecuaciones de balance. Estas ecuaciones determinan el número de tiempos en que cada actor se activará durante una ejecución.

El segundo paso en la construcción de una planeación SDF es el análisis del flujo de datos. El análisis del flujo de datos ordena la activación de los actores, basado en las relaciones entre ellos. Ya que cada relación representa el flujo de datos, el actor que está produciendo datos debe activarse antes que el actor consumidor.

4.3.1.2. PLANEACIÓN JERARQUICA

Hasta el momento se ha asumido que el gráfico SDF no es jerárquico. La manera más simple para planear un modelo SDF jerárquico es aplanar el modelo para quitar la jerarquía y luego planear el modelo como de costumbre. Esto se

interpreta en el sentido que en el momento de hacer la planeación se puede pasar por encima de la jerarquía sin tenerla en cuenta.

4.3.2. ARQUITECTURA

El paquete del kernel SDF implementa el modelo de computación SDF.

4.3.2.1. SDFDirector

La clase SDFDirector extiende la clase StaticSchedulingDirector. Cuando se crea un director SDF, automáticamente es asociado con una instancia de la clase scheduler por defecto, SDFScheduler.

El director tiene un parámetro, *iterations*, el cual determina un límite en el número de veces que el director desea ser activado.

El director SDF también tiene un parámetro que puede ser usado para requerir una ejecución vectorizada de un modelo. Este parámetro sugiere que el director modifique la planeación para que en lugar de activar cada actor una sola vez, se active las veces que se determine.

4.3.2.2. SDFScheduler

El SDFScheduler se deriva directamente de la clase Scheduler. Este planeador es el encargado de realizar la planeación básica de los gráficos SDF. Este planeador calcula la planeación en dos fases. Primero, resuelve las ecuaciones de balance del modelo y luego ordena los actores para que sean activados sólo cuando el planeador ha determinado que hay las suficientes señales en su puerto de entrada para permitir la ejecución.

4.3.2.3. SDFREceiver y SDFIOPort

Los modelos SDF tienden a producir y consumir grandes bloques de señales durante cada activación. Debido a esto los puertos y receptores SDF tienen métodos optimizados para el envío y recepción de bloques de señales en masa. Las clases SDFReceiver y SDFIOPort son las encargadas de implementar los métodos utilizados por los puertos y receptores del dominio SDF.

4.4. DOMINIO DE PROCESOS SECUENCIALES DE COMUNICACIÓN

El Dominio de procesos secuenciales de comunicación (Dominio CSP) en Ptolemy II modela un sistema como una red de procesos secuenciales que se comunican mediante el paso sincrónico de mensajes a través de canales. Si un proceso está listo para enviar un mensaje, se bloquea hasta que el proceso de recepción esté listo para aceptar el mensaje.

Los modelos CSP son útiles para aplicaciones donde la utilización compartida de recursos es un elemento clave en el sistema, como en el caso de las bases de datos cliente-servidor y la multiplexión o multitarea de recursos hardware.

4.4.1. PROPIEDADES

En el dominio CSP se tienen dos propiedades fundamentales para llevar a cabo la comunicación. La primera es la noción de comunicación atómica y la segunda es la noción de escogencia no determinista.

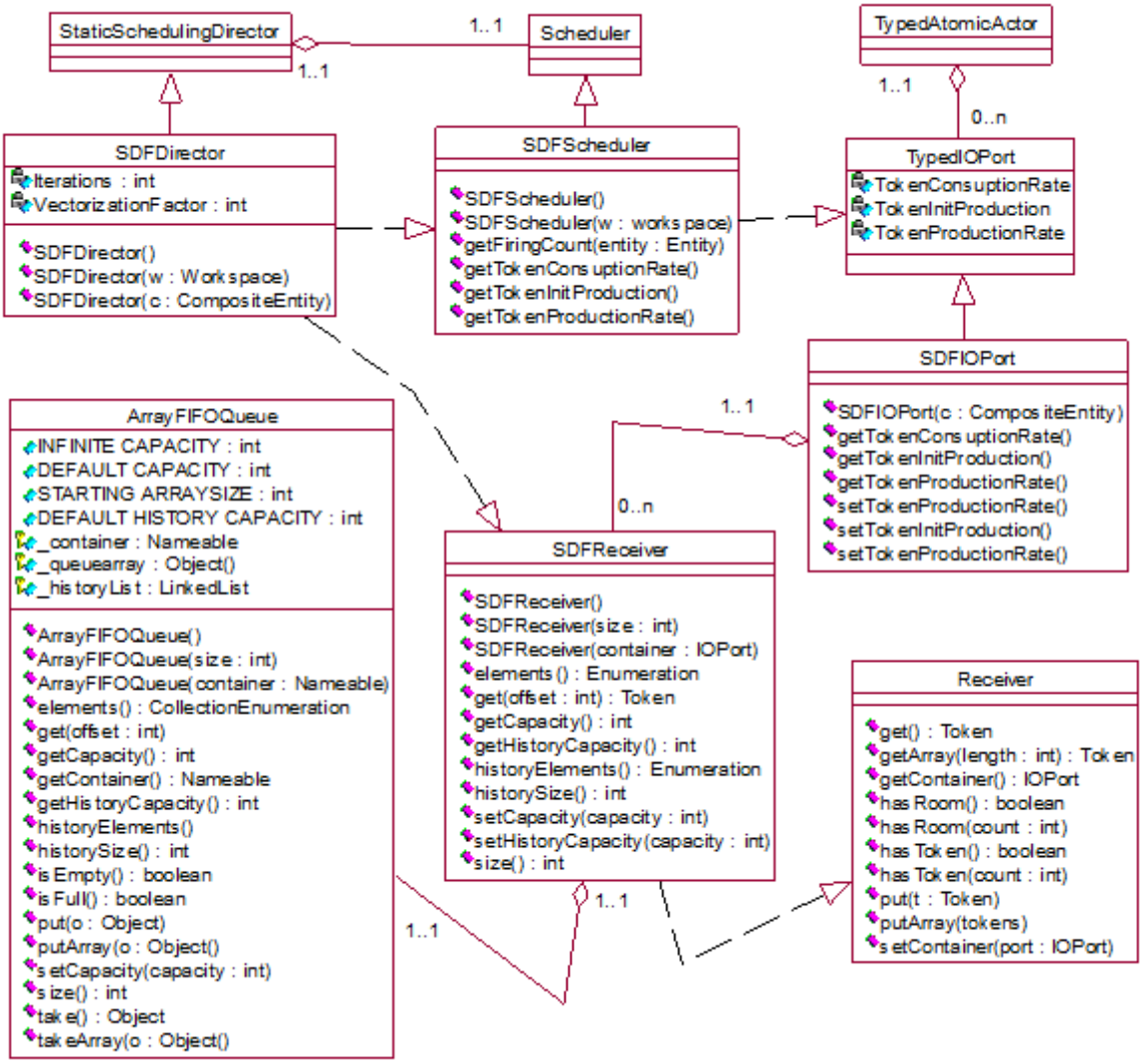


Figura 4.4. Arquitectura del Software del dominio SDF.

4.4.1.1. COMUNICACIÓN ATÓMICA: RENDEZVOUS

La comunicación atómica es implementada por medio de la técnica rendezvous e implica que el envío y la recepción de un mensaje ocurren simultáneamente. Durante el rendezvous los procesos de envío y recepción se bloquean hasta que el otro lado esté listo para comunicarse; el acto de enviar y recibir son actividades indistinguibles ya que una no puede ocurrir sin la otra. Una analogía en el mundo real de los procesos rendezvous son las comunicaciones telefónicas, en donde el

que llama y el llamado deben estar presentes simultáneamente para que una conversación telefónica ocurra.

4.4.1.2. ESCOGENCIA: RENDEZVOUS NO DETERMINISTA

La escogencia no determinista proporciona procesos con la habilidad de seleccionar de forma aleatoria entre un grupo de posibles comunicaciones atómicas.

4.4.1.3. PUNTO MUERTO

Un punto muerto es una situación en donde ninguno de los procesos puede progresar, ya sea porque están bloqueados tratando de comunicarse o están suspendidos. Como se puede ver se tienen dos tipos de puntos muertos, los cuales son:

- Punto muerto real. Todos los procesos activos están bloqueados tratando de comunicarse.
- Punto muerto de tiempo. Todos los procesos activos están bloqueados tratando de comunicarse o están suspendidos, y al menos un proceso esta suspendido.

4.4.1.4. TIEMPO

En el dominio CSP, el tiempo es centralizado. Esto quiere decir que todos los procesos en un modelo comparten el mismo tiempo, el cual es conocido como tiempo modelo actual. Cada proceso puede retardarse por un periodo relativo al tiempo modelo actual, en cuyo caso se dice que el proceso está suspendido.

4.4.2. ARQUITECTURA

En un modelo CSP, el director es una instancia de la clase CSPDirector. Ya que el modelo es controlado por un CSPDirector, todos los receptores en los puertos son

CSPReceivers. La combinación de estos CSPReceivers con el CSPDirector en los puertos constituye la semántica de un modelo CSP. El dominio CSP en Ptolemy II está compuesto por cinco clases principales, las cuales proporcionan toda la infraestructura necesaria para un modelo CSP. Las clases son las siguientes:

- CSPDirector. Proporciona la semántica para un modelo CSP.
- CSPReceiver. Asegura la comunicación de mensajes entre procesos.
- CSPActor. Adiciona la noción de tiempo y la capacidad de realizar comunicaciones condicionales.
- ConditionalReceive, ConditionalSend. Estas clases son utilizadas para construir comunicaciones condicionales.

4.4.2.1. INICIANDO EL MODELO

El director es el encargado de iniciar el modelo, asignando a cada actor un hilo el cual está bajo su control, y llamando a los métodos necesarios de todos los actores del modelo para que empiecen a iniciarse al mismo tiempo.

4.4.2.2. DETECCIÓN DE PUNTOS MUERTOS

Para detectar puntos muertos el director tiene tres contadores:

- El número de procesos activos.
- El número de procesos bloqueados.
- El número de procesos suspendidos.

Cuando el número de procesos bloqueados es igual al número de procesos activos entonces se ha llegado a un punto muerto real. Cuando el número de procesos bloqueados más el número de procesos suspendidos es igual al número de procesos activos, y al menos hay un proceso suspendido se tiene entonces un punto muerto de tiempo.

4.4.2.3. TERMINANDO EL MODELO

Un proceso puede finalizar ya sea porque realizó toda su ejecución sin ningún problema y termina normalmente, o porque ha ocurrido algún problema en la ejecución y se tiene una terminación temprana de los procesos.

4.4.2.4. PAUSANDO / REACTIVANDO EL MODELO

Pausar y reactivar un modelo no afecta el resultado de la ejecución de un modelo particular, lo único que afecta es la tasa de progreso. La ejecución de un modelo puede ser pausada en cualquier estado por medio del director. De la misma forma la ejecución se puede reactivar. La pausa y reactivación de un modelo es utilizada para funciones de control de interfaz de usuario.

4.5. DOMINIO DE REDES DE PROCESOS.

El Dominio de redes de procesos (Domino PN) en Ptolemy II modela un sistema como una red de procesos que se comunican entre si mediante el paso de mensajes a través de canales FIFO unidireccionales. Un proceso se bloquea cuando intenta leer de un canal que se encuentra vacío hasta que un mensaje este disponible en el canal. Este modelo es determinista en el sentido de que la secuencia de valores comunicados en los canales es completamente determinada por el modelo.

El dominio PN es un modelo natural para la descripción de sistemas de procesamiento de señales donde los flujos infinitos de muestras de datos son transformados incrementalmente por una colección de procesos ejecutándose en paralelo. Los sistemas empotrados de procesamiento de señales son buenos ejemplos de tales sistemas. Este dominio también puede ser usado para modelar la concurrencia en diferentes componentes hardware de un sistema empotrado.

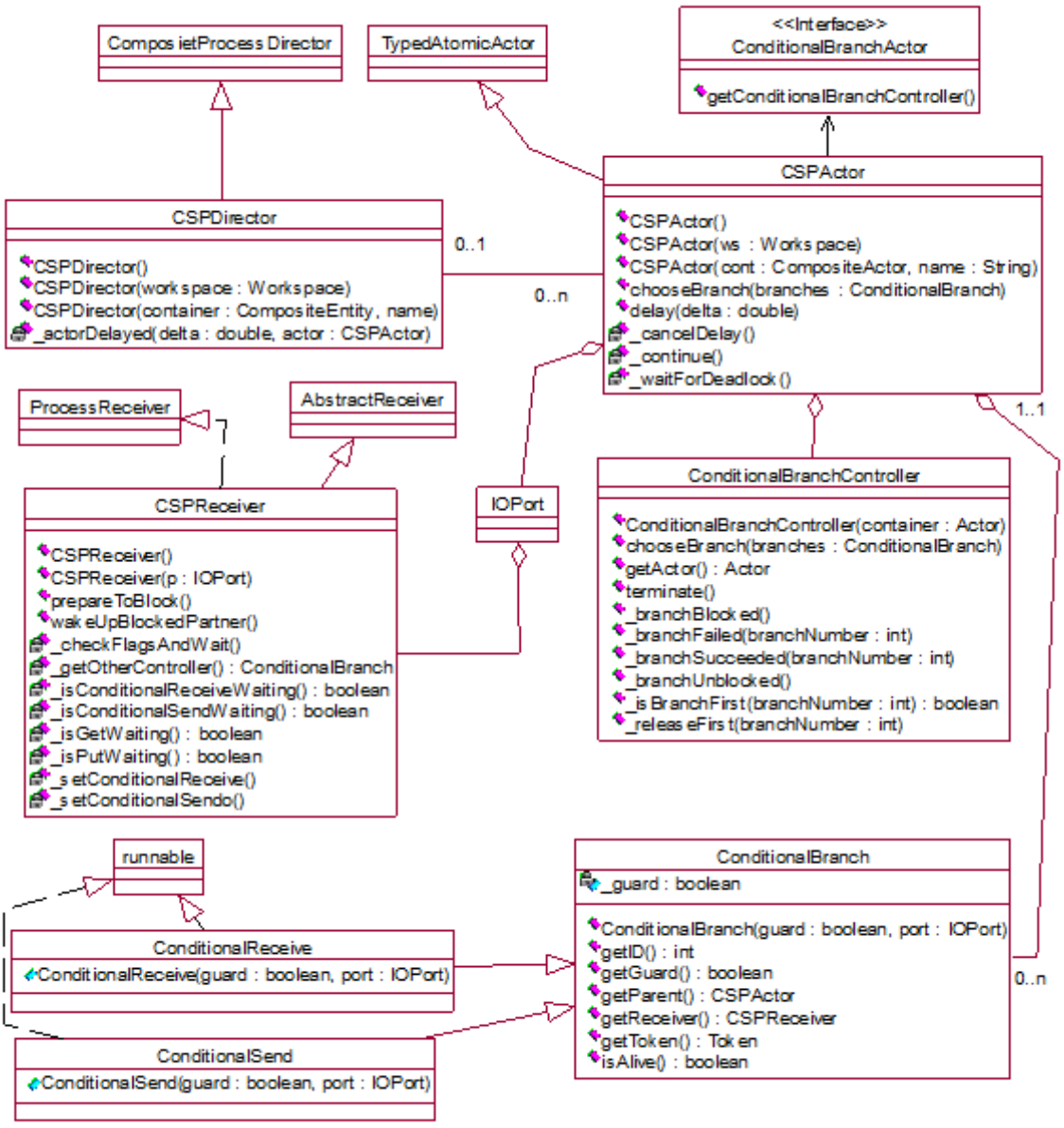


Figura 4.5. Arquitectura del Software del dominio CSP.

