INFRAESTRUCTURA DE EVENTOS PARA COOPERACIÓN DE APLICACIONES

Documento para optar al título de Maestria de Ingeniería de Sistemas

Nicolás López Giraldo

Asesor:

Rubby Casallas

Julio de 2005

Departamento de Ingeniería de Sistemas y Computación

Universidad de Los Andes

1 Introducción

El desarrollo de software globalizado representa una oportunidad para fortalecer la industria de software en Colombia. Algunas ventajas del desarrollo de software globalizado incluyen disminución de costos de mano de obra especializada y producción en fábricas de funcionamiento las 24 horas [3]. La reducción de costos de mano de obra especializada se da en los casos en que los costos por desplazar al individuo no son incurridos por la posibilidad de trabajar geográficamente dispersos.

Sin embargo, los procesos GSD presentan una serie de dificultades y problemas. Los procesos GSD deben tener mecanismos para ayudar a los desarrolladores a controlar las actividades, sincronizar el trabajo, comunicar resultados, considerar las personas y sus roles, y administrar la información y los resultados parciales. El monitoreo y administración del proceso introduce más dificultades [14].

Aunque los procesos de desarrollo como diseño, implementación y pruebas de software se ajusten a ambientes de trabajo de tipo GSD, se requieren herramientas de apoyo a dichos procesos, pues al eliminar el componente de interacción cara a cara, la comunicación y coordinación entre las personas debe ser llevada a cabo mediante mecanismos tales como teléfono, e-mail y mensajería instantánea [17]. Si bien estos medios de comunicación son parte de una infraestructura de soporte que facilita la comunicación de forma puntual, no hacen parte de herramientas informáticas especializadas en el soporte de procesos GSD [16].

Adicionalmente, aún suponiendo que un grupo de trabajo distribuido cuente con software de apoyo a los procesos de desarrollo, estas herramientas suelen ser diferentes y su interacción e intercambio de información ocasionan un problema más para solucionar. Se debe integrar en una visión general lo que sucede en los procesos GSD, obteniendo información de herramientas heterogéneas a las que, en general, no es posible o deseable hacerles cambios en su funcionalidad por razones prácticas y de costos [7]. Estas herramientas comparten conceptos de dominio similares; es necesario mantener consistencia de estos conceptos para efectivamente apoyar los procesos de negocio. [11].

Integrar estas herramientas es un reto pues están desarrolladas en diferentes tecnologías y con representaciones de datos en ocasiones incompatibles, dando como resultado que el costo de solucionar esta problemática sea mayor que el problema a solucionar [7]. Una agravante de esta situación es la distribución a escala Internet necesaria para los procesos GSD. Por estas razones el requerimiento principal de una plataforma de integración distribuida de escala de Internet para procesos GSD es la integración laxa y bajo acoplamiento de las aplicaciones; con la restricción de que es necesario que la infraestructura reduzca las modificaciones en las herramientas.

Este trabajo presenta una infraestructura de integración de aplicaciones basada en eventos distribuidos. Se introduce una nueva aproximación para el diseño de los modelos de notificación y observación del framework de diseño de sistemas de notificación basados en eventos presentado en [12]. El propósito central de la aproximación es reducir las modificaciones necesarias a las aplicaciones existentes para integrar y describir su cooperación a través de la infraestructura. Para conseguir esto, se definió la noción de un Proxy de Cooperación que intermedia la interacción y encapsula la definición de la cooperación entre el sistema de eventos y las aplicaciones.

La cooperación de las aplicaciones, que define la integración, presentada en este trabajo, es descrita en términos de:

- Eventos observados relacionados a modificaciones en el estado de una aplicación de los conceptos de dominio sobrepuestos.

- Condiciones y la reacción a eventos notificados a aplicaciones que comparten estos conceptos de dominio

La cooperación entre las aplicaciones en la infraestructura presentada es realizada a través de reglas de Evento Condición Acción (ECA), observación externa y procesamiento de eventos en las aplicaciones. La información recolectada de las aplicaciones es observada de manera externa y procesada para producir eventos. Las reglas ECA describen la reacción de las aplicaciones a los eventos notificados o generados por las observaciones.

La infraestructura presentada usa una estrategia que mezcla al uso de eventos lcomo mecanismo asincrónico de comunicación la comunicación con Web Services. El uso de Web Services ofrece una comunicación request/reply y es fácilmente implementado por aplicaciones legado.

El proceso de definición de la infraestructura incluyó el estudio en una serie de temas relacionados y de soporte al contexto en el que se elaboró el proyecto. Inicialmente se trabajo sobre los modelos de coordinación, lenguajes y sistemas de Workflow. En una fase temprana del proyecto se decidió iniciar el desarrollo de un sistema de de notificación basado en eventos, para esto se revisó el estado del arte en este tema. Estos trabajos nos llevaron a profundizar en varias tecnologías y modelos que apoyaran los requerimientos que identificamos como necesarios para la integración, éstas incluyen los sistemas de reglas Evento Condición Acción (ECA). Finalmente se revisaron las tecnologías de soporte a la infraestructura como tecnologías de comunicación, administración y monitoreo e intercepción de servicios a través de aspectos.

Como parte de la infraestructura se diseñó e implementó una serie de componentes. Primero, un sistema de notificación por eventos, llamado Middleware de Eventos Distribuidos (DEM de sus siglas en inglés) que ofrece la funcionalidad básica de un sistema publicador/subscriptor. Segundo, un modelo de notificación y observación de eventos, que usa el DEM, implementado por un componente llamado Proxy de Cooperación (CP de sus siglas en inglés). Tercero, un esquema de pruebas para la integración de aplicaciones. Finalmente, un sistema de monitoreo y administración de la infraestructura.

El proyecto fue validado en una empresa de software que en la actualidad afronta los problemas del desarrollo de software globalizado. La validación fue hecha dentro de contexto de un proyecto para definir e implementar un proceso de pruebas en esta empresa; el proyecto incluye la construcción de nuevas herramientas y su integración con herramientas legado. Una vez construida la infraestructura se definieron reglas de cooperación para las herramientas del proceso y se implementaron las reglas para su integración con la infraestructura de eventos. Finalmente se desarrolló y probó la integración con la infraestructura en un proyecto piloto. El piloto es ejecutado en el contexto del proyecto “Pruebas en ambientes globalizados” liderado por el Departamento de Ingeniería de Sistemas de la Universidad de los Andes y Heinsohn Software House S.A. y financiado parcialmente por COLCIENCIAS.

Este documento está organizado de la siguiente manera: el siguiente capítulo introduce los objetivos del proyecto. El capitulo 3 y 4 presentan el estado del arte de los sistemas de notificación por eventos y de otras tecnologías útiles para el proyecto respectivamente. El capitulo 5 introduce el modelo de Eleggua y un escenario de ejemplo. El capítulo 6 presenta el modelo de implementación, mientras el capítulo 7 presenta los detalles de implementación. El capítulo 8 muestra el proceso y los resultados de la validación de la infraestructura en un contexto real. Finalmente, el capítulo 9 concluye el documento y comenta los trabajos futuros.

2 Objetivos

2.1 General

Definir un modelo de integración de aplicaciones basado en eventos distribuidos que permita la ejecución de modelos de cooperación de aplicaciones en un contexto de dispersión geográfica. El modelo debe concretarse en una infraestructura que solucione algunos de los problemas que presenta el desarrollo de software globalizado. Adicionalmente, la infraestructura debe proveer herramientas de administración y monitoreo. La infraestructura y el modelo deben ser probados y validados en un contexto real.

2.2 Objetivos Específicos

2.2.1 Definir un modelo de integración

El modelo debe tener como preocupaciones centrales hacer una integración no intrusita, es decir que sean mínimas las modificaciones necesarias en aplicaciones existentes, y cumplir con los requerimientos de bajo acoplamiento e integración laxa. El modelo debe permitir que la descripción de la integración sea externa a las aplicaciones. A partir del modelo se debe poder definir un lenguaje de alto nivel que permita abstraer la lógica de las reglas.

2.2.2 Implementar la infraestructura.

El modelo debe concretarse en una infraestructura que implemente la integración de las aplicaciones. Esta infraestructura debe proveer los servicios básicos de un servicio de notificación por eventos basado en publicación /suscripción; adicionalmente, debe permitir la ejecución de las reglas de integración definidas por el modelo. La implementación de la infraestructura debe permitir probar algunas tecnologías nuevas de comunicación, administración y aspectos entre otras. Estas tecnologías deben definirse e investigarse.

2.2.3 Implementar un componente de monitoreo y administración

El componente debe permitir monitorear el estado de los componentes, configurar y modificar los componentes del sistema, y administrar y controlar la integración de las aplicaciones que usan la infraestructura. Adicionalmente es necesario que las reglas de integración puedan ser administradas y monitoreadas. Este componente es necesario debido a la alta relevancia que estos factores tienen en el contexto del proyecto.

2.2.4 Probar y validar la infraestructura en el contexto del proceso.

Elaborar pruebas unitarias, de sistema, de usuario, deseablemente automatizadas o asistidas. Las pruebas deben permitir la ejecución y validación de las reglas de integración.

La infraestructura y el modelo deben ser validados en un contexto real. Para esto es necesario usar un proceso definido. Dentro del proyecto debe implementarse el proceso de negocio a través de reglas de cooperación. Este proceso debe ser probado en un proyecto piloto. El proyecto piloto permitirá evaluar el desempeño de la infraestructura y validar que la infraestructura sí facilita el

desarrollo de la integración. Adicionalmente se debe evaluar como la infraestructura soluciona algunos de los problemas del desarrollo globalizado.