4.5.1. PROPIEDADES

En el dominio PN se han implementado dos propiedades importantes las cuales son: los procesos se comunican asincrónicamente (por colas ordenadas) y la memoria usada en la comunicación es limitada. El dominio PN puede ser usado con o sin noción de tiempo.

4.5.1.1. COMUNICACIÓN ASINCRONICA

En este modelo de computación los procesos son conectados por los canales de comunicación que forman una red. Estos procesos producen elementos de datos o señales y los envían a lo largo de un canal de comunicación unidireccional donde son almacenados en una cola FIFO hasta que el proceso de destino los consuma. Esta es una forma de comunicación asincrónica entre procesos. Los canales de comunicación son el único método en que los procesos pueden hacer uso del intercambio de información. Un grupo de procesos que se comunican a través de una red de colas FIFO define un programa.

4.5.1.2. EJECUCIÓN EN MEMORIA LIMITADA

El alto nivel de concurrencia que presentan las redes de procesos las hace ideales para aplicaciones software de sistemas empotrados y para el modelado de implementaciones hardware. La mayoría de estas aplicaciones son hechas para ejecutarse con una cantidad de memoria limitada.

Para implementar esta ejecución en memoria limitada, se utilizan los bloqueos de escritura, asignando una capacidad fija, para cada canal FIFO y obliga a los procesos a bloquearse temporalmente si un canal está lleno.

4.5.1.3. TIEMPO

En el dominio PN de Ptolemy II el tiempo es global. Esto quiere decir que todos los procesos en un modelo comparten el mismo tiempo, el cual es conocido como tiempo modelo.

4.5.1.4. MUTACIONES

El dominio PN tolera mutaciones, las cuales son cambios en tiempo de ejecución en la estructura del modelo. Por lo general, las mutaciones son realizadas en forma de requerimientos de cambios en una cola por parte del director o el

manager. En el caso del dominio PN, las mutaciones ocurren cuando se ha llegado a un punto muerto.

4.5.2. ARQUITECTURA

El kernel del dominio PN se encuentra en el paquete ptolemy.domains.pn.kernel.

4.5.2.1. BasePNDirector

Esta clase extiende la clase básica ProcesssDirector, y es la encargada de proporcionar los directores que gobiernan la ejecución de los actores en este dominio.

4.5.2.2. PNDirector

PNDirector extiende la clase BasePNDirector para manejar las mutaciones localmente. Este director soporta mutaciones y estas son procesadas tan pronto como se necesiten. La mutación es no determinista y puede pasar en cualquier punto de la ejecución del modelo.

4.5.2.3. TimedPNDirector

TimedPNDirector extiende la clase BasePNDirector introduciendo una noción de tiempo global al modelo. También proporciona una ejecución determinista de las mutaciones. Las mutaciones son necesarias cuando se llega a un punto muerto temporizado, y ya que la ocurrencia de estos puntos muertos es determinista, las mutaciones en estos puntos deben ser deterministas.

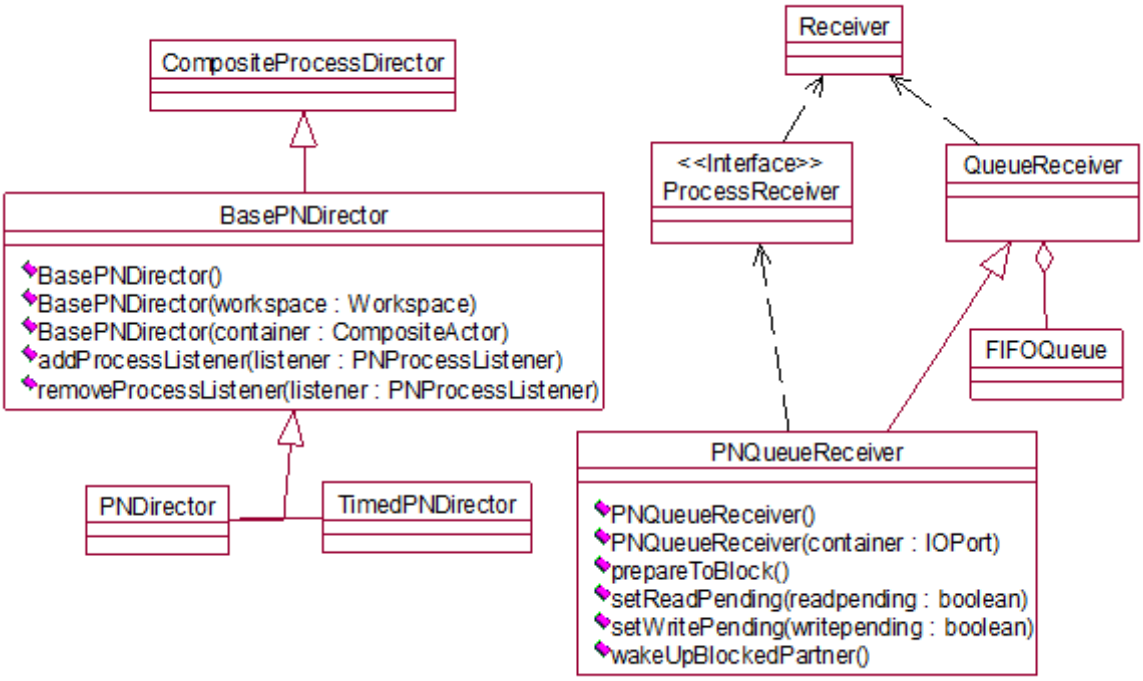


Figura 4.6. Arquitectura del Software del dominio PN.

4.5.2.4. PNQueueReceiver

El PNQueueReceiver contiene una cola FIFO que representa un canal de comunicaciones en una red de procesos. También proporciona el mecanismo para la ejecución determinista de mutaciones. Las mutaciones son requeridas cada vez que se llegue a un punto muerto temporizado, y como estos puntos muertos son deterministas, las mutaciones deben serlo también.

4.6. DOMINIO DE TIEMPO CONTINUO

El Dominio de tiempo continuo (Dominio CT) tiene como meta ayudar al diseño y simulación de sistemas que pueden ser modelados usando ecuaciones diferenciales ordinarias (ODE's). Las ODE's son usadas generalmente para modelar circuitos analógicos, plantas dinámicas en sistemas de control, conjuntos de parámetros en sistemas de carga y muchos otros sistemas físicos.

En general, una ODE basada en un sistema de tiempo continuo tiene la siguiente forma:

$$x' = f(x, u, t) \quad (1)$$
$$y = g(x, u, t) \quad (2)$$
$$x(t_0) = x_o \quad (3)$$

donde t pertenece a los números reales, $t \geq t_o$, y es el tiempo continuo. Y x' es la derivada de x con respecto al tiempo. Las ecuaciones 1, 2 y 3 se llaman dinámica del sistema, plano de salida y condición inicial del sistema.

El dominio CT es útil para modelar sistemas físicos que se pueden describir con ecuaciones algebraicas o diferenciales, tales como circuitos analógicos, circuitos de microondas y sistemas o componentes mecánicos, los cuales se encuentran generalmente en sistemas empotrados.

4.6.1. PROPIEDADES

Las propiedades del dominio CT se describen a continuación.

4.6.1.1. TIEMPO

La característica más importante del dominio CT es la continuidad del tiempo. Esto implica que un sistema continuo en el tiempo tiene un comportamiento en cualquier instancia de tiempo. La maquina de simulación del dominio CT es capaz de computar el comportamiento del sistema en cualquier punto del tiempo.

4.6.1.2. DISCONTINUIDAD

La existencia y unicidad de la solución de una ODE permite que la parte derecha de la ecuación 1 sea discontinua en un número contable de puntos discretos D, los cuales son llamados puntos de quiebre (breakpoints). Estos puntos de quiebre

pueden ser causados por la discontinuidad de una señal u. En teoría, las soluciones en estos puntos no están bien definidas. Por lo tanto, para encontrar las soluciones en estos puntos, se debe intentar encontrar los límites por la derecha y la izquierda en donde estas ecuaciones están bien definidas.

4.6.2. ARQUITECTURA

El dominio CT consiste de los siguientes paquetes, ct.kernel, ct.kernel.util, ct.kernel.solver y ct.lib; estos paquetes implementan la semántica del dominio y los actores y directores encargados de trabajar en este dominio.

4.6.2.1. PAQUETE ct.kernel.

Este es el paquete más importante del dominio CT ya que proporciona las interfaces para clasificar los actores, planeadores, directores, y la clase básica para resolver las ODE's.

4.6.2.2. PAQUETE ct.kernel.util

Este paquete proporciona una estructura básica de datos, la cual es usada para almacenar los puntos de quiebre y luego procesarlos en el orden en que aparecieron.

4.6.2.3. DIRECTORES CT

En el dominio CT hay tres directores encargados de la ejecución de los actores de este dominio. Estos directores son: CTMultiSolverDirector, CTMixedSignalDirector y CTEmbeddedDirector. El primero es usado para dirigir la ejecución global de un actor, el segundo es usado para dirigir la ejecución de un actor compuesto (un grupo de varios actores) y el tercero es usado para dirigir actores CT que se encuentran trabajando en otros dominios o modelos de computación.

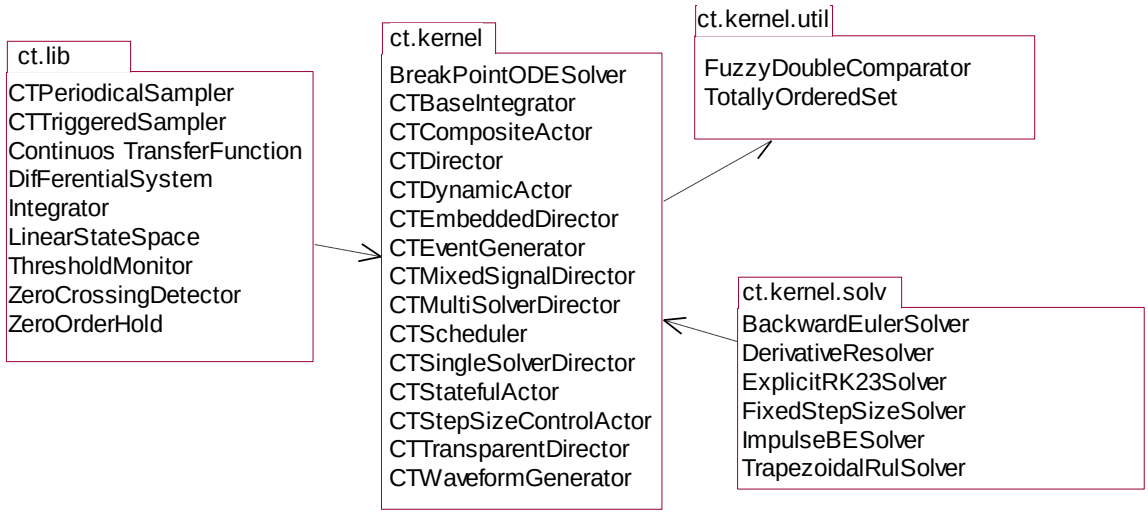


Figura 4.7. Arquitectura del Software del dominio CT.

5. METODOLOGÍA PARA LA CONSTRUCCIÓN DE ACTORES EN PTOLEMY II

El presente capítulo contiene una descripción de la metodología para el desarrollo de nuevos actores de la herramienta Ptolemy II v.1.0.1., así como también se presenta un ejemplo de aplicación de la misma

Esta metodología formaría parte de una etapa más amplia que constituiría la metodología para el Modelamiento de un Sistema Telemático con Ptolemy II.

En esta metodología se describirán las etapas que permitirán la construcción de nuevos actores que cumplan con los requerimientos de la herramienta es decir, que sean entidades funcionales, y harán posible el modelamiento de un sistema sin limitarse a los actores actuales de Ptolemy. Con esta metodología se busca construir los actores optimizando el tiempo y el trabajo del personal involucrado y logrando un producto con el mayor grado de calidad. Para la elaboración de esta metodología se han tomado pautas del M.R.D.P sobre todo en la etapa de planeación, así como algunas recomendaciones del Proceso Unificado de Desarrollo referentes al modelo del proceso de desarrollo.

La metodología es utilizada principalmente por el equipo de trabajo encargado de la creación del actor para Ptolemy II. Los usuarios de la metodología deben tener conocimiento del lenguaje de programación Java y principalmente conocimiento profundo de la herramienta Ptolemy II, que les permita comprender el vocabulario empleado en la misma y los conceptos y elementos que en ella se tratan; por lo tanto si no se tiene un conocimiento previo de Ptolemy II, es fundamental leer la documentación de la misma. Un usuario individual que pretenda construir un actor sencillo y que no necesariamente este dentro de un sistema especifico debe tener conocimientos básicos de Java y fundamentales de Ptolemy II.

5.1 PROCESO DE DESARROLLO

5.1.1 PLANEACIÓN

Esta fase es importante para determinar la organización de todos los recursos necesarios para el desarrollo del actor; tanto recursos técnicos como humanos, la definición y organización de estos recursos facilita la asignación de actividades y la distribución de los recursos existentes. Desde la etapa de identificación de funciones, hasta la etapa de validación del nuevo actor, los recursos cumplen labores especificas cuyo cumplimiento es una de las garantías de la calidad del actor que se va a generar.

Objetivo General:

- Elaborar el Plan del Proyecto para la creación del actor.

Objetivos Específicos:

1. Conformar el equipo de trabajo para el desarrollo del actor.
2. Definir el ambiente de soporte requerido para el desarrollo del actor

Prerrequisito:

- Funciones del sistema. Estas funciones las da la persona encargada de identificar las funciones del sistema en el proyecto general, que son entendidas como los bloques operacionales del mismo.
- Requerimiento para implementar en Ptolemy II funcionalidades no realizadas por los actuales actores de la herramienta.

Actividades:

1. Definir Cronograma de actividades.
2. Especificación de la organización del recurso humano que desarrollará el actor.
3. Definición y configuración del ambiente de desarrollo para realizar el actor.

Subproductos:

1. Cronograma de actividades.
2. Conformación del Equipo de Desarrollo.
3. Configuración del Ambiente de Desarrollo.

Desarrollo de Actividades:

1. DEFINIR CRONOGRAMA DE ACTIVIDADES.

En la planeación del proyecto se debe realizar un Cronograma de actividades que en una forma preliminar involucre los siguientes aspectos: actividad, encargado, recursos, esfuerzo y fecha. Este Cronograma podrá utilizarse como referencia del progreso del trabajo y como un control de las actividades que involucran el desarrollo del actor

A continuación se muestra la planeación del proyecto, en la cual se especifican las actividades, así como los recursos necesarios y el esfuerzo empleado en cada una de estas. Esta estructura da una referencia para realizar el Cronograma de actividades.

1. Planeación.
    a. Planeación del proyecto. Es recomendable que esta actividad sea realizada por la persona encargada del grupo de trabajo. Los recursos que se necesitan para llevar a cabo esta actividad son de índole logístico o administrativo.
    b. Especificación de la organización del recurso humano que desarrollará el actor. Esta actividad debería ser llevada a cabo por el encargado del proyecto. Los recursos utilizados para hacer esta tarea son materiales de tipo logístico.
    c. Definición y configuración del ambiente de desarrollo para realizar el actor. Esta actividad debería ser realizada por una persona que tenga experienca con Sistemas Telemáticos y otra persona que tenga conocimientos fundamentales en la herramienta Ptolemy II. Los recursos necesarios para realizar esta tarea son: la documentación de la herramienta y los conocimientos previos de esta, también se necesitan elementos de tipo logístico.

2. Etapa 1. Identificación de funciones.
    a. Establecimiento de la función a diseñar. Esta actividad debería estar a cargo de la persona experta en Sistemas Telemáticos. Para llevar

a cabo esta tarea se necesita documentación de las librerías de actores de Ptolemy II.

b. Identificación de los tipos de datos que el nuevo actor va a manejar. Es recomendable que esta actividad la realice un experto en Sistemas Telemáticos y en Ptolemy II. Los recursos que se necesitan para realizar esta actividad son la documentación de la herramienta Ptolemy II.

3. Etapa 2. Diseño del actor.
    a. Establecer las directrices básicas para la identificación de las clases. Esta actividad debe ser llevada a cabo por una persona experta en Ptolemy II. Se requiere para realizar esta tarea la documentación de la herramienta.
    b. Proporcionar las directrices para la identificación de las características de las clases. Esta actividad debería ser responsabilidad de un experto en Sistemas Telemáticos y un experto en Ptolemy II. En esta actividad los recursos necesarios son la documentación de la herramienta Ptolemy II y una herramienta para modelamiento.
    c. Reusabilidad. Esta actividad debería estar a cargo de un miembro del grupo experto en Ptolemy II. Para llevarla a cabo se necesita la documentación de la herramienta.

4. Etapa 3. Validación del actor como entidad funcional.
    a. Codificación del actor teniendo en cuenta los elementos dados por la Etapa 2. Se recomienda que esta actividad sea responsabilidad de un experto en Ptolemy II. Para esta actividad se requiere la herramienta de codificación de lenguaje Java (Java 2 SDK Standard Edition v. 1.3.0_02).
    b. Inclusión del nuevo actor en la librería de actores de Ptolemy II. Esta actividad debería estar a cargo de un experto en Ptolemy II. Los recursos necesarios para llevar a cabo esta actividad son el manual de Administración de Ptolemy II y las herramientas en él especificadas.

c. Elaboración de un plan de pruebas. Esta actividad debería ser realizada por un experto en Sistemas Telemáticos. Para la realización de esta actividad se necesitan los conocimientos de la entidad funcional que implementa el actor y elementos de tipo logístico.
d. Elaboración de un modelo sencillo que permita un comportamiento predecible del actor para comprobar su funcionalidad. Este modelo es recomendable que sea elaborado por un experto en Ptolemy II. Para la realización de esta actividad se requiere la herramienta Ptolemy II.
e. Probar el actor dentro del sistema especificado para el cual se creó. Esta actividad debería ser llevada a cabo por una persona experta en Sistemas Telemáticos y el experta en Ptolemy II. Los recursos necesarios para realizar esta actividad es la herramienta Ptolemy II y recursos adicionales de acuerdo al sistema.

Un factor a tener en cuenta es el esfuerzo requerido para cada una de las actividades mencionadas que dependerá en gran medida de la experiencia y conocimientos de las personas involucradas en el equipo de trabajo y de la acertada organización y planeación del proyecto.

2. ESPECIFICACIÓN DE LA ORGANIZACIÓN DEL RECURSO HUMANO.

Para que el proyecto se pueda llevar a cabo satisfactoriamente es necesario contar con un grupo humano que conozca a fondo el proyecto y esté interesado en éste. En el caso de la metodología para la construcción de nuevos componentes para la plataforma Ptolemy II, teniendo en cuenta el énfasis que se hace en los sistemas telemáticos en este trabajo, se decidió trabajar con el modelo de referencia organizacional propuesto en el Modelo de Referencia para Desarrollo de Proyectos versión 1.1 (MRDP), el cual es un modelo que tiene como sistemas objetivo los sistemas telemáticos y presenta características que lo convierten en

una referencia metodológica útil para construir soluciones de calidad, oportunas y con costos competitivos [MRDP ver. 1.1][9].

El grupo humano necesario para la realización del proyecto se compone de tres personas las cuales serán, un líder del proyecto, una persona que conozca a fondo la herramienta Ptolemy II y el lenguaje de programación Java y otra persona experta en los sistemas telemáticos, la cual pueda reconocer cuales son los componentes que ya se encuentran en la herramienta Ptolemy II y cuales se necesita crear para llevar a cabo el modelado y diseño de sistemas telemáticos.

El líder del proyecto debe estar encargado de definir la metodología que se debe seguir para el desarrollo del proyecto, en este caso para la construcción de un nuevo actor de la herramienta Ptolemy II, hacer que se aplique la metodología adoptada por el proyecto, verificar y validar cada fase del producto con el equipo desarrollador antes de pasar a la siguiente etapa y entregar el actor trabajando en la herramienta.

El experto en Sistemas Telemáticos es la persona que conoce a fondo este tema, sabe qué componentes son necesarios para realizar el diseño de estos sistemas y que tipo de aplicaciones abarcan. Esta persona es la encargada de acuerdo a los actores que se tiene en la herramienta de valorarlos para saber cuales de estos se pueden aplicar a los Sistemas Telemáticos y cuales deberían ser creados ya que no se encuentran en la herramienta.

El experto en Ptolemy II es la persona que conoce a fondo la herramienta, sus características y componentes. Es el encargado de aplicar la reusabilidad es decir ver qué componentes se pueden volver a utilizar en la herramienta, estos componentes pueden ser actores o clases que se puedan heredar para evitar la duplicación de código a la hora de construir un nuevo actor, también se encarga de que los nuevos componentes se puedan volver a usar en la herramienta y no sirvan para una sola aplicación. Cuando se tenga el modelo del nuevo actor esta persona es la encargada de generar el prototipo de este, es decir es el encargado de realizar el código del nuevo actor y de incluirlo en la herramienta.

[9] Carlos E. Serrano C. Modelo de Referencia para Desarrollo de Proyectos Versión 1.1. Universidad del Cauca. 1997.

El grupo de desarrollo está conformado por el experto en Ptolemy II y el experto en Sistemas Telemáticos, estas dos personas son las encargadas de todo el proceso de creación del actor, y están dirigidos por el líder del proyecto quien es el que se encarga de verificar que los requerimientos se cumplan en cada fase del proyecto.

3. DEFINICIÓN Y CONFIGURACIÓN DEL AMBIENTE DE DESARROLLO.

Para llevar a cabo satisfactoriamente un proyecto se necesita contar con las herramientas apropiadas, teniendo en cuenta los principios básicos que estas deben tener para obtener una solución de calidad. A continuación se describen las herramientas necesarias para trabajar en la creación de actores en Ptolemy II.

Sistema Operativo.
Ptolemy II puede ser instalado bajo Windows 2000 o bajo las diferentes versiones de Unix.

Windows 2000.
Este sistema operativo presenta grandes ventajas como la facilidad de uso, ya que posee una interfaz de usuario gráfica amigable, lo cual es optimo para usuarios que no están familiarizados con un ambiente de comandos, como Unix.

Herramientas de Desarrollo.
Las herramientas utilizadas para el desarrollo de actores en Ptolemy II son las siguientes.

Ptolemy II.
La herramienta Ptolemy II es un software de libre distribución desarrollado por la Universidad de California en Berkeley. Esta herramienta está desarrollada en el lenguaje de programación Java, y sus librerías pueden ser modificadas por el usuario.
Java Run Time Environment (Java 2 SDK Standard Edition v 1.3.0_02.).

La herramienta JDK es utilizada para poder correr Ptolemy II y para programar los nuevos actores. Una de las ventajas más importantes de utilizar Java es su independencia de la plataforma, lo que indica que es independiente del computador que se esté utilizando.

Java es un lenguaje orientado a objetos y esto hace que soporte las tres características más importante del paradigma orientado a objetos: encapsulación, herencia y polimorfismo.

Cygwin.

La herramienta Cygwin simula un ambiente Unix para Windows. Se necesita para llevar a cabo el proceso de compilación de Ptolemy II.

Herramientas de soporte a las actividades logísticas y administrativas tales como editores de texto, hojas de calculo entre otras.
Se utiliza esta herramienta para generar los diferentes documentos que se requieren.

Rational Rose

La herramienta Rational es utilizada para modelar los nuevos actores, y así obtener un modelo que permita realizar la codificación del mismo de una manera más sencilla.

5.1.2 ETAPA 1. IDENTIFICACIÓN DE FUNCIONES

Esta etapa es necesaria para filtrar las funciones y definir claramente el actor a diseñar.

Objetivo General:

- Determinar las funciones que no están implementadas en Ptolemy II.

Objetivo Específico:

1. Identificar los tipos de datos que el actor va a manejar.

Prerrequisitos:

- Lista de Funciones. Esta lista es producto del análisis del sistema.

Actividades

1. Establecimiento de la función a diseñar.
2. Identificación de los tipos de datos que el nuevo actor va a manejar.

Subproducto

1. Especificación de la función que se ha identificado.
2. Especificación de los datos que va a manejar el actor

Desarrollo de Actividades

1. ESTABLECIMIENTO DE LA FUNCIÓN A DISEÑAR.

En esta primera actividad se determinarán las funciones que han sido identificadas como aquellas que no son implementadas por ningún actor dentro del paquete de actores de Ptolemy II. Esta parte consiste en determinar las funciones que específicamente requieran una entrada de datos, hagan unas operaciones o modificación sobre los mismos y finalmente produzca una salida de datos.

2. RELACIÓN DE LOS DATOS QUE VA A MANEJAR EL ACTOR.

Identificar los tipos de datos que el nuevo actor va a manejar, tanto los que recibe, como los que produce. El objetivo de esta actividad es definir los tipos de datos que manejaran los puertos de entrada y salida del actor. Ptolemy II maneja diversos tipos de datos que se han especificado en el capítulo 3 sección 3.4.2.

2.1. Datos de Entrada.

De acuerdo a la funcionalidad del nuevo actor se determinan los tipos de datos que el actor va a recibir.

2.2 Datos de Salida
Se determinan los tipos de datos que se producirán a la salida teniendo en cuenta la funcionalidad del mismo lo que determina las operaciones que realiza sobre los datos de entrada.
→Procedimiento:
Identificar los tipos de datos que el actor va a manejar como entradas y salidas y clasificarlos dentro de los siguientes tipos:

- Enteros
- Enteros largos
- Reales
- Boléanos
- Cadenas
- Arreglos de cualquiera de los 4 anteriores
- Matrices

5.1.3 ETAPA 2. DISEÑO DEL ACTOR

Esta etapa contiene los elementos y las pautas básicas para el diseño del actor.

Una vez definida la función que implementará el actor y los tipos de datos que va a manejar se pasa a la etapa siguiente que consiste en definir la estructura del nuevo actor. En la estructura de Ptolemy II los actores son clases que implementan una funcionalidad especifica.

Objetivo General:

- Diseñar los componentes del actor que se va a generar.

Objetivos específicos

1. Identificar las clases al diseño.
2. Identificar las características de las clases.
3. Definir el manejo de puertos.

Prerrequisito

Subproducto de la etapa 1 el cual debe contener:

- Especificación de la función a implementar.
- Especificación de los tipos de datos que manejará el actor.

Actividades

1. Identificación de la clase básica.
2. Identificación de las características de la clase.
3. Identificación de reusabilidad

Subproducto

1. Modelo del actor. El modelo del actor debe contener:

- La clase básica que instancia.
- Los atributos que la determinan.
- Los métodos públicos y privados que implementa.
- Los paquetes que utiliza.
- Las clases básicas que hereda para el manejo de los puertos.

Desarrollo de Actividades

1. IDENTIFICACIÓN DE LA CLASE BÁSICA.

Esta actividad brida la posibilidad de establecer en forma preliminar, la clase básica que será implementada en el diseño del actor.