3 Sistemas de Notificación de Eventos (SNE)

Los sistemas basados en observación, generación y notificación de eventos son un estilo arquitectural ampliamente utilizado para sistemas de bajo acoplamiento en una variedad de dominios como monitoreo de aplicaciones, reconocimiento de aplicaciones y movilidad [13]. Cada domino introduce una serie de requerimientos funcionales y no funcionales. Estos sistemas son usados como framework para herramientas colaborativas [16], son útiles para seguimiento y administración de procesos e ingeniería de software [14].

Las ventajas principales en sistema de notificación por eventos (SNE) para soporte al desarrollo de software globalizado son: flexibilidad, integración laxa y notificaciones a escala de Internet [16]. La flexibilidad es lograda por el desacoplamiento de las fuentes y consumidores de eventos. Esto crea una aproximación “plug and play” donde es simple introducir nuevos componentes al sistema. Un SNE puede ser utilizado para apoyar la integración débil de herramientas de soporte a los procesos de desarrollo, esto ocurre porque cada componente tiene el conocimiento mínimo posible de otros componentes. Estos sistemas permiten la construcción de sistemas distribuidos a escala de Internet.

Los conceptos principales en la interacción con un SNE son generación de eventos, notificación de eventos, despacho de mensajes y suscripción [4], [16]. La generación de eventos es el proceso en el que un componente como entidad autónoma comunica al ambiente cambios en su estado interno; estos componentes son las fuentes de eventos. La notificación de eventos es la observación de un componente de estímulos externos que pueden modificar el estado interno; los componentes que reciben notificaciones son los consumidores de eventos [4]. El despacho de mensajes es el proceso de direccionamiento de eventos a los consumidores que han declarado interés previamente. Un servicio de suscripción permite a los consumidores declarar interés a diferentes tipos de eventos o patrones de eventos usando mecanismos como expresiones regulares o conectores lógicos [16].

En este capítulo se presenta un framework de diseño de SNEs. A partir de éste se describen propuestas de sistemas, plataformas e infraestructuras basadas en eventos. Finalmente, se discute la relación de estos sistemas en el contexto GSD con respecto al modelo presentado.

3.1 Framework de diseño de un Sistema de Notificación de Eventos

El framework de diseño de un SNE propuesto por Rosenblum et al. en [12] puede ser utilizado para caracterizar y comparar el diseño de varios SNE. Este framework está compuesto de siete modelos que cubren la mayoría de los aspectos relevantes del diseño de un sistema de notificación de escala de Internet. Estos son el modelo de objetos, modelo de eventos, modelo de nombramiento, modelo de observación, modelo de tiempo, modelo de notificación y modelo de recursos.

3.1.1 Modelo de Objetos

El modelo de objetos caracteriza los componentes que pueden producir y recibir notificaciones. El aspecto más relevante en este modelo es la definición de los objetos de interés y los invocadores. Un objeto de interés es una entidad o servicio cuya invocación puede producir eventos. El invocador es el objeto que invoca la operación o servicio en el objeto de interés.

3.1.2 Modelo de Eventos

El modelo de eventos incluye una caracterización precisa de los eventos. Un evento es el efecto de la terminación de una invocación en un objeto de interés. Esto implica que los eventos ocurren aún cuando no sean observados. Un evento puede ser caracterizado por la identidad del objeto de interés, la operación invocada, el invocador y la marca de tiempo de producción del evento.

3.1.3 Modelo de Nombramiento

El modelo de nombramiento define mecanismos para identificar eventos, objetos, operaciones y otra información asociada con los eventos. Este modelo debe proveer un lenguaje para identificar de manera única un evento y crear expresiones que son interpretadas como un conjunto de eventos.

3.1.4 Modelo de Observación

El modelo de observación define la observación de eventos y de patrones o secuencias de eventos. Esta observación es lograda a través de objetos observadores. El modelo define varias políticas: La política de observación especifica el mecanismo de observación de objetos. Los mecanismos comunes para esta política son de tipo push o pull. En el mecanismo push los eventos son empujados del objeto que produce el evento al SNE. En el mecanismo pull los eventos son halados o solicitados del SNE al objeto que produce el evento. La política de información incluye qué tipos de patrones pueden ser especificados y cómo son identificados por un objeto observador. La política de particionamiento define cómo son divididas las tareas de observación entre los objetos observadores. Finalmente, la política de filtros especifica cómo la información de un evento es utilizada para seleccionar los eventos para notificar.

3.1.5 Modelo de Tiempo

El modelo de tiempo incluye consideraciones de cómo la latencia de red y el tiempo variable en distribución de eventos afectan la notificación en una red amplia. Define el momento o momentos en que el tiempo está relacionado con las actividades de observación y notificación

3.1.6 Modelo de Notificación

El modelo de notificación caracteriza la comunicación entre los observadores y los receptores interesados. Debe considerar la comunicación bidireccional, primero, la declaración de interés de un receptor y posteriormente, la comunicación de notificaciones. Para la comunicación de notificaciones los mecanismos pueden ser push o pull. En el mecanismo push los eventos son empujados del SNE al objeto notificado. En el mecanismo pull los eventos son halados o solicitados por el objeto notificado al SNE.

3.1.7 Modelo de Recursos

Finalmente, el modelo de recursos define la arquitectura y topología de los recursos distribuidos en una red amplia.

En las secciones siguientes utilizamos el framework para presentar algunos SNE existentes y poder comparar las ventajas y desventajas que cada uno tiene.

3.2 JEDI

JEDI es un servidor de notificación de eventos desarrollado por el departamento de electrónica e Información en el Politécnico de Milano [4]. La infraestructura es usada para implementar un sistema distribuido de administración de flujo de trabajo (WFMS) OPSS (ORCHESTRA Process Support System) [19].

Las nociones centrales de la arquitectura son los objetos activos (AOs), despachador de eventos (ED) y servidores de despacho (DSs). Un AO corresponde al modelo de objetos del sistema; son entidades autónomas con su propio hilo de control. El modelo de eventos define un evento como un conjunto ordenado de cadenas. El interés en eventos es expresado a través de un patrón de eventos, una forma simple de expresión regular. La observación de eventos es lograda a través de un mecanismo push; los AOS notifican explícitamente al ED la ocurrencia de un evento. Para el modelo de tiempo, la infraestructura provee orden causal de eventos, a través del uso de relojes vectoriales.

Los modelos de notificación y de recursos de JEDI están relacionados de la siguiente manera. El ED es un componente lógicamente centralizado pero implementado de manera distribuida por un grupo de servidores de despacho (DS). Este componente se encarga la notificación de eventos con un modelo push/pull. La topología de los DS es jerárquica; un DS propaga las subscripciones hacia su ancestro. Los eventos son propagados al ancestro, a todos los descendientes que están suscritos al evento y a todos los AOs que han declarado interés en el evento directamente al DS. JEDI define un modelo explícito de migración; al migrar un AO la infraestructura almacena suscripciones y eventos por entregar.

JEDI muestra una serie de inconvenientes. El modelo de objetos es vagamente definido que a su vez crea un modelo de observación que no considera varias de las políticas; esto es porque sin un modelo claro de objetos que defina las entidades que se pueden observar, no es posible definir cómo se van a observar estas entidades. El modelo de tiempo puede crear problemas para notificación a nivel de Internet; mantener un reloj vectorial puede ser muy costoso. La topología y mecanismos de propagación permiten el escalamiento de la infraestructura; sin embargo, el fallo de un servidor en este tipo de topologías puede ser crítico.

3.3 SIENA

SIENA, desarrollado en la Universidad de Colorado en Boulder [2], es un SNE de escala de Internet implementado por los autores que definen el framework de diseño de SNEs. Su consideración central está en el diseño del modelo de recursos y el impacto de éste en el desempeño del sistema en una red amplia.

El modelo de objetos no es especificado, los objetos interactúan a través de una interfaz con el sistema. El modelo de eventos es liviano y simple; los eventos son un conjunto de parámetros con tipo, nombre y valor. Los patrones de eventos y filtros de eventos son definidos junto con un lenguaje completo y complejo de expresiones para declarar suscripciones y anuncios (advertisements)

El modelo de observación define un mecanismo simple de push basado en la interface ofrecida, éste se enfoca en políticas de filtro y de patrones y define una semántica rica para estos aspectos. El modelo de tiempo define consideración sobre la latencia entre suscripciones y la notificación de eventos; es decir, definen explícitamente el comportamiento del sistema cuando una suscripción es declarada en el momento en que un evento que cumple esta suscripción es notificado. No existen mecanismos explícitos para el ordenamiento de eventos.

El modelo de notificación es una preocupación central del framework. Varias consideraciones son tomadas por los autores para la propagación de suscripciones y anuncios. El modelo minimiza la replicación de suscripciones y anuncios a través de la definición de una lógica de suscripción que incluye el concepto de cubrimiento. Las suscripciones propagadas son solamente las que no son cubiertas por una suscripción previa. La notificación de eventos está basada en un mecanismo pull por parte de los consumidores, la propagación de notificaciones está basada en algoritmos de camino más corto.

El aspecto central de la infraestructura es el impacto del modelo de recursos en el desempeño en el contexto de Internet. El servicio de notificación de eventos distribuido compara varias topologías incluyendo jerárquica y grafo no cíclico. La comunicación entre los servidores en la topología es P2P. La principal desventaja de este sistema en el contexto de nuestro proyecto es la vaga definición del modelo de objetos. Esto hace que las aplicaciones que usan el SNE deban ser modificadas para explícitamente generar los eventos y empujarlos al sistema.