1.1 Pautas para la identificación de clases:

→Procedimiento:

- Con la funcionalidad que el actor va a implementar ya definida, se procede a clasificarla dentro de una de las siguientes categorías:

A. Es una función que indica la transformación de un dato de entrada mediante algún procedimiento en un dato de salida. Tiene entrada y salida de datos.

B. Es una función que indica que produce o es una fuente de señales. Generalmente solo requiere salida de datos, pero bajo ciertas condiciones puede requerir una entrada que condicione la salida.
C. Es una función que despliega señales, ya sea en forma de texto, esquema o en tablas. Solo tiene entradas.
D. Funciones que impliquen especificar los tipos de los puertos y los parámetros para permitir restricciones en el tipo de los mismos, pero a la vez no caen en las categorías anteriores. Las entradas y salidas tienen un tipo especifico de datos.

- Una vez identificada la categoría de la funcionalidad se hace la siguiente clasificación:
  - Si la función fue clasificada como A: Se puede decir que es una función transformadora, esta función puede ser implementada por la clase básica Transformers.
  - Clasificada como B: Se puede concluir que es una función que genera una señal con una característica especifica, es una fuente de señal, esta función puede ser implementada por la clase básica Sources.
  - Es una función que despliega algún tipo de resultados, de señales, bien sea en tablas, en esquemas o simplemente texto, esta función describe el destino final para la señal, esta función puede ser implementada por la clase básica Sinks.
  - Definitivamente la función no implementa ninguna de las anteriores, pero requiere la definición del tipo de datos y puertos, en este caso la clase que la implementa sería una extensión de la clase básica TypedAtominActor.

2. IDENTIFICACIÓN DE LAS CARACTERÍSTICAS DE LA CLASE

En esta actividad se va a diseñar los atributos y los métodos de la clase.

2.1 DISEÑO DE ATRIBUTOS

2.1.1. DISEÑO DE PUERTOS

Por convención los puertos son miembros públicos de los actores. Ellos median entrada y salida. Los puertos en general son instancias de la clase básica TypedIOPort, a menos que requieran los servicios de dominios específicos, en tal caso pueden ser instancias de una subclase de dominio especifico.

→Procedimiento: Al crear el puerto se debe tener en cuenta lo siguiente:

a) Tipos de datos que manejará el puerto.
   →Procedimiento: Revisar los tipos de datos que manejará el actor del subproducto de la Etapa 1. Esto se utiliza generalmente cuando el diseñador quiere restringir el tipo del dato que va a manejar el puerto. Si los puertos van a manejar todos los tipos de datos no se requiere definir el tipo.

b) Definir la clase básica del puerto:
- Si el puerto operará como instancia de TypedIOPort.
- Si el puerto requiere servicios específicos de algún dominio en particular.
  →Procedimiento:
  Definir la clase básica del puerto:
  - Puertos simples polimórficos: son instancias de la clase básica TypedIOPort.
  - Puertos simples de dominio especifico: son instancias de un subpaquete de dominio especifico, como por ejemplo DEIOPort, para el dominio DE (Dominio de eventos discretos).

c) El puerto debe tener un contenedor.
d) Asignar al puerto un nombre de identificación.
f) Establecer si el puerto es de entrada o salida o se desea que cumpla ambas funciones.
   →Procedimiento:
   - Al crear el actor se debe introducir:
     - El contenedor del puerto.
     - El nombre del puerto.

- Determinar si el puerto es de entrada o salida.

- La siguiente es la estructura formato del código para el puerto:

```
portName:newTypedIOPort(contenedor,"portname",true, false);
```

→Procedimiento: Verificar los siguientes campos:

- El campo 1 contiene al nombre del puerto.
- El campo 3 corresponde al contenedor.
- Los campos 5 y 6 se refieren a si son puertos de entrada o salida respectivamente.

g) Si el puerto manejara un solo canal o se requiere que maneje múltiple canales ya sea de entrada o de salida; por consiguiente el puerto se clasificará como puerto simple o multipuerto.

→Procedimiento:

- Multipuertos: Por defecto los puertos son simples, si se requiere un multipuerto puede ser declarado con una sentencia como:
  portName.setMultiport(true);

2.1.2 PARÁMETROS:

Como los puertos, por convención, los parámetros son miembros públicos de los actores. Los parámetros son las variables que modifican los métodos. Estos representan los diferentes valores que pueden tener o se les puede asignar a los métodos para que con la misma secuencia de instrucciones pueda generar distintos resultados.

→Procedimiento: Identificar los parámetros que declaran los métodos.

- De acuerdo a la funcionalidad que se ha seleccionado se deben identificar aquellos parámetros que contiene las variables que utiliza el método para trabajar sobre los datos de entrada.
- La abstracción que se haga de la función brinda las pautas para la determinación de los parámetros que se necesitan en el diseño del actor.

- Una utilidad importante es restringir el tipo de los parámetros, si se requiere que el actor maneje solo cierto tipo de señal.
  →Procedimiento: Verificar si se requieren parámetros con o sin restricción de tipo.

- La clase básica Parameter, permite la modificación de los parámetros desde la interfaz de usuario, así que si requiere emplear esta utilidad debe emplear esta clase.
  →Procedimiento: Definir los parámetros que el usuario podrá visualizar y editar desde la interfaz de usuario.

- El tipo del parámetro se puede restringir o no. Esta restricción se puede hacer con el fin de evitar el manejo de tipos arbitrarios de datos cuando se requiere algún tipo determinado. La restricción más común es fijar el tipo a uno especifico, aunque también se permiten restricciones arbitrarias en los valores del parámetro.
  →Procedimiento: Verificar si se requiere hacer una restricción en el tipo de algún parámetro determinado.

  Nota:
  - Los actores por defecto no permiten los cambios de tipo en los parámetros, se puede cambiar el valor pero no el tipo.
  - El cambio de tipo del parámetro se permite anulando el método attributeTypeChanged() definido en la clase básica para actores NamedObj.

2.2. DISEÑO DE MÉTODOS

Las operaciones propias del actor serán ejecutadas por los métodos. Un método es una secuencia de instrucciones a la que se da un nombre.

Se debe determinar qué tipo de método, si es público o privado, además la operación que realiza y los resultados que esta arroja,

2.2.1 Métodos Públicos:

Existen en la herramienta unos métodos públicos que son básicos para la invocación, ejecución y terminación de un actor, estos son llamados los métodos de acción. A continuación se mencionarán estos métodos y su función:

- initialize(): Es el método utilizado para invocar la ejecución de un actor.

- prefire(), fire() y postfire(), son los métodos encargados del proceso de activación de un actor.

- stoptfire() es invocado para pedir la suspensión de una activación de un actor.

- wrapup() es el responsable de borrar todo después de que la ejecución de un actor ha sido completada, como por ejemplo vaciar los buffers y eliminar hilos activos.
- terminate(), este método podría ser llamado en cualquier momento de la ejecución para detenerla enseguida. Este método es usado como último recurso para interrumpir una ejecución.

→Procedimiento:

- El método fire() es el que activa al actor y permite el paso de los mensajes a través de los puertos, por lo tanto este método es fundamental en el diseño del actor.
- Los otros métodos de acción se utilizan de acuerdo a las necesidades del diseñador en cuanto a invocación y terminación de un actor.

2.2.1.1 Métodos para los puertos:

- Lectura y escritura en puertos:

Los métodos send() y get() sirven para enviar y recibir señales a través de puertos respectivamente. Se puede enviar la señal a un puerto en particular o a todos los

puertos disponibles. Para el primer caso se debe especificar el número del canal y en el segundo se utiliza el método broadcast().

2.2.2 Métodos Privados.

Como métodos privados se clasifican aquellos que realizan las operaciones propias del actor, es decir, aquellas que implementan su funcionalidad.

2.2.2.1 Verificación de los tipos de métodos:

Los métodos pueden ser de dos tipos: los que arrojan resultados y los métodos que ejecutarán el proceso sin arrojar resultados.

→Procedimiento: Verificar si la funcionalidad a implementar arroja algún o ningún resultado. En el primer caso se debe tener en cuanta que se debe tener una sentencia return; en el segundo el método tendrá la expresión void.

- Teniendo en cuenta los componentes de los métodos tenemos:

- Instrucciones: Tener en cuenta que las instrucciones se clasifican dentro de las siguientes categorías:

1) Llamada: Hacen que se ejecute un método.

→Procedimiento: Si se requiere hacer una llamada a un método en el cuerpo de otro se utilizará una llamada a método.

2) Asignación: Cambian el estado de un campo usando otro valor del mismo.

→Procedimiento: Si se requiere cambiar los valores de la clase se utiliza un método de asignación.

3) Repetición: Llevan a cabo la instrucción una y otra vez

→Procedimiento: Si se requiere ejecutar un ciclo o instrucción dado por una condición.

4) Selección: Deciden si ejecutar ciertas instrucciones o no.

→Procedimiento: Verificar si se tendrán operaciones cuya ejecución depende del cumplimiento de ciertas condiciones o de la decisión de diseño del programador.

5)Excepción: Detectan y reaccionan en circunstancias excepcionales.
→Procedimiento: Si se requiere tener control sobre alguna condición inusual dentro de un método, se pueden implementar excepciones. Java tiene muchos objetos de excepción predefinidos y también se pueden crear excepciones propias.

3. IDENTIFICACIÓN DE REUSABILIDAD

El actor puede hacer uso de las clases básicas que tiene la herramienta y que le permitirán utilizar eficientemente el código ya generado por los desarrolladores de la misma. El objetivo de esta actividad es reducir o evitar la cantidad de código duplicado en el diseño además de asegurar y reforzar la consistencia con los elementos existentes en la herramienta.

3.1 Actores.

Existen clases básicas que pueden ser instanciadas. Estas clases representan las diferentes funcionalidades que tienen los actores. Por ejemplo, los actores transformers, extienden la clase básica Transformer. Estos actores leen datos desde las entradas, luego los modifican de alguna manera produciendo entonces un resultado. La clase Transformer implementa un puerto de entrada y uno de salida, los cuales son instancias de TypedIOPort. Los actores Sinks, extienden la clase básica Sinks y sirve para el despliegue de datos. Los actores Sources extienden la clase básica Sources; esta clase provee un puerto de salida y un puerto de entrada trigger, cuyo propósito es almacenar eventos que ocasionen la activación del actor. En el anexo Librerías de Actores encuentra las clases básicas que extienden los actores.

Estas son las principales clases básicas que describen funcionalidad de los actores, para más información sobre otras clases básicas se sugiere consultar la documentación que trae la herramienta.

3.2 Puertos.

Todo actor debe contar con puertos de entrada y/o salida. Existe una clase básica para la creación de los puertos, por lo tanto dentro de los actores sólo se crearían instancias de esa clase básica para la creación de los puertos. Esta clase es TypedIOPort. Si los puertos requieren los servicios específicos de un dominio en particular se utiliza la clase basica xIOPort donde x representa el dominio.

3.3 Referencia de los paquetes que debe importar el actor a diseñar.

La arquitectura global de Ptolemy consiste en un grupo de paquetes que proveen soporte genérico para todos los dominios. Los subpaquetes extienden la funcionalidad de algunos paquetes y proporcionan modularidad y encapsulamiento.

Los principales paquetes y subpaquetes a tener en cuenta en el cuerpo del actor son:

- **actor.** Este paquete soporta entidades ejecutables que reciben y envían datos a través de puertos. Éste incluye actores con tipo y actores sin tipo. Para actores con tipo, implementa un sistema tipo sofisticado que soporta polimorfismo. Incluye la clase base Director para clases de dominios específicos que controla la ejecución de un modelo.
  →Procedimiento:
  Este es el paquete principal donde se encuentra la librería de actores, su utilización es básica pues indica en donde se encuentra el actor. Dentro del cuerpo del actor no se hace alusión explícita a este paquete padre.

- **actor.gui.** Este subpaquete es una librería de actores polimórficos con componentes de interfaz de usuario, más algunas clases básicas convenientes para applets y aplicaciones.

- **actor.lib.** Este subpaquete es una librería de actores polimórficos.

→Procedimiento: Es un subpaquete del paquete actor y al diseñar un actor polimórfico, este es el paquete contenedor de dicho actor. Es el que explícitamente se escribe dentro del cuerpo del actor mediante la linea:
package actor.lib

- **data.** Este paquete proporciona clases que encapsulan y manipulan datos que son transportados entre actores en los modelos de Ptolemy II.
  →Procedimiento: Este es el paquete padre que encapsula los tipos, operaciones y utilidades de los datos que maneja Ptolemy II, cuando el actor manipula datos explícitamente se debe incluir este paquete.

- **data.expr.** Esta clase soporta un lenguaje de expresión extensible y un intérprete para este lenguaje. Los parámetros pueden ser valores especificados por expresiones. Estas expresiones podrían referirse a otros parámetros. Las dependencias entre parámetros son manejadas transparentemente, como en una hoja de cálculo, donde la actualización del valor de uno resultará en la actualización de todos los que dependan de él.
  →Procedimiento:
  - Cuando se desea que el actor compute expresiones dadas por el usuario final del mismo; estas expresiones pueden representar entradas y parámetros del actor. En este caso se debe usar este subpaquete.
  - Si se requiere diseñar un actor que manipule datos, símbolos o arreglos, se debe utilizar este paquete.
  - Cuando se definen parámetros en términos de otros parámetros, se requiere que los cambios en uno se reflejen en el otro, en este caso también se emplea este subpaquete.

- **graph.** Este paquete proporciona algoritmos para la manipulación y análisis de gráficas matemáticas. Las gráficas matemáticas son más simples que las gráficas agrupadas de Ptolemy II en las que no hay jerarquía.
  →Procedimiento:
  Cuando dentro de la funcionalidad del actor se requiere la utilización y manipulación de gráficas matemáticas, se debe emplear este paquete.

- **kernel.** Este paquete proporciona la arquitectura del software para la sintaxis abstracta clave, gráficas agrupadas. Las clases en este paquete soportan entidades con puertos, y relaciones que conectan los puertos. El agrupamiento es donde una colección de entidades es encapsulada en una sola entidad compuesta, y un subgrupo de los puertos de las entidades internas son expuestos como puertos de la entidad agrupada.