3.4 EDEM

EDEM [8] desarrollado en la Universidad de California en Irvine, es una infraestructura de monitoreo de uso de aplicaciones. La infraestructura permite evaluar el uso de aplicaciones por parte de los usuarios. El modelo de observación es un aporte interesante de la infraestructura, ya que ésta es lograda de manera externa a las aplicaciones a través de *probes*. Un *probe* es un componente externo que observa la ejecución de una aplicación y genera eventos. EDEM usa gatillos (triggers), guardas y acciones similares a reglas ECA (Evento Condición Acción) para el monitoreo y modificación del estado de aplicaciones usando 'probes' externos.

Los autores de este SNE introducen 4 actividades diferentes en su framework de monitoreo de eventos: Observación, procesamiento, notificación y acciones. La observación es el proceso de recolección de información básica de las aplicaciones. El procesamiento incluye computaciones sobre información básica para producir eventos. La notificación incluye la comunicación a los grupos interesados de la ocurrencia de eventos. Finalmente, las acciones son el proceso ejecutado como reacción a la notificación u observación en las aplicaciones.

Esta infraestructura presenta una visión innovadora para el modelo de observación y notificación basado en las actividades de monitoreo; los requerimientos principales son similares de los presentados en el contexto de esta tesis. Sin embargo, el dominio de aplicaciones al que se puede aplicar es bastante restringido.

3.5 HERMES

Hermes [26] es un middleware de eventos que da soporte estándar a servicios de observación y notificación de eventos. El sistema es de propósito general, su enfoque central es el modelo de recursos, de eventos y de notificación. Los requerimientos principales que aborda el middleware son escalabilidad, interoperabilidad, expresibidad, usabilidad y confiabilidad.

El middleware propuesto está compuesto por dos componentes: clientes de eventos y *brokers*. Los Clientes de eventos pueden ser publicadores o subscriptores, son livianos y desarrollados en cualquier lenguaje de programación. La única restricción es que debe comunicarse con los *brokers* a través de mensajes XML. Los *brokers* pueden ser de 3 tipos: *publishing-hosting broker*, *subscriber-hosting broker* o *routing broker*.

Los *brokers* implementan el modelo de recursos del sistema; estos están conectados en topología basada en una red de enrutamiento con comunicación p2p. Los *brokers* están distribuidos en una

topología arbitraria. La red de enrutamiento garantiza la disponibilidad del sistema en caso de la caída de uno o más *brokers.*

El modelo de eventos define una aproximación de eventos con tipo organizados en una jerarquía con herencia. Los eventos son definidos en un esquema orientado a objetos que integra la infraestructura con lenguajes, middleware y arquitecturas orientadas a objetos.

Los *brokers* son el mecanismo de distribución de eventos, este caracteriza el modelo de notificación del sistema. El sistema usa algoritmos escalables para la declaración de suscripciones y diseminación de eventos. Una suscripción define un tipo de eventos y una expresión de filtro. El enrutamiento de eventos es implementado con el algoritmo de *content-based routing* [21], los eventos son filtrados tan cerca como sea posible a las fuentes de eventos para reducir el ancho de banda e incrementar escalabilidad.

Al igual que SIENA este sistema no define un modelo de objetos explícitamente, esto conlleva a las mismas desventajas que presenta SIENA en este aspecto, en particular presenta desventajas en la definición de un modelo de observación claro.

3.6 Discusión

Los sistemas presentados ofrecen una visión de las decisiones de diseño relevantes para un SNE. Todos cumplen con los requerimientos básicos de bajo acoplamiento e integración laxa y la comunicación para aplicaciones distribuidas a escala Internet. Hemos tomado ideas de varios de ellos en el modelo de nuestro proyecto.

Sin embargo, <los requerimientos particulares de estas infraestructuras difieren de los necesarios para el contexto GSD presentado. Esto es principalmente porque cumplen con los requerimientos de un dominio diferente, porque el enfoque en el diseño de sus modelos tiene otras preocupaciones, o porque los requerimientos no funcionales, como la plataforma de las aplicaciones que integran, difieren. Sin embargo, algunos de los requerimientos de varios SNEs son similares a los de este proyecto. Por esta razón las soluciones presentadas son de nuestro interés y resaltan factores críticos que nuestro modelo debe considerar.

Solo EDEM presenta un modelo de objetos que cumple con el requerimiento principal de reducir las modificaciones en las aplicaciones para usar la infraestructura. Las actividades de monitoreo presentadas por los autores son un mecanismo interesante para describir la interacción entre el SNE y las aplicaciones. Sin embargo, como fue mencionado anteriormente, este sistema tiene restricciones fuertes sobre el tipo de aplicaciones que se pueden integrar.

Los autores de JEDI proponen un esquema interesante de comunicación mixto de eventos y request/reply aunque no lo implementan en su propuesta. Nosotros creemos que este esquema presenta varias ventajas. En particular la carga sobre el sistema de eventos se puede reducir a la mínima necesaria. Las aplicaciones que reciben notificaciones pueden solicitar información adicional que complemente la del evento por request/reply.

Hermes presenta una descripción y caracterización de los eventos bastante robusta, la tipificación y jerarquización de eventos son factores positivos. Adicionalmente, nos parece interesante que los clientes de eventos pueden sean publicadores o subscriptores. Nosotros tomamos estas ideas, sin embargo nuestros eventos no son jerárquicos.

El modelo propuesto en el proyecto presentado posteriormente aprovecha los resultados y experiencias de los varios modelos y sistemas que se estudiaron. A partir de estos se pretendió construir un nuevo SNE, en parte por las desventajas resaltadas en algunos de los sistemas presentados; en parte para experimentar y tomar decisiones de diseño que permitieran mejorar y

contribuir a la investigación en el tema. Adicionalmente nuestro modelo pretende llevar la descripción de la integración a un nivel más alto que el que, en general, proveen estos sistemas.

4 Tecnologías de Integración

Este capitulo presenta una serie de tecnologías y conceptos que complementan el modelo de integración del proyecto. Las reglas Evento Condición Acción son introducidas como un mecanismo eficiente para reaccionar a eventos notificados por un SNE.

El monitoreo y administración de aplicaciones se introduce por su alta relevancia en el contexto GSD [14]. Junto con los conceptos centrales se presentan algunas tecnologías, en particular JMX que es el estándar de facto en la industria para monitoreo y administración de aplicaciones en JAVA [24].

4.1 Reglas ECA

Las reglas Evento Condición Acción han sido utilizadas en varios dominios desde su inicio en bases de datos activas [5], [20]; uno de estos dominios es “enactment” de procesos. Las ideas presentadas por varios autores [1], [6], [9], [10], [18] apoyan el uso de reglas ECA para control de procesos y ejecución de workflow a varios niveles.

4.1.1 EVE

EVE [6], [18], desarrollado en la Universidad de Zurich, es un componente de middleware para ejecución de workflow extensible que usa una infraestructura basada en eventos. El sistema propuesto provee un framework arquitectural basado en componentes; esta aproximación permite la reutilización y composición de entidades de un proceso. El sistema es construido extendiendo las funcionalidades de un SNE, este sistema provee administración y monitoreo del proceso a través de historiales de eventos.

La plataforma está basada en el uso de componentes reactivos cuya interacción es caracterizada por reglas ECA. Estos componentes reactivos son llamados brokers o entidades de procesamiento. Las reglas ECA implementan la interacción entre los brokers y el proceso. Una regla ECA en EVE es una tripleta del estilo (tipo de evento, condición, acción). Cuando una instancia del evento ocurre, si la condición se cumple, se ejecuta la acción. Las condiciones devuelven un conjunto de referencias a objetos; una condición se considera cumplida si devuelve un conjunto no vacío. Las referencias son pasadas como parámetros a la acción. Las acciones son fragmentos de código arbitrario, típicamente incluyen notificar a un broker una solicitud de servicio.

El modelo de ejecución de workflow presentado tiene varias ventajas: El modelo de coordinación basado en eventos permite la especificación completa de un proceso sin imponer suposiciones sobre la arquitectura concreta del proceso. Adicionalmente, permite modificaciones en el proceso durante ejecución

4.1.2 YEAST

YEAST [10], desarrollado en los laboratorios AT&T Bell, es un sistema especializado en procesar eventos complejos a través de reglas de evento-acción para aplicaciones en una red local. El sistema es de propósito general y tiene como requerimientos principales: Apertura en las interfaces, documentación y funcionamiento del sistema, para que el sistema pueda ser acoplado con otras herramientas fácilmente; baja restricción en acciones de respuesta a eventos; que provea un lenguaje de especificación simple; y que sea extensible, confiable, monitoreable y seguro.

El sistema tiene una arquitectura cliente/servidor. Las responsabilidades del servidor son registrar y administrar especificaciones, y durante ejecución emparejar (match) eventos con sus especificaciones. Una especificación describe un patrón de eventos de interés al usuario y la acción a ejecutar cuando se identifique la ocurrencia del patrón. Un patrón de eventos es un patrón compuesto de descriptores de eventos primitivos. La acción es una secuencia válida de comandos que pueden ser ejecutados por el interpretador de comandos del sistema. Al terminar la ejecución de una acción, todas las salidas que no fueron dirigidas a archivos o a pipes son enviadas por correo electrónico al usuario que registro la especificación.

En el modelo, los eventos primitivos corresponden al cambio en el valor de un atributo de un objeto de alguna clase. Los eventos primitivos son divididos en dos clases: eventos de tiempo y eventos de objetos.

Este sistema presenta serias desventajas pues está diseñado para ejecución en una red local, y tiene una arquitectura de detección y ejecución de eventos centralizada.

4.1.3 Otros sistemas ECA

Bae et Al., [1] en la Universidad Nacional de Chonbk en Korea, introducen una propuesta para la ejecución y control de workflow usando reglas ECA. El método propuesto combina una representación gráfica del proceso con reglas ECA; los autores introducen un modelaje de procesos que permite la ejecución de reglas a través de reglas ECA. Este modelo de reglas es bastante interesante en cuanto al nivel de control de workflow que permite con las reglas, el interés principal del sistema es en el lenguaje de descripción de los procesos.

Kappel et Al., [9] en la Universidad de Linz, Austria, presentan un sistema de administración de workflow con preocupación central en la reusabilidad y adaptabilidad. El framework presentado incluye un modelo basado en reglas para caracterizar la coordinación de actividades y recursos. El modelo está basado en reglas ECA en bases de datos activas orientadas a objetos, que encapsulan las políticas de coordinación para ordenamiento de actividades, selección de agentes y administración de lista de trabajo.

Estos dos sistemas a pesar de presentar ventajas no son distribuidos, una desventaja crítica en el contexto GSD. Sin embargo, tomamos ideas de ellos y las integramos en nuestra propuesta.

4.1.4 Discusión

Los sistemas presentados nos muestran que, a pesar de la larga historia que han tenido las reglas ECA, todavía son relevantes, y pueden dar soporte a la ejecución y control de procesos en varios niveles. El estudio de este tema nos ha ayudado a diseñar la interacción con las aplicaciones en nuestro modelo; nuestra propuesta toma algunas ideas de estos sistemas.

El uso de componentes reactivos presentado por EVE es de alto interés para nuestro proyecto. Es altamente deseable que la interacción caracterizada por reglas ECA esté definida en un componente diferente al que se encarga de las notificaciones de eventos. Sin embargo, en este caso es necesaria una herramienta que permita administrar estos componentes y sus reacciones.

El enfoque principal de algunas de estas propuestas es en el control a través de reglas en niveles diferentes a los que abarca este proyecto. La mayoría son usados a un nivel de alta abstracción: la ejecución y control a nivel del workflow.

Las reglas ECA en nuestra propuesta son usadas para describir la reacción de componentes a notificación de eventos; sin embargo, los componentes que reaccionan no hacen parte de nuestro modelo. La mayoría de estas propuestas establecen demasiadas restricciones sobre estos aspectos, no son distribuidas o en general no cumplen con los requerimientos básicos del contexto GSD. Nuestra propuesta para ejecución de reglas ECA hace un aporte relevante al integrar este mecanismo con Web Services para interacción con aplicaciones heterogéneas, imponiendo un mínimo de restricciones sobre las aplicaciones que reaccionan a las reglas.

4.2 Monitoreo y Administración

La administración de aplicaciones se refiere a las capacidades de una aplicación relacionadas a las funciones de supervisión y administración [24]. En la práctica ofrece servicios a los desarrolladores para hacer deploy y configurar una aplicación, monitorear su estado y desempeño, predecir cuando se producen problemas y tenerla información y herramientas para analizar, corregir y reportar fallas.

En la actualidad existen cinco tecnologías de administración ampliamente usadas, algunas como reportes a archivos de logs o SNMP son protocolos de bajo nivel que ofrecen servicios a plataformas de administración de más alto nivel como WBEM, ARM, y JMX [24].

Los reportes a archivos de logs son el mecanismo más simple de administrar una aplicación. En este mecanismo la aplicación escribe directamente a un archivo los logs de eventos relevantes para la administración. La ventaja de este mecanismo es la simplicidad. Las desventajas son que ofrece solo comunicación en un sentido y que puede presentar problemas de desempeño o producir logs demasiado extensos y complejos.

SNMP, Simple Network Management Protocol, es una tecnología basado en el paradigma administrador/agente; cada objeto administrado es atendido por un agente responsable de actualizar la información del objeto. Es usada principalmente en aplicaciones simples para generar alertas. Sus ventajas son simplicidad y ubicuidad. Sus desventajas principales son que el sistema de transporte de mensajes no garantiza entrega y que la simplicidad del modelo puede causar problemas para expresar requerimientos de administración complejos.

WBEM o Web-Based Enterprise Management es una tecnología sofisticada y poderosa de administración. Consiste de un modelo de datos orientado a objetos una especificación de codificación de mensajes y un mecanismo de transporte. Las ventajas principales de esta tecnología son administración en dos sentidos, comprehensiva y poderosa, y su soporte extenso a plataformas Windows. Sus desventajas principales son que es una tecnología intrusiva y tiene una curva de aprendizaje inicial compleja.

ARM es una herramienta para seguimiento y monitoreo de transacciones con altos requerimientos de desempeño. Para usar esta tecnología el desarrollador debe insertar llamados a servicios de ARM en el código de la aplicación en los puntos relevantes de una transacción. Sus ventajas principales son la alta utilidad de los datos que recolecta y el poco costo en desempeño con su uso. Las desventajas principales son que puede ser intrusivo en algunos casos y la alta dependencia de los componentes de ejecución de ARM.

JMX es actualmente el estándar industrial para administración de aplicaciones Java [24]. JMX define una arquitectura de administración, sus APIs y servicios [25]. La arquitectura de JMX está dividida en tres niveles: nivel de instrumentación, nivel de agentes y nivel de administración. El nivel de instrumentación ofrece administración de cualquier objeto java. Este modelo describe a través de Management Beans, o MBeans, qué servicios de administración que ofrecerá la aplicación. Para el desarrollador de una aplicación la parte más importante de JMX es este nivel [24].

El nivel de agentes provee agentes de administración. Un agente de JMX es un contenedor que provee servicios de administración centrales que puede ser extendido dinámicamente. El nivel de administración provee componentes de administración que pueden operar como administración o agente para la distribución y consolidación de servicios de administración.

4.2.1 Discusión

Las opciones estudiadas para administración y monitoreo de aplicaciones presentan ventajas y desventajas para su uso en el proyecto. La desventaja principal de las plataformas como WBEM y ARM es la modificación del código para que explícitamente incluya los llamados a servicios administrativos. Los protocolos básicos estudiados, como escritura a arcivos de logs son ampliamente usados en la plataforma, estos son una herramienta de soporte a la administración, sin embargo, las pocas posibilidades que ofrece lo hacen un soporte útil, más es insuficiente como única herramienta de administración y monitoreo.

Para el proyecto se seleccionó JMX para administración y monitoreo, esto es, primero por que es el estándar actual para aplicaciones JAVA. Adicionalmente, el uso de JMX nos ha permitido dividir el proceso de desarrollo de la administración de la infraestructura. Con JMX el desarrollo de la instrumentación de la aplicación es separado al desarrollo de una consola o de los agentes.

5 Eleggua

5.1 Introducción

El contexto GSD impone una serie de restricciones adicionales que debe cumplir la infraestructura, los componentes deben ser distribuidos, y deben dar un soporte a comunicación sincrónica y asincrónica de las aplicaciones. Adicionalmente el monitoreo del proceso y la administración de la infraestructura son necesarias para garantizar el flujo correcto del proceso o recuperación de fallos que asegure la integridad del proceso.

Los requerimientos principales de nuestra infraestructura son bajo acoplamiento e integración laxa para componentes distribuidos y reducción de modificaciones en las aplicaciones para la integración a nivel de Internet. Se entiende bajo acoplamiento e integración laxa como una integración que permita la adición de nuevos componentes con bajo impacto sobre la infraestructura; las aplicaciones, ya desarrolladas y heterogéneas, deben conocer lo mínimo posible unas de otras. La reducción de modificaciones en las aplicaciones pretende reducir el esfuerzo necesario para integrar las herramientas. A largo plazo esto, además, facilita la evolución y mantenimiento de la integración.

En la sección anterior se presentaron tecnologías y modelos que pretenden abordar este tipo de requerimientos. Los sistemas de notificación y publicación basados en eventos cumplen con los requerimientos de bajo acoplamiento e integración laxa. Se presentó un sistema que pretende solucionar el problema de observación de aplicaciones a través de componentes externos que reaccionan al cambio de estado de una aplicación. Adicionalmente, se presentaron algunos sistemas que usan reglas de Evento Condición Acción (ECA) para describir la reacción de componentes a eventos. Finalmente, se presentaron algunas tecnologías de comunicación útiles en el contexto actual.

El proyecto presenta tres aportes principales: un modelo de integración para aplicaciones heterogéneas y distribuidas, la definición de una infraestructura para dar soporte al modelo, la integración de varias tecnologías para su implementación, y la validación del modelo y la implementación en un contexto real.

El modelo propuesto en el proyecto aborda los requerimientos presentados anteriormente desde su diseño y especificación, de esta manera soluciona y facilita el proceso de integración de aplicaciones en un proceso GSD. Para este fin se definió un modelo basado en notificación de eventos, reglas ECA y observación de aplicaciones.

El proyecto retoma el conjunto de conceptos y tecnologías presentadas y las agrupara para dar una solución a un problema en un dominio bien definido. El modelo se concretó en una infraestructura usando varias tecnologías para la comunicación, intercepción de servicios para observar de manera no intrusiva las aplicaciones, administración y control de aplicaciones.

Por último, se hizo una validación de la infraestructura, para esto se definió, diseñó, implementó y ejecutó la integración en un proceso en una casa de software que se enfrenta a los problemas y dificultades del desarrollo de software globalizado.

A continuación se presenta un escenario de ejemplo de un proceso de pruebas de software para ilustrar con mayor facilidad la infraestructura. Después se introducen los conceptos principales de Eleggua, y el enfoque tomado para el diseño del modelo. Posteriormente se presentan los detalles de la infraestructura y cada uno de sus componentes. Finalmente, se describen los servicios de administración y monitoreo de Eleggua.