  →Procedimiento:

  Este paquete padre contiene las clases básicas para la caracterización de los paquetes, para los puertos y demás elementos de los actores, debe estar presente en el cuerpo del actor.

- **kernel.util.** Este subpaquete del paquete kernel proporciona una colección de clases que no dependen del paquete kernel.

  →Procedimiento:

  Este paquete debe estar presente en el actor porque contiene todas las utilidades de nombramiento, contención y manejo del actor.

5.1.4 ETAPA 3. VALIDACIÓN DEL ACTOR COMO ENTIDAD FUNCIONAL

La creación del actor tuvo como base la implementación de una funcionalidad concreta y bien definida que será la base para hacer la validación del actor.

En esta etapa se espera construir un prototipo que en una primera aproximación pueda ser incluido en la librería de actores de la herramienta.

Una vez creado el prototipo del actor es necesario realizar una serie de pruebas que evalúen el desempeño del nuevo actor, tanto como una entidad funcional, como en un entorno específico.

Objetivo General:

- Verificar que las funcionalidades planteadas sean las que el actor está ejecutando.

Objetivos Específicos:

1. Realización del prototipo inicial.

3 Determinar las pruebas de diseño del actor necesarias para comprobar su funcionamiento.

4 Realizar pruebas al actor como entidad funcional.

5 Realizar pruebas de integración del actor.

Prerrequisitos:

Modelo del actor. El modelo del actor debe contener:

- La clase básica que instancia.
- Los atributos que la determinan.
- Los métodos públicos y privados que implementa.
- Los paquetes que utiliza.
- Las clases básicas que hereda para el manejo de los puertos.

Actividades:

1. Codificación del actor .
2. Inclusión del nuevo actor en la librería de actores de Ptolemy II.
3. Elaboración de un plan de pruebas.
4. Elaboración de un modelo sencillo que permita un comportamiento predecible del actor para comprobar su funcionalidad y capacidad de integración.
5. Probar al actor dentro del sistema especificado para el cual se creó.

Subproducto

Prototipo del Actor.

Plan de pruebas de funcionamiento.

Desarrollo de Actividades

1. CODIFICACIÓN DEL ACTOR

Teniendo el modelo del actor generado en la Etapa 2 se procede a hacer la implementación del mismo.
→Procedimiento: Esta actividad implica la codificación del actor teniendo en cuenta el diseño previo especificado en el modelo del actor entregado en la etapa.

2. INCLUSIÓN DEL NUEVO ACTOR EN LA LIBRERÍA DE ACTORES DE PTOLEMY II.

Teniendo el manual de Administración de la herramienta, se procede a introducir el nuevo actor en la librería de actores de la misma con el fin de hacer las pruebas de funcionamiento requeridas.
→Procedimiento: Con el manual de administración realice los pasos necesarios para la inclusión del nuevo actor en la herramienta.

3. ELABORACIÓN DE UN PLAN DE PRUEBAS.

El diseño del actor determina el tipo de pruebas que deben ser realizadas al actor para validar su funcionalidad, es decir, los tipos de datos que maneja y los tipos de puertos que tiene.
→Procedimiento: Elabore un plan de pruebas del actor generado que tenga el siguiente esquema básico:

Valor de entrada al Actor	**Resultado Esperado**	**Resultado Obtenido**

El usuario puede especificar parámetros adicionales a tener en cuenta en el plan de pruebas que va a implementar sobre el actor generado.

4. ELABORACIÓN DE UN MODELO SENCILLO QUE PERMITA UN COMPORTAMIENTO PREDECIBLE DEL ACTOR PARA COMPROBAR SU FUNCIONALIDAD Y CAPACIDAD DE INTEGRACIÓN.

Para poder implementar el plan de pruebas se debe construir un modelo sencillo que tenga un comportamiento predecible de acuerdo a la funcionalidad del actor, el modelo puede tener las siguientes categorías:

A. Es un actor heredado de la clase básica Transformer:

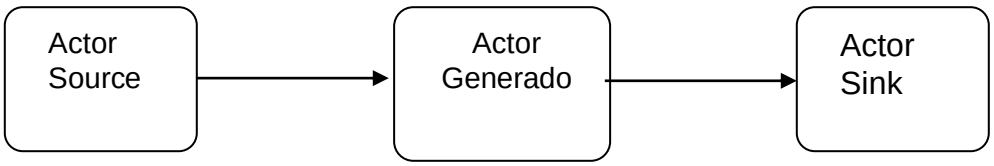


Figura 5.1. Modelo básico de prueba para la categoría A.

El tipo de datos depende del especificado en el diseño del actor.

B. Actor heredado de la clase básica Source:

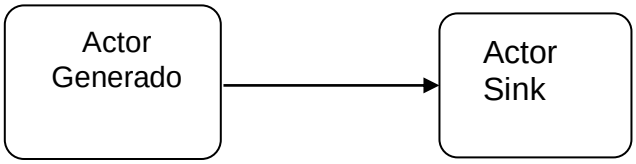


Figura 5.2. Modelo básico de prueba para la categoría B.

El despliegue de resultados debe corresponder al tipo de datos especificado en el diseño del actor.

C. Actor heredado de la clase básica Sinks:

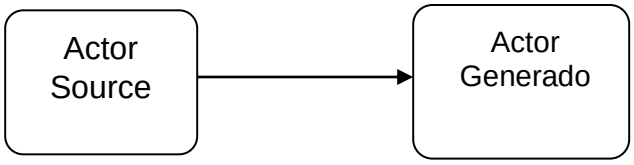


Figura 5.3. Modelo básico de prueba para la categoría C.

El despliegue de datos debe realizarse en la forma especificada en el diseño del actor, bien sea como texto, tablas o gráficas.

Estos modelos proporcionan una forma inmediata de verificar la funcionalidad del actor y su integración y comportamiento con otros actores de la herramienta.
→Procedimiento: Realice el modelo de prueba del actor de acuerdo a la categoría del mismo.

4.1 Si se ha determinado una restricción en el tipo de los datos, esta debe verificarse cambiando el tipo de dato establecido por otro y esperando un mensaje de error.
→Procedimiento: En el menú del actor Edit Parameters modifique el tipo del actor por cualquier otro y corra el modelo, debe aparecer un mensaje de error que notifique la condición de error en el tipo de dato que esta entrando al actor.

5.2. CONSTRUCCION DE UN ACTOR UTILIZANDO LA METODOLOGIA PROPUESTA.

En el siguiente apartado se describe el desarrollo de un codificador de audio G.711 utilizado en el estándar H.323 para la codificación de voz en redes de paquetes realizado en la herramienta Ptolemy II, siguiendo la metodología propuesta para la construcción de actores en Ptolemy II. Se hará una introducción al sistema general del codificador G.711, enmarcándolo dentro de la Gateway para voz sobre IP y el estándar H.323.

5.2.1. ESTÁNDAR H.323

H.323 es un estándar que especifica los componentes, protocolos y procedimientos necesarios para utilizar servicios de comunicación multimedia (audio en tiempo real, video y datos) sobre redes de paquetes, incluyendo redes IP. El estándar H.323 hace parte de una familia de recomendaciones ITU-T llamadas H.32x que tratan acerca de servicios de comunicación multimedia sobre una gran variedad de redes.

El estándar H.323 es el punto de partida para la transmisión de audio, video y datos en tiempo real sobre redes de paquetes.

5.2.1.1. COMPONENTES DENTRO DEL ESTÁNDAR H.323

El estándar H.323 especifica cuatro clases de componentes que al ser unidos, proveen servicios de comunicación multimedia punto a punto y punto a multipunto incluso a terminales de una RCC (Red de Conmutación de Circuitos). Ellos son:

- Terminales.
- Gateways.
- Gatekeepers.
- Unidades de Control Multipunto (MCU's).

5.2.1.1.1. Gateways

Una gateway conecta dos redes diferentes. Un gateway H.323 proporciona conectividad entre una red H.323 y una red que no los es. Por ejemplo, una gateway puede habilitar la comunicación entre un terminal H.323 y un terminal perteneciente a una red de conmutación de circuitos. La gateway no es necesaria para comunicar dos terminales dentro de una red H.323.

5.2.1.2. PILA DE PROTOCOLOS ESPECIFICADOS POR H.323

H.323 es independiente de la red de paquetes y de los protocolos de transporte sobre los cuales corra y no los especifica (ver Figura 5.4). Los protocolos especificados por H.323 se enlistan a continuación

- Codificadores y decodificadores de audio
- Codificadores y decodificadores de video
- H.225 registro, admisión y estado (RAS)

- H.225 señalización de llamada
- H.245 control de la señalización
- Protocolo de trasferencia de tiempo real (RTP)
- Protocolo de control de tiempo real (RTCP)

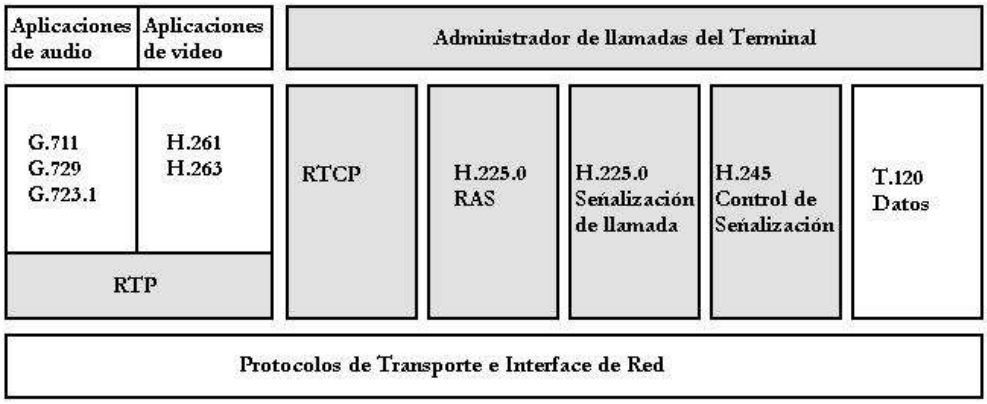


Figura 5.4. Protocolos del estándar H.323.

5.2.1.2.1. Codificadores y decodificadores de audio

Un Codificador de audio codifica la señal de audio proveniente del micrófono para trasmitirla a través del terminal H.323 y el decodificador decodifica la señal de audio codificada que se recibe, enviándola luego al parlante del terminal. Dado que audio es el servicio mínimo especificado por el estándar H.323, todos los terminales H.323 deben tener soporte para al menos un CODEC (codificador y decodificador) de audio, tal como lo especifica la recomendación ITU-T G.711 (codificación de audio a 64Kbps). Otras recomendaciones acerca de CODEC's de audio que también puede ser soportadas son la G.722 (64, 56 y 48 Kbps), la G.723.1 (5.3 y 6.3 Kbps), la G.728 (16Kbps) y la G.729 (Kbps).

5.2.2. CODIFICADOR DE AUDIO G.711

La recomendación G.711 especifica la Modulación por Impulsos Codificados (PCM) de frecuencia vocales a 64 Kbps. La ley de codificación que implementa G.711 debe cumplir con las siguientes características:

- Se deben utilizar 8 dígitos binarios por muestra.
- Se recomiendan dos leyes de codificación, las cuales son ley A y ley μ.
- El número de valores cuantificados viene dado por la ley de codificación.
- Conversión a PCM uniforme y a partir de ella.

5.2.3. DESARROLLO DEL CODIFICADOR DE AUDIO G.711 CON PTOLEMY II

A continuación se presenta el proceso de desarrollo de un codificador G.711, el cual implementa la ley de codificación μ, usado en el estándar H.323 para codificar señales de voz y transmitirlas por redes de paquetes.

El diagrama de bloques de un codificador G.711 es el siguiente (figura 5.5):

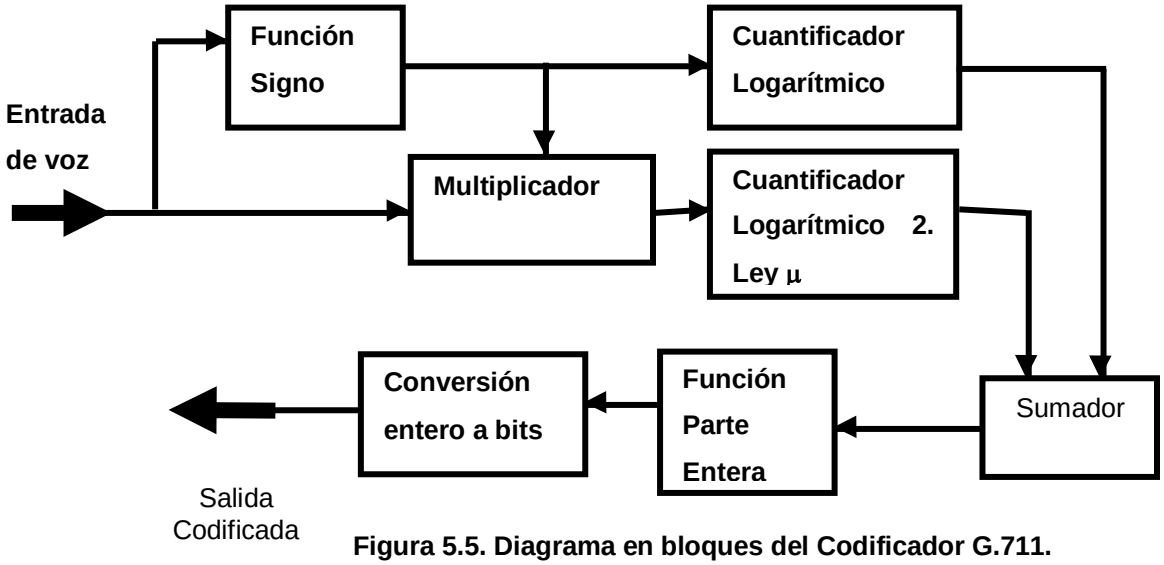


Figura 5.5. Diagrama en bloques del Codificador G.711.

En el diagrama cada uno de los bloques representa una función especifica que puede ser realizada con los actores de Ptolemy II. La función de cuantificación

logarítmica, no se encuentra implementada en ninguno de los actores de la herramienta, por lo tanto se deberá crear un actor que la implemente.

5.2.3.1. CREACIÓN DEL ACTOR CUANTIFICADOR LOGARÍTMICO

Según la metodología propuesta para la creación de nuevos actores en la herramienta Ptolemy II, el desarrollo del nuevo actor debe seguir las siguientes etapas.

- Fase de Planeación.
- Etapa 1. Identificación de Funciones.
- Etapa 2. Diseño del actor.
- Etapa 3. Validación del actor como entidad funcional.

5.2.3.1.1. Planeación

En esta fase se determinan los recursos técnicos y humanos necesarios para la creación del nuevo actor. Las actividades que involucra esta fase se explican a continuación.

1. Definir Cronograma de Actividades

En esta actividad se realiza el cronograma de actividades y planeacion para la realización de todas las actividades de cada una de las etapas en que se divide el proceso de creación de un actor. A continuación se muestra en las tablas 1.1.a. y 1.1.b. la planeación del proyecto, la cual incluye las personas responsables de llevar a cabo cada actividad, los recursos necesarios y el esfuerzo empleado para realizar dichas actividades, y que se constituye también en el Cronograma de Actividades.

Etapa	Actividad	Responsable	Recursos	Esfuerzo
Planeación	Definir cronograma de Actividades	Ing. Mario Solarte.	Materiales de escritorio.	2 horas.
Planeación	Especificación de la organización del recurso humano.	Ing. Mario Solarte.	Materiales de escritorio.	1 hora.
Planeación	Definición y configuración del ambiente de desarrollo.	Alejandro Barragán. Marina Tejada.	Documentación de la herramienta	2 horas por persona.
Etapa 1.	Establecimiento de la función a diseñar.	Marina Tejada.	Documento de librería de actores de Ptolemy II.	8 horas.
Etapa 1.	Identificación de los tipos de datos que el nuevo actor va a manejar.	Marina Tejada.	Documentación de la herramienta Ptolemy II.	1 hora.
Etapa 2.	Identificación de la clase básica.	Alejandro Barragán.	Documentación de la herramienta Ptolemy II.	4 horas.
Etapa 2.	Identificación de las características de la clase.	Alejandro Barragán. Marina Tejada.	Documentación de la herramienta Ptolemy II y herramienta de modelamiento Rational Rose.	4 horas por persona.

Etapa 2.	Identificar Reusabilidad.	Alejandro Barragán.	Documentación de la herramienta Ptolemy II.	4 horas.
Etapa 3.	Codificación del actor.	Alejandro Barragán.	Herramienta de programación en Java (Java 2 SDK).	6 horas.
Etapa 3.	Inclusión del nuevo actor en la librería de actores de Ptolemy II.	Alejandro Barragán.	Manual de Administración de Ptolemy II y herramientas de soporte de Ptolemy II.	1 hora.
Etapa 3.	Elaboración de un plan de pruebas.	Marina Tejada.	Conocimientos de la entidad funcional que implementa el nuevo actor.	2 horas.
Etapa 3.	Elaboración de un modelo sencillo que permita un comportamiento predecible del actor para comprobar su funcionalidad.	Alejandro Barragán.	Herramienta Ptolemy II.	1 hora.
Etapa 3.	Probar el actor dentro del sistema especificado para el cual se creó.	Alejandro Barragán. Marina Tejada.	Herramienta Ptolemy II.	1 hora por persona.

Tabla 1.1.a.

El trabajo se realizará en las siguientes fechas:

ETAPA	ACTIVIDAD	FECHA INICIO	FECHA FINALZACION
Planeación	Planeación del Proyecto.	3 Sept. / 2001	4 Sept. / 2001
Planeación	Especificación de la organización del recurso humano.	4 Sept. / 2001	4 Sept. / 2001
Planeación	Definición y configuración del ambiente de desarrollo.	5 Sept. / 2001	6 Sept. / 2001
Etapa 1.	Establecimiento de la función a diseñar.	7 Sept. / 2001	10 Sept. / 2001
Etapa 1.	Identificación de los tipos de datos que el nuevo actor va a manejar.	10 Sept. / 2001	10 Sept. / 2001
Etapa 2.	Identificación de la clase básica.	11 Sept. / 2001	12 sept. / 2001
Etapa 2.	Identificación de las características de las clase.	12 Sept. / 2001	13 Sept. / 2001
Etapa 2.	Identificación de Reusabilidad.	13 Sept. / 2001	13 Sept. / 2001
Etapa 3.	Codificación del actor.	14 Sept. / 2001	17 Sept. / 2001
Etapa 3.	Inclusión del nuevo actor en la librería de actores de Ptolemy II.	17 Sep. / 2001	17 Sept. / 2001
Etapa 3.	Elaboración de un plan de pruebas.	19 Sept. / 2001	21 Sept. / 2001
Etapa 3.	Elaboración de un modelo sencillo que permita un comportamiento predecible del actor para comprobar su funcionalidad.	21 Sept. / 2001	25 Sept. / 2001
Etapa 3.	Probar el actor dentro del sistema especificado para el cual se creó.	27 Sept. / 2001	1 Oct. / 2001

Tabla 1.1.b.

2. Especificación de la organización del Recurso Humano

Para la realización de este proyecto se definen tres personas las cuales deben desempeñar los roles descritos en la sección de la organización del recurso

humano de la metodología en el capítulo 5 de la monografía, estas personas son las encargadas de llevar a cabo el proyecto de creación de un actor. Las personas encargadas de desempeñar estos roles se nombran a continuación:

- Líder del proyecto. Ing. Mario Fernando Solarte.
- Experto en Sistemas Telemáticos. Marina Tejada A.
- Experto en Ptolemy II. Alejandro Barragán C.

3. Definición y configuración del ambiente de desarrollo

En esta actividad se definen las herramientas necesarias para llevar a cabo satisfactoriamente el proceso de creación del actor. Esta herramientas son:

- Sistema Operativo. El sistema operativo utilizado en este proyecto es Windows 2000, el cual fue escogido debido a que es uno de los sistemas en que se puede instalar la herramienta Ptolemy II, y a su interfaz gráfica amigable, la cual permite que su manejo sea mucho más fácil.

- Ptolemy II. Esta es la herramienta en la cual se centra el proyecto.

- Java Run Time Environment (Java 2 SDK Standard Edition ver. 1.3.0_02. Esta herramienta es utilizada como soporte para Ptolemy II y para la programación de los nuevos actores, los cuales deben ser escritos en el lenguaje de programación Java.

- Cygwin. Esta es una herramienta de soporte para Ptolemy II.

- Rational Rose. Esta herramienta se utiliza para realizar el modelo del actor que va a ser creado.

5.2.3.1.2. Etapa 1. Identificación de funciones

1. Establecimiento de la función a diseñar

Después de revisar el diagrama de bloques del Codificador G.711, e identificar las funciones que se deben implementar, el experto en Ptolemy II debe, según su conocimiento de la herramienta, identificar qué funciones ya están implementadas en los actores y cuáles deben ser generadas con la creación de nuevos actores. En este proyecto se encontró que la función de cuantificación logarítmica no estaba implementada por ningún actor en la herramienta, así que se debe crear un actor que realice esta función.

2. Identificación de los tipos datos que va a manejar el actor

Una vez encontrada la función a implementar, se debe definir el tipo de datos con los que va a trabajar el nuevo actor, estos son datos de entrada y datos de salida. Debido a que un cuantificador logarítmico recibe señales que pueden ser continuas, abarcan números reales, y las lleva hacia un nivel especificado por el usuario, estos niveles pueden ser números reales, el tipo de datos a utilizar en este nuevo actor serán los números reales (tipo double en Ptolemy II), tanto para la entrada como para la salida.

2.1. Datos de Entrada

- Tipo de datos de entrada: double.

2.2. Datos de Salida

- Tipo de datos de salida: double.

5.2.3.1.3. Etapa 2. Diseño del actor

La siguiente etapa define la estructura del actor, y se realiza un modelo de este para su posterior codificación.

1. Identificación de la clase básica.

Debido a que la función que va a implementar el actor cuantificador logarítmico es una función de transformación, se clasifica en la categoría A de acuerdo a la metodología propuesta, la clase básica de la cual hereda sus atributos es la clase Transformer.

2. Identificación de las características de la clase.

Los parámetros son aquellas propiedades del actor, como puertos y variables, sobre las cuales los métodos actúan.

2.1. Diseño de Atributos

2.1.1. Diseño de Puertos.

El actor Cuantificador Logarítmico implementará un puerto de entrada y uno de salida, que manejan datos tipo double (números reales). Estos puertos son instancias de la clase TypedIOPort, la cual se hereda desde la clase Transformer, que es la clase básica del actor.

Los métodos que utiliza el actor para enviar y recibir señales, los cuales son get() y send(), son heredados de TypedIOPort. El contenedor del puerto, al igual que su nombre estàn definidos también por la clase básica del puerto TypedIOPort. El nombre del puerto es por defecto input para un puerto de entrada, y output para un puerto de salida.

Los puertos con los que trabaja el nuevo actor son puertos simples, es decir solo manejan un canal de comunicación, este es el estado por defecto.

2.1.2 Parámetros del actor.

Los parámetros que va a implementar el actor Cuantificador Logarítmico son niveles y umbrales, los cuales son las variables necesarias para realizar un proceso de cuantificación, de acuerdo a un valor de entrada, este se compara con los valores de los umbrales y de allí se obtiene el nivel de cuantificación.

Los parámetros niveles y umbrales, son arreglos de reales (array doubles), por lo tanto su tipo se debe restringir a este tipo, y el usuario no podrá cambiar este tipo ya que de lo contrario generará un error.

Estos dos parámetros pueden ser vistos y modificados por el usuario desde la interfaz gráfica de la herramienta Ptolemy II, por lo tanto estos deben heredar los atributos y métodos de la clase Parameter, la cual permite que esto sea posible.

2.2 Identificación de los métodos que va a implementar el actor.

2.2.1 Métodos públicos.

Debido a la funcionalidad del actor que esta siendo generado se debe implementar el método público fire(), el cual es el encargado de realizar las operaciones de captura de datos de entrada y envío de datos de salida a través de los puertos del actor.

2.2.2 Métodos privados.

Estos son los métodos propios del actor y los encargados de realizar la funcionalidad especifica de este. Para el actor Cuantificador logarítmico se tiene un método privado, este método arroja un resultado. El método que implementa este actor es un método de asignación, ya que de acuerdo a un nivel de entrada le asigna a una variable el nivel que debe entregar. Con cada valor de entrada se tiene un nivel de salida diferente.

3. Identificación de Reusabilidad.

Para evitar la duplicación de código, el actor Cuantificador Logarítmico hereda clases para implementar los puertos, parámetros y métodos que utiliza. Estas clases que hereda son clases básicas de la herramienta Ptolemy II, que se encuentran en los diferentes paquetes que conforman la herramienta.

Los subpaquetes donde se encuentran los actores polimorficos se encuentran en el paquete actor. Para este caso el nuevo actor se encuentra en el subpaquete ptolemy.actor.lib.
Debido a las propiedades que se han mencionado anteriormente, de puertos, parámetros, tipos de datos; el nuevo actor deberá importar otros subpaquetes en los que se encuentran las clases que implementan estas propiedades. Estos paquetes son:

- ptolemy.actor
- ptolemy.kernel.CompositeEntity
- ptolemy.kernel.util
- ptolemy.data
- ptolemy.data.type.ArrayType
- ptolemy.data.type.BaseType
- ptolemy.data.expr.Parameter

4. Modelo del Actor cuantificador Logarítmico

De acuerdo con la estructuración que se ha hecho del actor, se tiene el siguiente modelo del mismo. El modelo está hecho en lenguaje UML, y muestra los atributos y métodos del actor, así como las clases de las cuales hereda estos atributos y sus métodos. El modelo del actor cuantificador logarítmico se muestra en la figura 5.6.

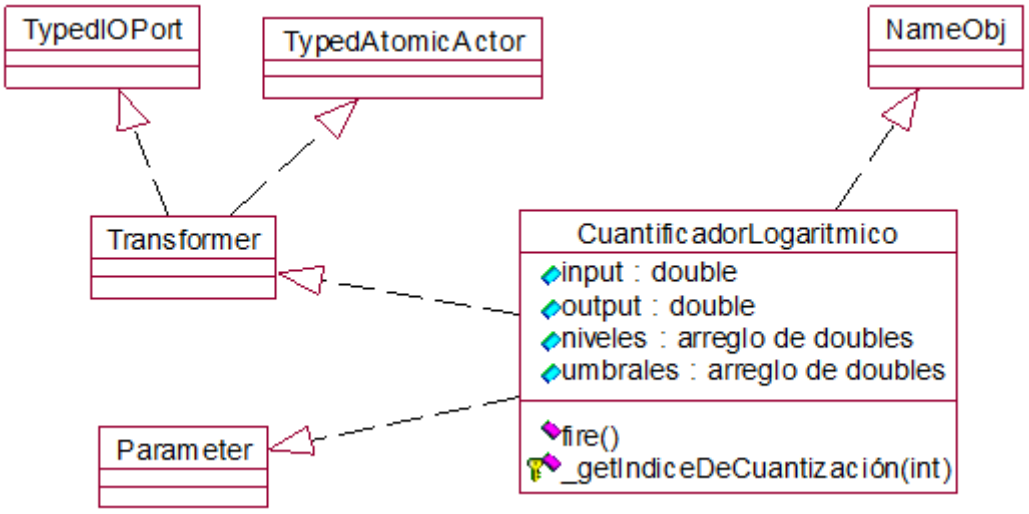


Figura 5.6. Modelo del actor cuantificador logarítmico.

5.2.3.1.4. Validación del actor como entidad funcional

En la última etapa del proceso de creación del actor se realiza la codificación del mismo de acuerdo al modelo del mismo que se tiene de la etapa previa y se realizan las pruebas mencionadas en la metodología.

1. Codificación del actor

Esta actividad corresponde al proceso de codificación del actor y su inclusión en la herramienta.

Código.

El código del actor generado se encuentra en el Anexo 4.

2. Inclusión del actor Cuantificador logarítmico en la herramienta Ptolemy II.

El nuevo actor, debido al tipo de función que realiza se incluyó en la librería de actores matemáticos, para incluirlo en la interfaz gráfica consulte el manual del administrador.

Una vez el actor está en la herramienta, está listo para trabajar con los demás actores en cualquier modelo que se realice.