5.2 Escenario de Ejemplo

Antes de introducir los detalles de la infraestructura se presenta un ejemplo de un proceso que incluye la cooperación de tres aplicaciones; este proceso hace parte de la validación que será presentada posteriormente. El escenario está basado en la ejecución de un proceso sencillo de pruebas de software. En paralelo a este proceso se deben ejecutar actividades de planeación y seguimiento para programar todas las tareas requeridas durante el desarrollo (incluyendo las actividades del proceso de pruebas), y para registrar el tiempo gastado por los participantes del proceso en las tareas.

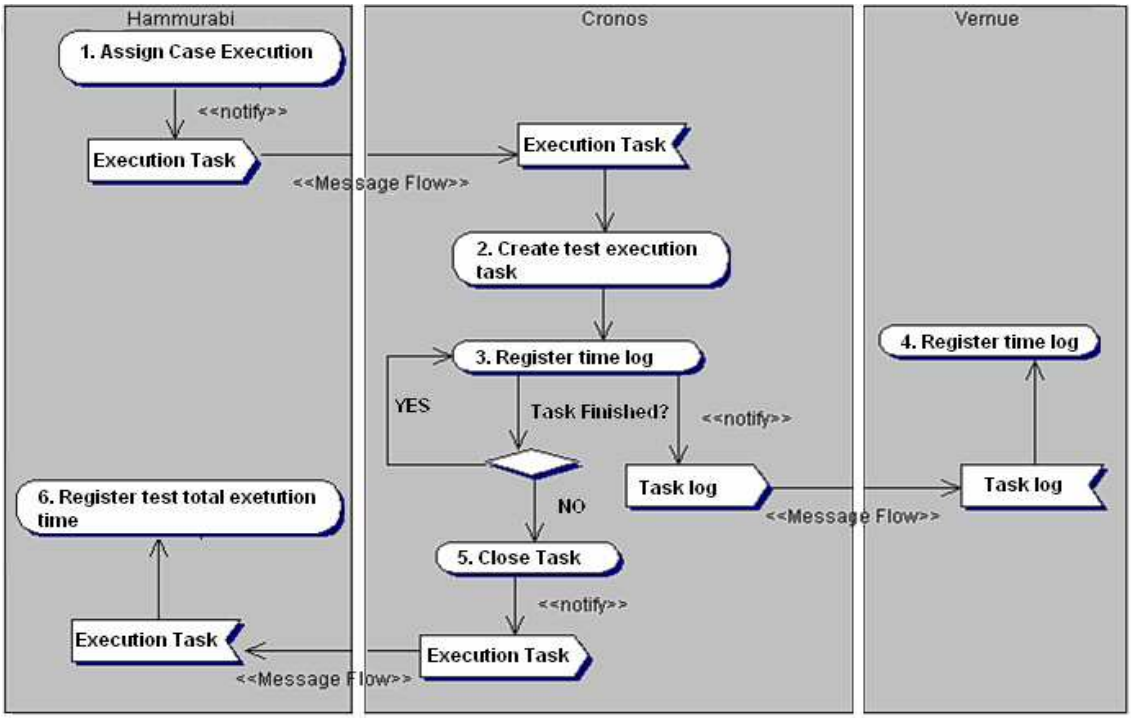

Figura 5.1 Escenario de ejemplo

El escenario de ejemplo es presentado en la Fig. 5.1 usando la notación de diagramas de actividad de UML. En este se presenta el proceso de ejecución de un caso de prueba desde su asignación hasta el registro total de tiempo. Durante el proceso la tarea de ejecución es asignada por un usuario y creada para que esté disponible para, posiblemente otro usuario responsable de la ejecución. El usuario responsable debe registrar el tiempo invertido en la ejecución; al finalizar este tiempo debe estar disponible para reportes del proyecto.

Existen varias aplicaciones disponibles para dar soporte al proceso. Hammurabi es usada en el proceso para crear planes de prueba, ejecuciones y registrar reportes de defectos. Cronos es una aplicación de agenda y rastreo del proceso; provee servicios para registrar tareas planeadas y registrar tiempo gastado en las tareas. Vernue, una aplicación de administración y control de proyectos es usada para calcular pagos de empleados con base al tiempo trabajado en las actividades de los proyectos.

Vernue, Hammurabi y Cronos fueron desarrolladas independientemente, Vernue es una aplicación legado desarrollada en plataforma AS400, Cronos y Hammurabi son aplicaciones J2EE. Las aplicaciones no comparten datos, y aunque desde una perspectiva abstracta utilizan con los mismos conceptos de dominio, estos suelen tener representaciones y propiedades diferentes en

cada aplicación. Estas aplicaciones ofrecen servicios de acceso a sus funcionalidades a través APIs, de Web Services y de componentes que integran a nivel de datos (en el caso de AS400).

Eleggua debe proveer los siguientes servicios:

- Cuando una ejecución de caso de prueba es asignada a un usuario en Hammurabi (Fig. 5.1 actividad 1), automáticamente, en Cronos se debe crear una tarea de ejecución de caso de prueba y debe ser asignada al desarrollador responsable (Fig. 5.1 actividad 2). Con esta información se planea la tarea de ejecución de caso de prueba.

- A continuación, el desarrollador a cargo de la tarea debe registrar el tiempo gastado en las actividades de ejecución del caso de prueba (Fig. 5.1 actividad 3). El tiempo gastado en estas tareas debe ser enviado automáticamente a Vernue cada vez que un registro es creado para calcular varios reportes.

- Una vez la tarea es terminada, el desarrollador debe cerrar la tarea de ejecución de caso de pruebas (Fig. 5.1 actividad 5), esto debe enviar de manera automática el tiempo total gastado en la tarea a Hammurabi para generar otros reportes.

5.3 Modelo de Integración

La figura 5.2 presenta un esquema del modelo de integración de Eleggua. Los elementos principales del modelo son el SNE y los representantes de las aplicaciones. El SNE se encarga de administrar los tipos de eventos y las suscripciones; este recibe eventos y los despacha hacia los representantes de las aplicaciones. Un representante de una aplicación encapsula las reglas que definen la integración de esta aplicación con Eleggua a través de reglas ECA y reglas de Observación. Las reglas de observación definen qué eventos son observados y generados por la aplicación externa; las reglas ECA describen cómo se procesan los eventos para despachar al SNE y las reacciones a la notificación de eventos del SNE.

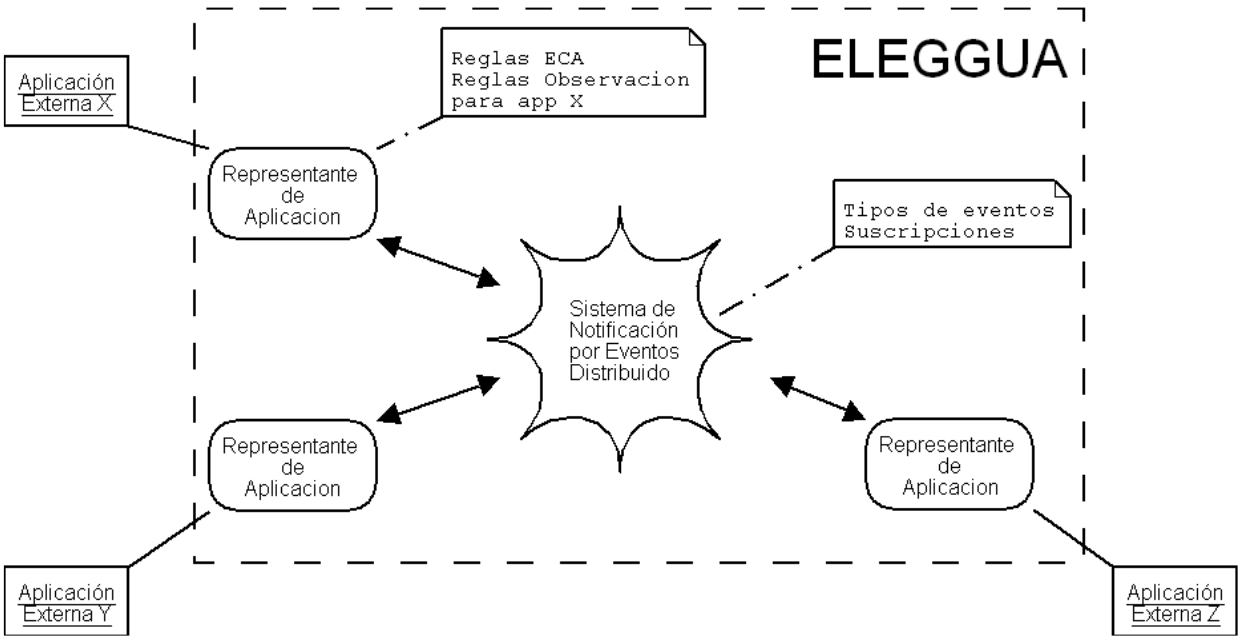

Figura 5.2 Componentes del Modelo de Eleggua

A continuación se describen los conceptos básicos del modelo y posteriormente se detalla el enfoque de notificación por eventos para describir a mayor detalle las reglas de observación y ECA que describen y encapsulan la integración.

5.4 Conceptos Básicos

Los conceptos principales introducidos en Eleggua que encapsulan la cooperación de aplicaciones son los eventos lógicos y los tipos de eventos. Estos se introducen como base para la descripción de la propuesta. La especificación detallada del modelo de Eleggua se encuentra en los anexos.