3. Elaboración del plan de pruebas.

Para el nuevo actor Cuantificador Logarítmico se deben crear dos pruebas, una que compruebe la funcionalidad del actor, es decir que demuestre que el actor esta realizando la función para la cual fue creado, y otra prueba de integración con Ptolemy II, la cual debe dar como resultado una buena adaptación del actor creado a todos los componentes de Ptolemy II.

A continuación se describen las dos pruebas que se crearon para la validación del actor.

4. Prueba del actor actor como entidad funcional.

El actor es un cuantificador logarítmico que recibe una señal de entrada y la compara con unos umbrales para obtener el nivel de cuantificación de acuerdo a estos.

El actor se prueba mediante un modelo sencillo mostrado en la figura 5.7, en el que se tiene una entrada de una constante al actor cuantificador logarítmico, al cual se le han definido unos niveles de cuantificación y unos umbrales para definir cual es el nivel de cuantificación que entrega a la salida, y la salida va a un despliegue de datos que muestra la salida del actor.

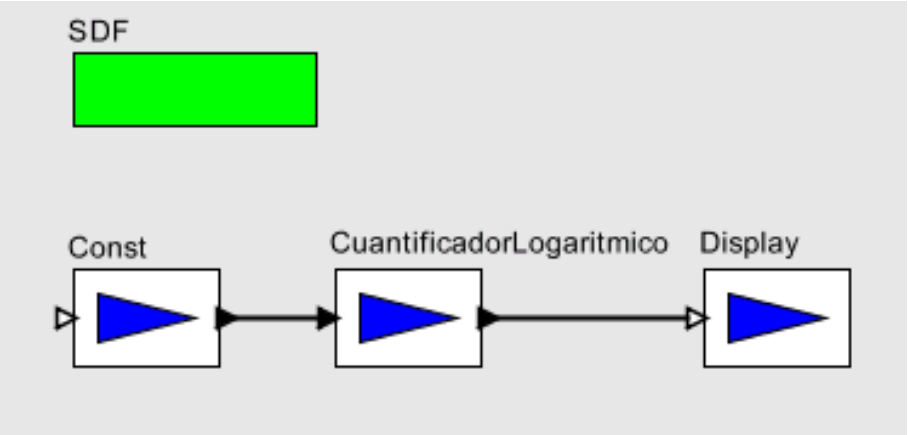


Figura 5.7 Modelo de prueba.

A continuación se muestran dos tablas con los valores de niveles y umbrales escogidos para la prueba. El número de niveles debe ser mayor en uno al número de umbrales. En la figura 5.8 se ve el campo de edición de parámetros del actor cuantificador logarítmico.

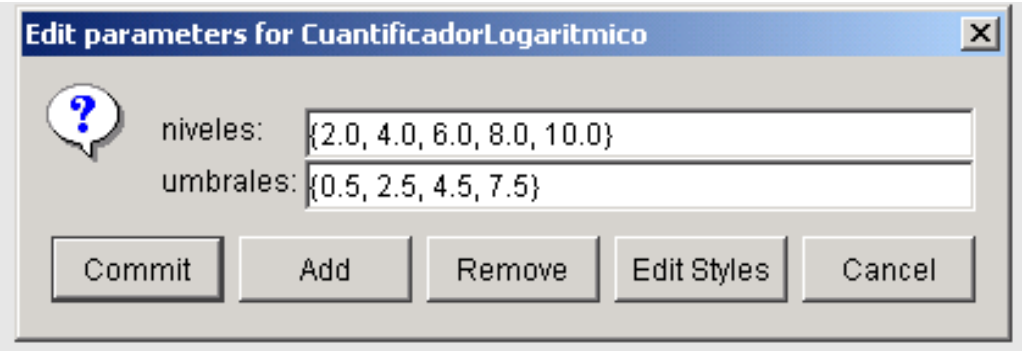


Figura 5.8. Edición de los parámetros niveles y umbrales.

# nivel	1	2	3	4	5
Niveles	2.0	4.0	6.0	8.0	10.0

Tabla 1.2.

# umbral	1	2	3	4
Umbrales	0.5	2.5	4.5	7.5

Tabla 1.3.

Se realizó la prueba con unos valores de entrada y previamente se predijo el comportamiento del actor ante estas entradas según su comportamiento. Estos valores se muestran en la columna 1 y 2 de la tabla 5.7 La columna 3 muestra el resultado obtenido.

Entrada	***Valor esperado***	***Valor Obtenido***
-2.0	2.0	2.0
0.5	2.0	2.0
3.0	4.0	4.0
4.5	6.0	6.0
8.0	10.0	10.0

Tabla 1.4.

5. Integración del actor con la herramienta Ptolemy II.

Las pruebas de integración del actor, deben demostrar si el nuevo actor funciona correctamente con los demás componentes de la herramienta Ptolemy II, como por ejemplos con los diferentes directores que implementan los dominios en que se pueden implementar los sistemas.

Estas pruebas de integración se pueden llevar a cabo realizando diferentes modelos con el nuevo actor, en el cual este interactúe con diferentes actores dentro del modelo y que pueda trabajar en los diferentes dominios de Ptolemy II, es decir se trabaje con diferentes directores.

Para este caso la prueba de integración de el nuevo actor cuantificador logarítmico se hará con el modelo del Codificador G.711, en el cual se tiene un modelo compuesto por diferentes actores los cuales deben interactuar con el actor cuantificador logarítmico. El resultado que se espera de esta prueba es que no se

generen errores de interacción entre el nuevo actor y los demás actores y que este pueda trabajar bajo el dominio de tiempo continuo. La figura 5.9 muestra el modelo creado para llevar a cabo la prueba de integración.

La simulación de este modelo dio como resultado el buen comportamiento del actor cuantificador logarítmico con los demás actores que forman parte del modelo y también se pudo comprobar que este funciona correctamente bajo el dominio del director CT. El resultado de la simulación se muestra en la figura 5.10.

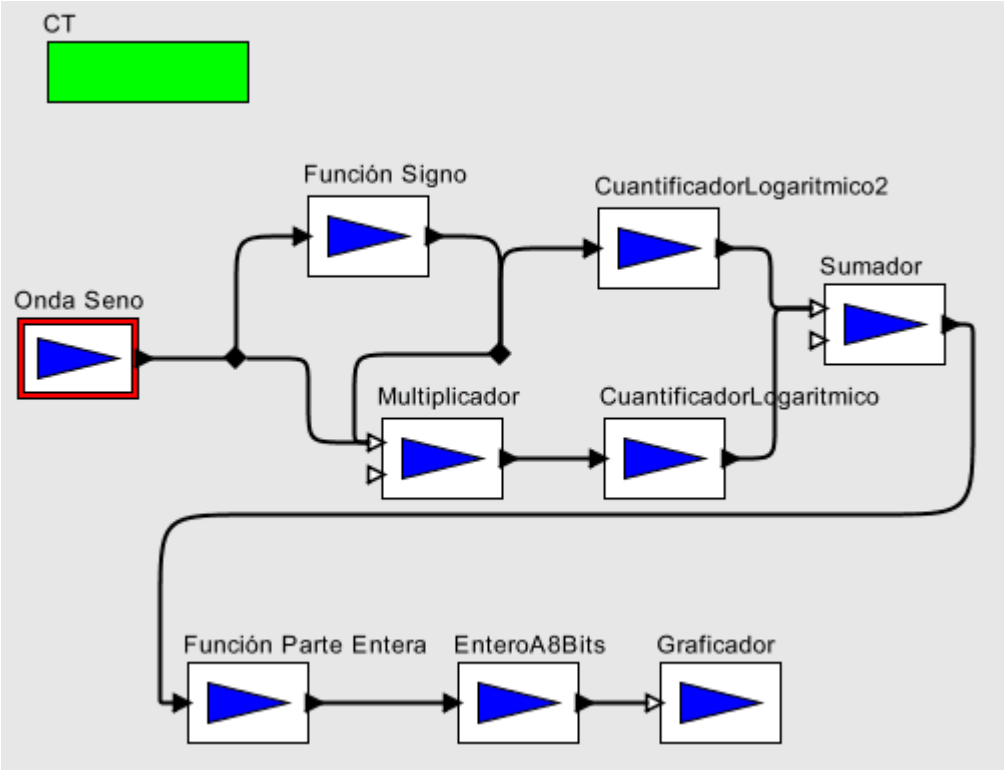


Figura 5.9. Modelo de prueba de integración.

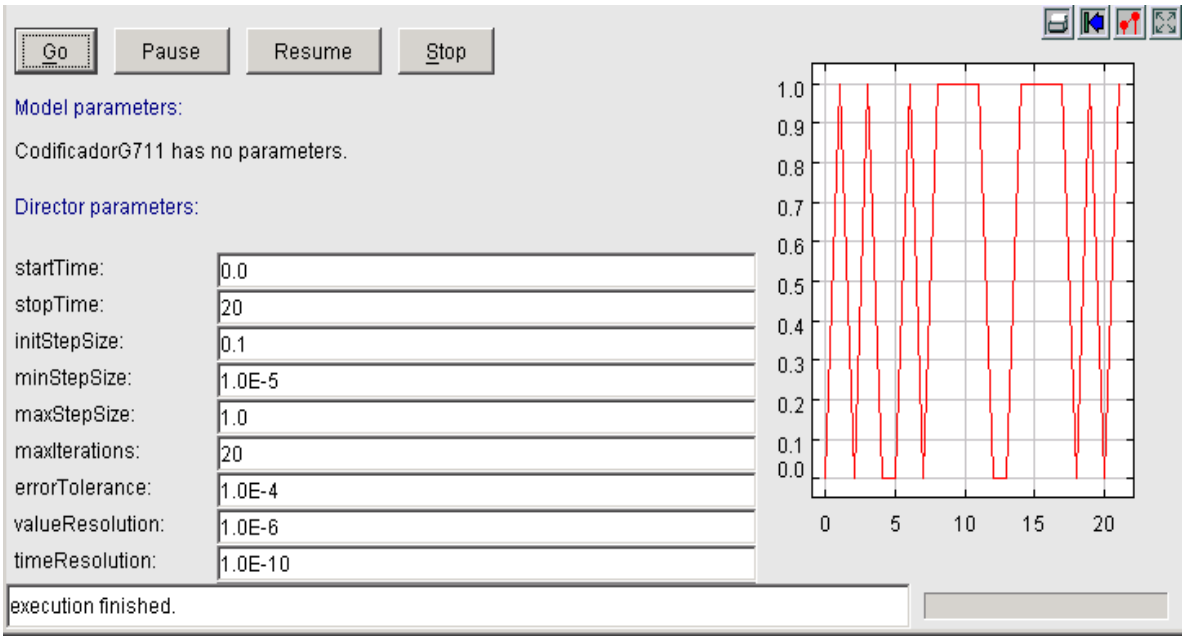


Figura 5.10. Simulación del modelo de prueba de integración.

5.2.3.2. CREACIÓN DEL MODELO DEL CODIFICADOR G.711 EN PTOLEMY II

Una vez creado el actor con la funcionalidad que no estaba implementada en ninguno de los actores de la herramienta Ptolemy II, se crea el modelo, de acuerdo a la figura 5.5. El modelo creado en la herramienta Ptolemy se muestra en la figura 5.11.

La gráfica siguiente implementa el diagrama en bloques del Codificador G.711, como un modelo en Ptolemy II, donde el puerto Entrada recibe la señal de voz y el puerto Salida entrada la señal codificada.

Este modelo es la composición del actor compuesto Codificador G.711 (ver manual de usuario, creación de actores compuestos), el cual es el que actuará directamente sobre las señales de voz, el icono del actor compuesto Codificador G.711 se muestra en la figura 5.12.

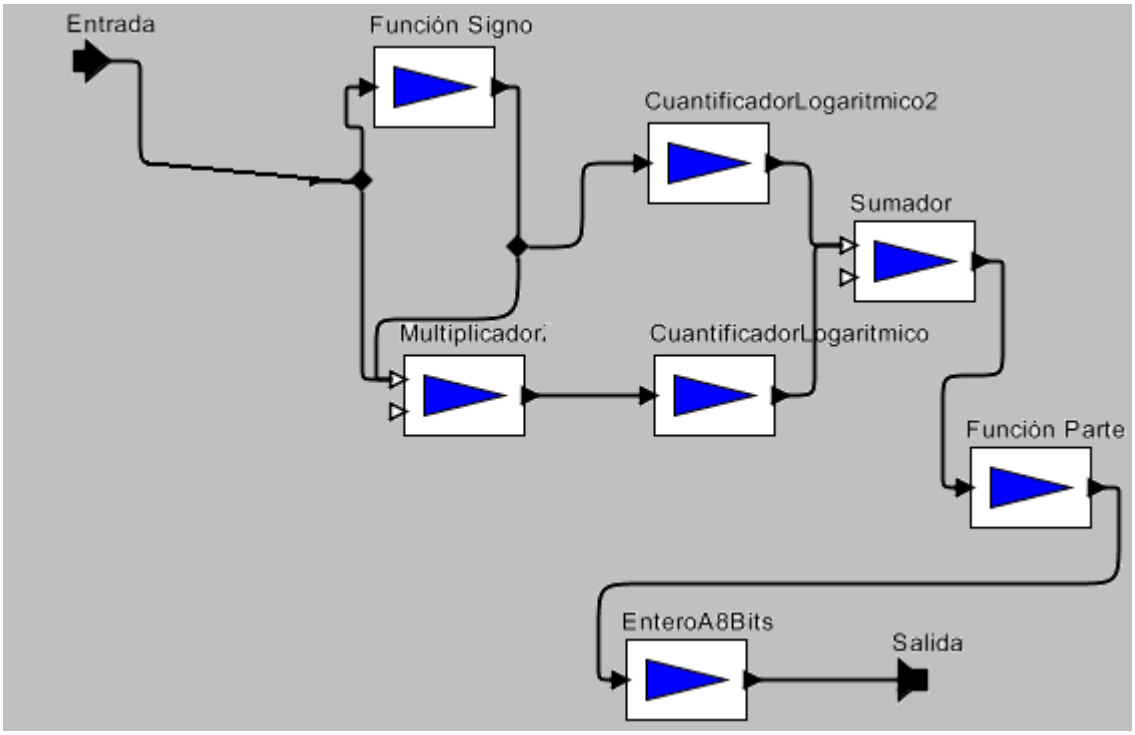


Figura 5.11. Modelo del Codificador G.711 en Ptolemy II.

Figura 5.12. Actor compuesto Codificador G.711

5.2.3.3. SIMULACIÓN DEL CODIFICADOR G.711 EN PTOLEMY II

La simulación del Codificador G.711, se realiza por medio de una entrada de audio, la cual puede ser un archivo de audio (archivo wav), el cual entra al codificador. La salida debe ser binaria y es vista por un graficador. Este modelo se implementa en el dominio de tiempo continuo, en el cual se puede especificar la cantidad de tiempo que dura la simulación, dependiendo de la duración del archivo de audio. La figura 5.13 muestra el modelo del Codificador G.711.

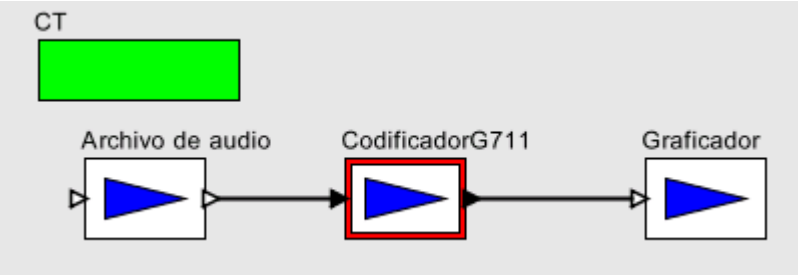


Figura 5.13. Modelo del Codificador G.711.

Para examinar con más detalle el proceso de codificación que implementa el G.711, la entrada a este puede ser una señal seno, con la cual es mucho más fácil comparar la salida con respecto a la entrada.

6. CONCLUSIONES Y RECOMENDACIONES

- La herramienta Ptolemy II v.1.0.1 presenta un avance importante respecto a la versión Clásica de la misma. Basándose en un trabajo previo con Ptolemy clásico se pudieron establecer comparaciones entre estas dos versiones. La versión clásica está implementada en lenguaje C++, esta herramienta es monolítica, el procesamiento de los componentes es secuencial, pero es una herramienta totalmente madura. Ptolemy II está implementada en java, la nueva versión tiene una arquitectura del software compuesta por paquetes modulares, el procesamiento es multihilos, es decir, concurrente, tiene muchas limitaciones en algunos de sus paquetes, sobre todo el paquete de actores es bastante limitado en la parte de actores de la librería de comunicaciones aunque está en permanente mejoramiento. Estas características le dan nuevas habilidades entre las que están el trabajo con redes, el manejo de conceptos de lenguajes de programación como semántica, teoría de tipos de datos y teorías de concurrencia, entre otros.

- Con el trabajo desarrollado, se pudo comprobar su uso exitoso en el modelamiento de sistemas telemáticos, tales como Sistemas Distribuidos y Aplicaciones de Red. El trabajo intensivo con la herramienta para su aplicación en Sistemas Telemáticos se está realizando en diversas Universidades, tales como la Universidad de California, adecuando la herramienta para el trabajo en redes y sistemas distribuidos{1}; en la Universidad de Dresden (Alemania) se esta trabajando en un Simulador de redes inalámbricas para redes celulares de tercera generación basado en Ptolemy II {2}.

- El trabajo de investigación de la aplicación de la herramienta también se hace a nivel industrial, múltiples empresas trabajan en cooperación con las Universidades como es el caso de Agilent Technologies[10] con su trabajo, Diseño y Verificación para RFICs (Chipsets de Circuito Integrado y Radio Frecuencia) Inalámbricos de tercera generación {3}.

- Hasta este momento la herramienta es útil en las etapas de diseño, modelamiento dentro del desarrollo de un Sistema Telemático, así como la simulación del mismo. Las etapas de captura de requerimientos, análisis e implementación no se descartan como futuras capacidades de la misma por el permanente desarrollo del proyecto y el proceso de maduración que sufre. Actualmente se está trabajando sobre la capacidad de generar código de los modelos simulados.

- Del conocimiento de la arquitectura de la herramienta, los paquetes que la constituyen y su interacción se pudo concluir que su modularidad proporciona la flexibilidad necesaria para la introducción de nuevos paquetes en el caso que se de una nueva funcionalidad a la herramienta, por ejemplo para la interacción con bases de datos o que se vaya a manejar nuevos modelos de computación.

- La elaboración de una metodología para la creación de actores, facilita el proceso de creación de nuevos componentes para la herramienta, y se constituye en un aporte valioso para la herramienta y una novedad por cuanto no se tenía una metodología que comprendiese las etapas básicas para la identificación de todos los elementos constitutivos de un actor. Esta metodología fue exitosamente aplicada en la creación de un actor el cual se introdujo a la librería de actores de la herramienta.

[10] http://agilent.com

- Como producto de la apropiación de la herramienta se generaron dos manuales, el manual de usuario y el manual de administración que facilitan la utilización de la herramienta tanto en la parte de usuarios básicos a nivel de interfaz gráfica, hasta la parte de gestión para administradores de Ptolemy II.

- Se recomienda dar continuidad al proyecto para lograr que Ptolemy II sea utilizado en la Facultad de Ingeniería Electrónica y Telecomunicaciones como una herramienta para la construcción de Sistemas Telemáticos. Este trabajo se ve como la primera fase de un proyecto, del cual se proponen las siguientes fases para lograr la consolidación de la herramienta:

  Fase 2: Elaboración de una metodología para el modelamiento y diseño de Sistemas Telemáticos con Ptolemy II.
  Fase 3: Modelamiento de un sistema Telemático especifico con Ptolemy II.
  Fase 4: Elaboración de una guía que incorpore la utilización de Ptolemy II en la construcción de Sistemas Telemáticos.

  El resultado de estas cuatro fases podrá consolidar a Ptolemy II como una herramienta robusta que podrá ser adoptada por la comunidad de la Facultad de Ingeniería Electrónica y Telecomunicaciones.

- Durante el desarrollo del presente trabajo de grado se pudo comprobar que la herramienta tiene los elementos básicos para la creación de laboratorios virtuales; con base en este trabajo de grado y la continuación de las dos fases siguientes, podría llegarse a la implementación de laboratorios virtuales para la facultad en las materias: Laboratorio 1, 2 y 3 de Sistemas de Telecomunicaciones. Entre los sistemas que puede manejar la herramienta se encuentran Sistemas empotrados, de tiempo real, Sistemas optoelectrónicos entre otros.

- Se recomienda mantener contacto con el equipo desarrollador del proyecto Ptolemy de la Universidad de California, quien ha mostrado interés en difundir su trabajo y resolver las inquietudes sobre la herramienta, además, el proyecto Ptolemy se actualiza permanentemente y se genera nueva documentación constantemente. Por otra parte el equipo de trabajo está por lanzar una nueva versión de la herramienta.

- Se debe seguir con la exploración de la herramienta ya que tiene muchos aspectos que aun no se han examinado a fondo ya que están en etapa desarrollo e investigación por parte del equipo Ptolemy, cuyo estudio puede aportar nuevos elementos para el desarrollo de Sistemas Telemáticos en la Facultad de Ingeniería Electrónica. Estos aspectos comprenden:

    - Análisis del comportamiento de sistemas con UML y Ptolemy {4}, esta parte fortalecerá a Ptolemy II las etapas de captura de requerimientos y análisis este trabajo es realizado por el grupo Thales Optronique[11].

    - Trabajo con CORBA para sistemas distribuidos{1}, este estudio se realiza en la Universidad de California por parte del equipo principal.

[11] http://thomson-csf.com

7. BIBLIOGRAFIA

1. BISHOP, Judy. Java Fundamentos de Programación. Ed. Addison-Wesley, Ed. 2, Madrid España 1999.

2. JACOBSON, Ivar. BOOCH, Grady. RUMBAUGH, James. El Proceso Unificado de Desarrollo de Software. Ed. Addison-Wesley, Ed. 1, Madrid España 1999.

3. SERRANO, Carlos E. Modelo de Referencia para Desarrollo de Proyectos Versión 1.1. Universidad del Cauca. 1997.

4. LEE, Edward. HYLANDS, Christopher. LIU, Jie. Ptolemy II Heterogeneous Concurrent Modeling and Desing in Java. Versión 1.0. Universidad de California. Berkekely. 2001.

5. {1}. Jie Liu, Xiajun Liu, Sonia Sachs. ePtolemy: Distributed Modeling and Design in Ptolemy II. Ptolemy Miniconference Document, Marzo 22 y 23 de 2001. Grupo Ptolemy. Universidad de California. Berkeley USA.

6. {2}. W. Rave, J. Voigt, T. Köhler, G.Fettweis. A Wireless Network Simulator for 3rd Generation Cellular Networks based on Ptolemy. Ptolemy miniconference Document, Marzo 22 y 23 de 2001. Universidad de Dresden. Alemania.

7. {3}. Agilent Technologies. Design and Verification for 3G Wireless RFICs. . Ptolemy miniconference Document, Marzo 22 y 23 de 2001.
8. {4}. Xavier Warzee, Jean-Charles Causse. System Behaviour Análisis with UML and Ptolemy. . Ptolemy miniconference Document, Marzo 22 y 23 de 2001. Grupo Thomsom CSF.

9. Pagina web: http://ptolemy.eecs.berkeley.edu/

8. REFERENCIAS

[1] http://www.gdl.uag.mx/66/portada.htm
Desarrollado por: Lic. Yadira Esparza Rodríguez

[2] http://www.geocities.com/SiliconValley/Bit/6238/diap3.htm
Fuente: Revista, Tecnología de Punta. Autor: Ing. Víctor Castañeda Guzmán

[3] http://www.map.es/csi/silice/Herayudes2.html

[4] El Paradigma Orientado a Objetos. Sinan Si Alhir. Octubre 6 de 1999.
Traducido por Camilo Torres Días.

GLOSARIO

A

Actor: Es una entidad ejecutable.

Actor Atómico: Es un actor primitivo. No es un actor compuesto.

Actor Compuesto: Es un actor que esta internamente compuesto por otros actores y relaciones.

Actor Compuesto Transparente: Es un actor compuesto sin un director local.

Actor Compuesto Opaco: Es un actor compuesto con un director local. Este actor aparece fuera del dominio como atómico, pero internamente esta compuesto de una interacción de otros actores.

Anchura de un puerto: La suma de los anchos de las relaciones enlazadas a un determinado puerto; es cero si no hay ninguna.

C

Canal: Un camino desde un puerto de salida a un puerto de entrada.

Contenedor: Un objeto que lógicamente contiene a otros. Un objeto de Ptolemy II puede tener a más de un contenedor.

Conexión: Una trayectoria desde puerto a otro a través de relaciones y posiblemente puertos transparentes. Una conexión consiste de una o más relaciones y dos o más enlaces.

D

Director: Es un objeto que controla la ejecución de un modelo o de una entidad opaca compuesta director de acuerdo al modelo de computación.

Director Local: Desde la perspectiva de un actor dentro de un actor compuesto opaco, es el director del contenedor del actor compuesto opaco.
Dominio: Es una implementación de un modelo de computación en Ptolemy II.

E

Entidad: Es un nodo en una gráfica agrupada de Ptolemy II.
Ejecución: Es una invocación del método initialize(), seguida por cualquier número de iteraciones, seguida por una invocación del método wrapup()

G

Gráficas Agrupadas: Es una gráfica con jerarquía. Las topologías de Ptolemy II son gráficas agrupadas.

M

Modelo: Es una aplicación completa de Ptolemy II.
Modelo de Computación: Son la reglas que gobiernan la interacción, comunicación y flujo de control de un conjunto de componentes.
Multipuerto: Es un puerto que puede enviar o recibir señales sobre más de un canal.

O

Opaco: Para una entidad o un puerto compuestos, es un atributo que indica que el interior no debe ser visible desde el exterior.

P

Paquete: Es una colección de clases que forman una unidad lógica y ocupa un directorio en el árbol del código fuente.
Parámetro: Es un atributo con un valor.
Puerto: Es una interfase nombrable de una entidad a la cual se pueden hacer conexiones.
Puerto Transparente: El puerto de una entidad transparente compuesta.

Ptolemy II: Es un sistema software construido en Java para la construcción y ejecución de modelos concurrentes.

Proyecto Ptolemy: Es un proyecto de investigación de la Universidad de California en Berkeley que investiga modelamiento, simulación y diseño de sistemas empotrados, sistemas de red y sistemas concurrentes.

S

Señal: Es una unidad de datos que es comunicada por los actores.

Sintaxis Abstracta: Es una organización de datos conceptual. Medio de relaciones que puede transportar un flujo único de señales.

Subpaquete: Es un paquete que esta lógicamente relacionado a un paquete padre y ocupa un subdirectorio dentro del paquete padre en el árbol del código fuente.

T

Token: Es una unidad de datos que es comunicada por los actores.

Topología: Define la estructura de interconexión entre entidades (por medio de relaciones) en un modelo Ptolemy II. Ver gráficas agrupadas.

Transparente: Para una entidad o puerto no opaco.

Transversales deep: Son transversales de una gráfica agrupada que ven a través de fronteras agrupadas transparentes.

LISTA DE ACRÓNIMOS

API: Interfaz de Programa de Aplicación

CSP: Procesos Secuenciales de Comunicación

CT: Tiempo Continuo

DE: Eventos Discretos

DDE: Eventos Discretos Distribuidos

FIFO: Primero en Entrar Primero en Salir

GUI: Interfaz Gráfica de Usuario

JDK: Kit de Desarrollo de Java

JSD: Desarrollo Estructurado de Jackson

MEMS: Sistemas Mecánicos Micro Electrónicos

MRDP: Manual de Referencia para el Desarrollo de Proyectos

OO: Orientado A Objetos

ODE: Ecuación Diferencial Ordinaria

OMT: Técnica de Modelado de Objetos

PN: Redes de Procesos

RFIC: Chipsets de Circuito Integrado y Radio Frecuencia

SA/SD: Análisis estructurado / Diseño estructurado

SDF: Flujo de Datos Síncrono

UI: Interfaz de Usuario

UML: Lenguaje de Modelamiento Unificado

4GL: Lenguajes de Cuarta Generación