5.4.1 Tipos de eventos

Un tipo de evento es una dupla <*topic, contexto*>; un tópico representa un concepto de dominio común para un grupo de aplicaciones. Estos conceptos están basados en acuerdos de nombramiento e identificación de estos conceptos de dominio sobrepuesto en las aplicaciones. Un contexto representa un dominio de eventos lógicos; por ejemplo, el contexto *'process.testing'* agrupa los eventos enviados durante la ejecución del proceso de pruebas del escenario; el contexto 'error', incluye todos los eventos relacionados con errores en el proceso.

5.4.2 Eventos Lógicos

Los eventos lógicos son el concepto central para el intercambio de información entre las aplicaciones. Un evento lógico es una instancia de un tipo de evento, la identificación del productor, una marca de tiempo establecida en el momento de producción y un conjunto de parámetros nombrados.

En el escenario presentado hay varios eventos lógicos, uno para cada mensaje enviado entre los sistemas. Primero, la tarea de ejecución es representada por un evento lógico con la información de una ejecución de caso de prueba, este evento contiene como parámetros el identificador de la ejecución, identificación del proyecto e identificación del usuario responsable. El productor del evento lógico es Hammurabi. Segundo al registrar tiempos, se debe generar un evento lógico con el registro de tiempo de un desarrollador para una tarea de ejecución de caso de prueba. Este evento debe contener información sobre el desarrollador, el tipo de tarea y el tiempo gastado. Finalmente, al cerrar la tarea, se debe generar un evento lógico que señala la finalización de la tarea de ejecución de caso de prueba. Este evento debe contener como parámetros el identificador del proyecto, de la ejecución y el tiempo total gastado en la tarea. Estos dos eventos son producidos por Cronos.

5.5 Enfoque de notificación por eventos

La propuesta se enfoca en dos de los modelos del framework de diseño de un ENS [12]: el modelo de observación y el de notificación. Además se usan las actividades de monitoreo de aplicaciones [8] para describir estos modelos: las actividades de observación y procesamiento describen el modelo de observación; las actividades de notificación y acciones describen el modelo de notificación.

5.5.1 Modelo de observación

El modelo de observación está basado en describir cómo observar y recolectar información de las aplicaciones para procesar y producir eventos lógicos. El procesamiento es necesario para reducir la carga en el sistema causada por eventos con información muy pesada. Sólo se generan eventos de interés con el mínimo de información necesaria.

La observación y procesamiento de eventos es descrita en reglas de observación. Eleggua permite la intercepción de ejecución de los servicios de una aplicación. Una regla de observación especifica la intercepción de un servicio de interés y describe cómo debe ser procesado para generar un evento lógico.

5.5.2 Modelo de Notificación

El modelo de notificación está basado en la funcionalidad de un SNE que ofrece servicios de publicación/suscripción. Una suscripción es declarada para un tipo de eventos; el SNE notifica eventos lógicos a las aplicaciones interesadas. Se usan reglas ECA para describir la reacción a eventos lógicos notificados. El mecanismo de notificación es externo al código de la aplicación que usa la infraestructura. Eleggua genera filtros sobre los parámetros especificando el tipo de eventos y las condiciones sobre los parámetros descritos en la regla: solo se procesan los eventos lógicos que pasan los filtros.

Los eventos lógicos contienen el mínimo posible de información para reducir la carga. Las acciones combinan la aproximación por eventos con un mecanismo de request/reply: Web Services. Cualquier información adicional, requerida para completar las reacciones a eventos lógicos notificados, es solicitada a través de Web Services o algún otro mecanismo de solicitud/respuesta. Las acciones usualmente incluyen la invocación del API la aplicación que modifica la regla, la invocación del API de otras aplicaciones a través de Web Services o un API y la generación de eventos lógicos.

En este escenario se debe describir la cooperación para todos los eventos que cruzan la frontera de un sistema: mensajes de tarea de ejecución y de registro de tiempo de tarea. Los mecanismos para expresar esto en Eleggua son eventos lógicos, reglas ECA y reglas de observación. El escenario de ejemplo es usado para ilustrar el comportamiento de los componentes de Eleggua.

En el escenario de ejemplo debe haber observaciones en todos los puntos que queremos generar un evento lógico. Debe observarse la asignación de ejecuciones de casos de prueba en Hammurabi, el ingreso de registros de tiempo en Cronos y la finalización de una tarea en Cronos. Adicionalmente cada una de estas observaciones debe tener asociada una regla ECA que procesa el evento observado y lo envía a la infraestructura.

Por otra parte, debe haber una regla ECA en cada sistema que recibe estos eventos y que debe describir la reacción a estos. Para Cronos hay una regla ECA que describe la reacción al evento de asignación de ejecución de caso de prueba; especifica que se debe crear una nueva tarea para esta ejecución. Para Vernue se declara una regla ECA que describe la reacción a eventos de registro de tiempo; la regla declara que la información del registro es almacenada en el sistema local. Por último, para Hammurabi una regla ECA caracteriza la reacción a eventos de finalización de una tarea de ejecución de caso de prueba; esta especifica que el tiempo total de la tarea debe ser asignado a la información de la ejecución del caso de prueba.

6 Arquitectura de la Infraestructura

6.1 Descripción Global

La figura 6.1 presenta el modelo de implementación y distribución de Eleggua. Cada aplicación, en un servidor de aplicaciones distribuido, tiene un representante (RA) que se encarga de observar eventos a nivel del API de las aplicaciones externas y reaccionar a notificaciones en las aplicaciones (App x) modificando el estado de la aplicación a través de llamados del API. Las aplicaciones desarrolladas en plataformas que no puedan ser observadas a nivel del API pueden ser integradas a nivel de datos construyendo extensiones del RA. El representante envía y recibe eventos a través de los componentes locales del SNE. Finalmente los componentes locales del SNE recaen en el servicio de distribución del SNE para la diseminación y notificación de eventos. A continuación se presentan la implementación e interacción de estos componentes.

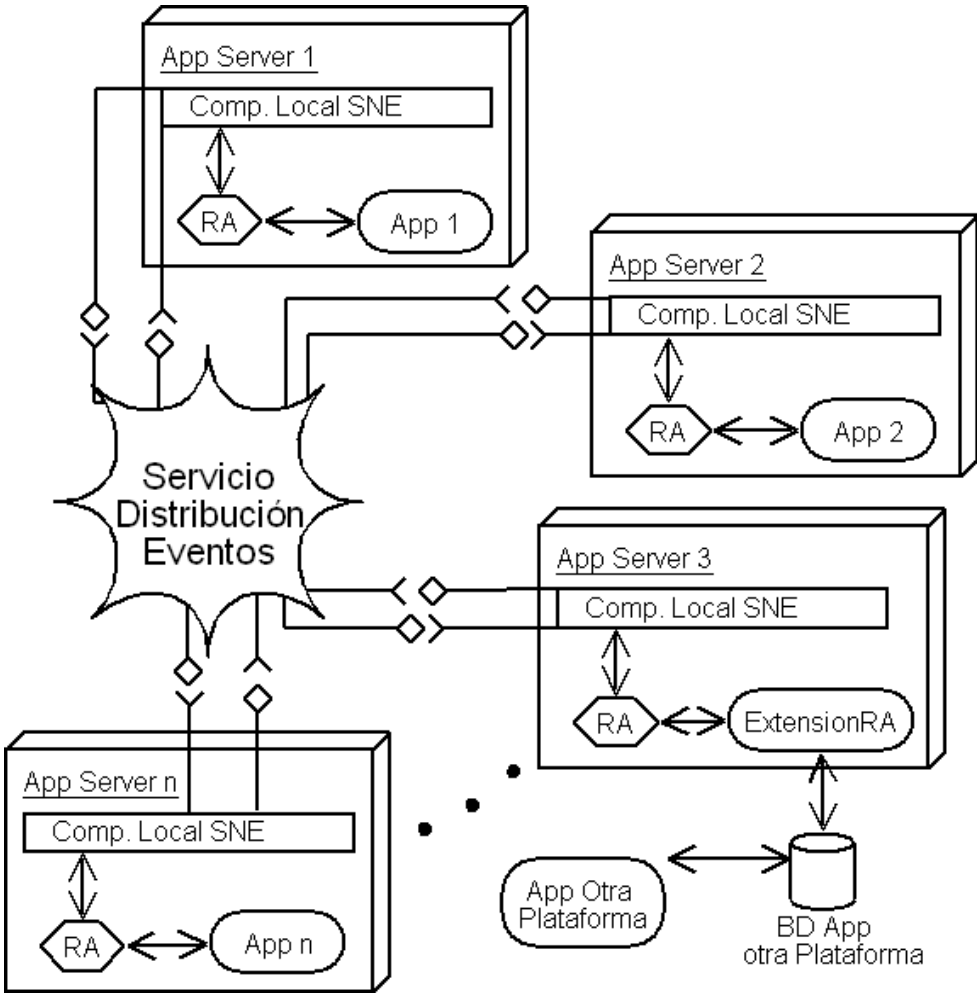

Figura 6.1 modelo de implementación y distribución de Eleggua

6.2 Detalles de la Arquitectura

La figura 6.2 presenta los componentes principales de Eleggua que implementan los presentados en la figura 6.1 y su interacción. El Middleware de Eventos Distribuidos (DEM), implementa el servicio de distribución eventos y el componente local eventos; este provee el servicio de notificación de eventos. El componente Proxy de Cooperación (CP), implementa los Representantes de Aplicación (RA en la figura 6.1) intermedia la interacción entre el DEM y las aplicaciones. Un 'proxy' es un mecanismo externo que observa y extrae información. El DEM notifica eventos lógicos al CP a través de un mecanismo push/pull. Usualmente los eventos son notificados a través de un mecanismo push. El mecanismo pull es usado en caso de que una aplicación no se encuentre disponible al momento de procesar un evento. En este caso, al reestablecer la conexión el CP solicita por pull los eventos recibidos.

Las aplicaciones externas interactúan a través del CP; las reglas ECA y de observación describen esta interacción. En caso de ser posible la intercepción de servicios del API la interacción se hace a este nivel para la observación, en caso contrario una extensión al CP debe ser desarrollada para integrar a nivel de datos. Esta extensión debe encargarse de revisar con cierta periodicidad el estado de la aplicación y notificar al CP. Los componentes son presentados en detalle en las siguientes secciones.

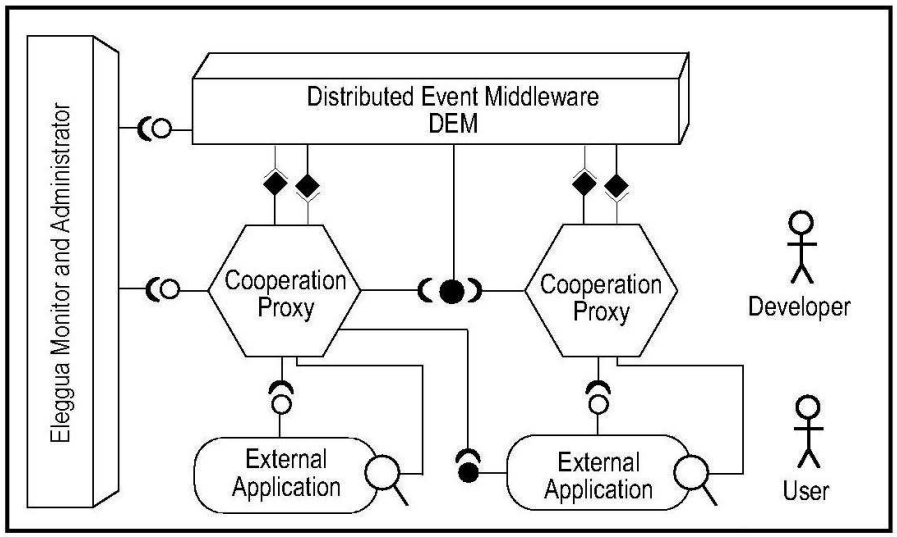

Figura 6.2 componentes Eleggua

La figura 6.3 presenta la notación gráfica usada en Eleggua. La figura 6.3 (a) representa el intercambio de eventos lógicos notificados a una aplicación. La figura 6.3 (b) representa una interfaz de Web Services ofrecida por un componente (circulo negro) e invocado por otro componente (media circunferencia). Finalmente, la figura 6.3 (c) representa la intercepción de la ejecución de un servicio en el componente que contiene la lupa, es decir, cuando un usuario ejecute cierto servicio en la aplicación externa el componente CP es notificado y adquiere acceso a los datos de la solicitud y respuesta del servicio.

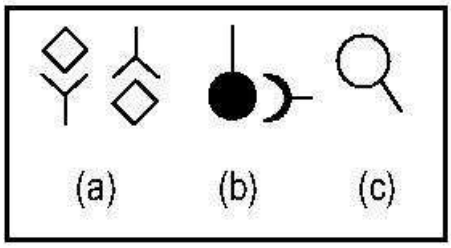

Figura 6.3 Notación de Diagrama

6.2.1 Middleware de Eventos Distribuidos (DEM)

El DEM ofrece la funcionalidad básica de un SNE de tipo publicar/suscribir. Sus responsabilidades principales son: 1) registro de aplicaciones, 2) suscripción de aplicaciones, 3) despacho de eventos y notificación a los CPs, 4) persistencia de eventos, y 5) búsquedas de eventos históricos. El DEM está compuesto de tres tipos de sub-componentes: componente de despacho, componente consumidor y componente productor.

El componente de despacho es el nodo principal de comunicación entre componentes consumidores y productores. El componente consumidor y productor proveen un API para interactuar con el DEM para consumir y producir eventos respectivamente. Los componentes están organizados en una topología que tiene como nodo central al componente de despacho que ofrece servicios a varios componentes productores y consumidores.

Las siguientes secciones detallan cómo se materializan las responsabilidades del DEM en los sub-componentes.

Registro de Aplicaciones

Este es el primer servicio invocado por una aplicación que va a usar el DEM; este registra la información básica de una aplicación. El registro de una aplicación para consumir y producir eventos es independiente. El registro para consumir eventos puede incluir opcionalmente la especificación de un manejador de eventos para notificación de eventos por mecanismo de push. El registro de una aplicación es hecho a través del componente productor o consumidor; la información de registro es almacenada y propagada al componente de despacho. El registro de aplicaciones permite identificar a una aplicación en el sistema, sin embargo, esta no puede publicar ni recibir eventos hasta que no se suscriba a algún tipo de evento.

Suscripción de aplicaciones

Una aplicación debe suscribirse a un tipo de eventos antes de poder publicar o recibir notificaciones de este tipo de eventos; la suscripción es independiente para publicar o para recibir. La suscripción a un tipo de eventos es ofrecida como servicio por los componentes productor y consumidor.

Despacho de eventos

El despacho de eventos es la distribución de eventos a aplicaciones interesadas, suscritas como consumidores al tipo de eventos. Los eventos producidos por una aplicación son empujados (pushed) de la aplicación al componente productor; este componente envía los eventos al componente de despacho via push. El componente de despacho propaga a todos los componentes consumidores. Los componentes consumidores almacenan información sobre las aplicaciones

suscritas; los eventos son almacenados y opcionalmente empujados a las aplicaciones interesadas si un manejador de eventos fue declarado. Un manejador de eventos es una entidad declarada al momento de registrar una aplicación que se encargará de procesar un evento, este servicio es ofrecido por el Proxy de Cooperación que será detallado en la siguiente sección.

Persistencia de eventos

Los eventos lógicos son persistidos en el componente despachador y en los componentes consumidores; una copia del evento por cada aplicación que ha declarado interés en el tipo del evento. Los eventos son marcados como consumidos si el manejo del evento se efectuó sin problemas, en caso de que el evento haya sido empujado, o cuando sea solicitado explícitamente por la aplicación. Los eventos son persistidos temporalmente en el componente productor para recuperación a fallos.

El paso de eventos entre los componentes es asincrónico, cada componente tiene un manejo transaccional separado que busca minimizar las posibilidades de pérdida de información. En general al recibir un evento este es almacenado localmente en cada componente, posteriormente se inicia una transacción, la transacción termina al despachar, notificar o procesar el evento.

Otros Servicios

El DEM ofrece servicios extendidos a través de Web Services; estos servicios incluyen búsquedas de eventos históricos, de tipos de eventos y de suscripciones de productores y consumidores. Estos servicios son usados como parte de las acciones en reglas ECA.

6.2.2 Proxy de Cooperación CPs

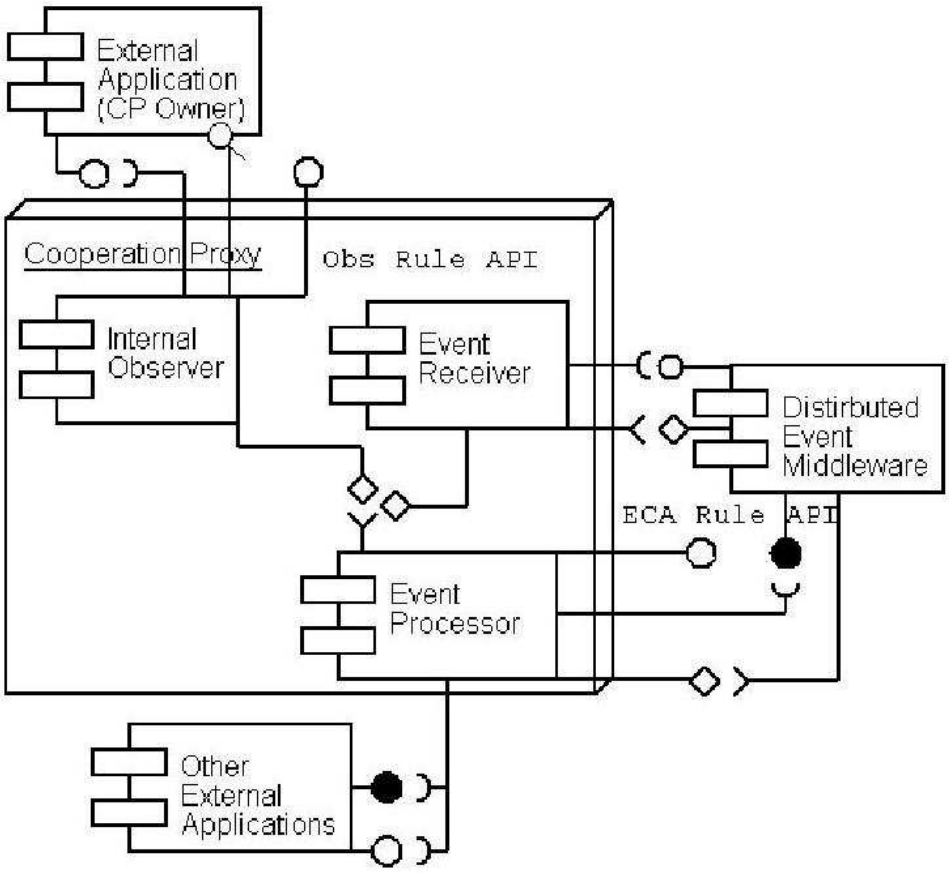

Figura 6.4 Detalle de los componentes del Proxy de Cooperación

Los Proxies de Cooperación (CPs) intermedian la comunicación entre una aplicación y el DEM como se muestra en la figura 6.4. No es necesario modificar las aplicaciones antes de compilación; cualquier aplicación que exponga servicios a través de un API conocido que pueda ser interceptado con tecnología de aspectos y de Web Services puede ser integrado a la infraestructura; en caso contrario la integración debe hacerse a nivel de datos. Las responsabilidades del CP son: 1) observación y generación de eventos, 2) recepción de notificaciones de eventos, y 3) procesamiento de eventos. Estas responsabilidades están agrupadas en tres sub-componentes: Observador Interno (OI), Receptor de Eventos (RE) y Procesador de Eventos (PE).

Observación y generación de eventos

El Observador Interno (OI) ofrece servicios para observar una aplicación y generar eventos, sus servicios son: 1) registro de reglas de observación y 2) observación de aplicaciones y generación de eventos lógicos a partir de una observación.

El registro de una observación incluye un servicio de un API a observar e interceptar, los parámetros del servicio y el tipo de evento. Los parámetros del evento generado son establecidos usando los parámetros del método interceptado. Cada parámetro del evento tiene un nombre, un parámetro del método, y opcionalmente un método ‘get’ para invocar en el parámetro del método; en este caso, el parámetro del evento es el objeto retornado por el método ‘get’.

La observación es lograda a través de intercepción de métodos por tecnología de aspectos. En versiones futuras, Eleggua generará la clase de intercepción del aspecto; actualmente entrelaza la clase interceptada, y recompila y hace deploy de la aplicación. Durante la ejecución, la invocación del método interceptado dispara la regla de observación. El componente genera un evento lógico usando los parámetros de entrada del método observado y redirige el evento al Procesador de Eventos (PE). En caso de que la aplicación externa no ofrezca un API que pueda ser interceptado es necesario desarrollar un componente que permita la integración a nivel de datos. Este componente debe ofrecer servicios para modificar el estado de los conceptos de dominio que se integran y notificaciones al CP en caso de cambios en estos conceptos.

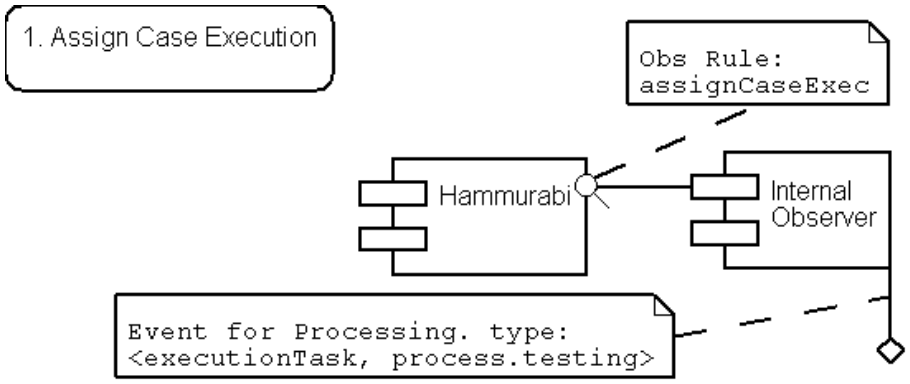

Figura 6.5 escenario de ejemplo para el Observador Interno

La figura 6.5 retoma el escenario de ejemplo presentado en la sección 5.2 para ilustrar la funcionalidad del componente Observador Interno (OI). El OI del Proxy de Cooperación que ofrece servicios a Hammurabi tiene registrada una regla de observación que intercepta la ejecución del servicio de asignación de ejecución de un caso de prueba. Un método 'get' es invocado en el objeto que recibe este método como parámetro para buscar el identificador del caso de prueba. Un evento lógico es generado con el tipo <*executionTask, process.testing*> y un parámetro con nombre *executionId*. El evento es enviado al Procesador de Eventos (PE)

Recepción de notificaciones de eventos

El Receptor de Eventos (RE) tiene como servicios principales 1) el registro de aplicaciones y suscripción a tipos de eventos, y 2) administración de notificaciones de eventos lógicos.

El componente interactúa directamente con el DEM usando el API que provee un componente consumidor local para registro de aplicaciones y suscripción a tipos de eventos. El RE recibe los tipos de eventos para suscribirse del Procesador de Eventos (PE). Con esta información el ER suscribe la aplicación en el DEM a cada tipo de eventos de interés.

Los eventos notificados son empujados (pushed) del DEM a este componente. En la siguiente versión, el componte generará un manejador de eventos a cargo de la recepción de eventos y su paso al EP para procesamiento.; actualmente esta clase debe ser creada manualmente.

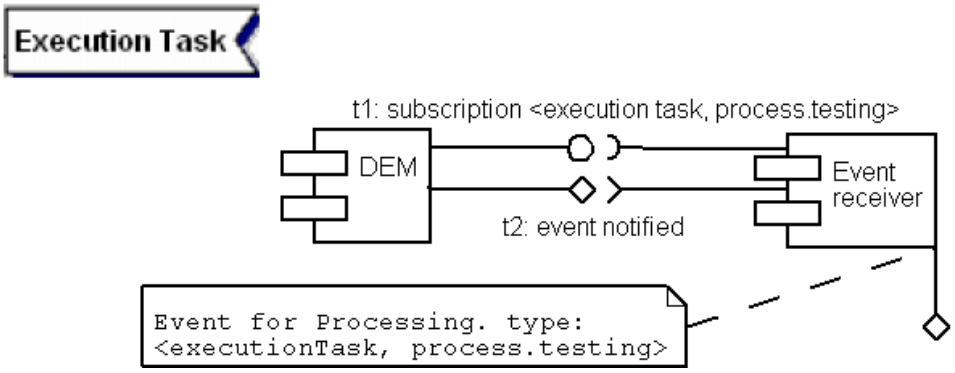

Figura 6.6 escenario de ejemplo Receptor de Eventos

La figura 6.6 retoma el escenario de ejemplo para ilustrar el rol del componente RE. En el CP que ofrece servicios a Cronos hay una regla ECA previamente registrada para eventos con tipo <executionTask, process.testing>; el RE se suscribió a este tipo de eventos con el DEM. Los eventos recibidos de este tipo son pasados al PE

Procesamiento de eventos

El Procesador de Eventos (PE) ofrece servicios para 1) declaración y 2) ejecución de reglas Evento Condición Acción (ECA).

Cada regla ECA especifica un tipo de eventos lógicos, una condición y una acción a ejecutar.

- La parte del evento de la regla declara el tipo de eventos que procesa la regla; más de una regla puede ser definida para un tipo de eventos.

- Las condiciones especifican parámetros con nombre que deben estar presentes en el evento lógico. Actualmente estas deben ser un método parte de la clase de acción, esto permite flexibilidad.

- Las acciones son código arbitrario a ejecutar. El código está en una clase que debe implementar una interfaz definida, el método de acción debe recibir como parámetro la instancia del evento lógico.

Al registrar una nueva regla para un nuevo tipo de eventos el componente se comunica con el Receptor de Eventos (RE) para que se suscriba al tipo.

La ejecución de una regla ECA sigue los siguientes pasos:

- El PE ofrece un API para procesar eventos lógicos del Receptor de Eventos (RE) y el Observador Interno (OI)

- Los eventos son comparados con las reglas registradas por tipo de evento y son filtrados posteriormente revisando los parámetros de acuerdo con la condición de la regla. Por cada regla que concuerde con el tipo de eventos la condición es evaluada, y si pasa, la acción es ejecutada.

- Las acciones reciben la instancia del evento lógico como parámetro. Si un error ocurre, durante la ejecución de una acción, un evento de error es generado y enviado al DEM. Las acciones pueden cambiar el estado de la aplicación, hacer búsquedas de información adicional y generar eventos lógicos.

Este componente ofrece servicios adicionales para facilitar la tarea de codificación de acciones. Estos servicios adicionales incluyen la ejecución de Web Services, y la creación y notificación de eventos lógicos al DEM. El componente debe tener acceso a los APIs de la aplicación, estos son referenciados por las acciones ejecutadas como reacción a los eventos.

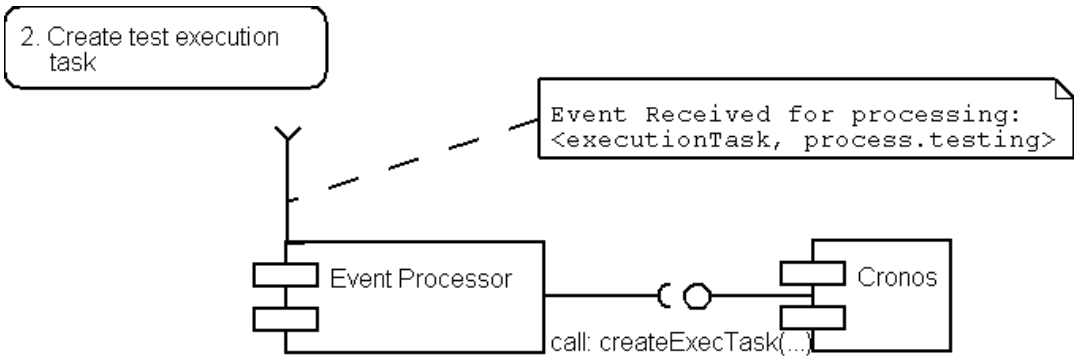

Figura 6.7 escenario de ejemplo para el Procesador de Eventos

La figura 6.7 ilustra el rol del componente PE en el escenario ejemplo. Una regla ECA registrada en el CP que ofrece servicios a Cronos ha sido registrada para procesar eventos con el tipo <*executionTask, process.testing*> y parámetros con nombres *projectId*, *executionId*, *caseId*, y *userId*. La acción de esta regla es ejecutada cuando un evento lógico con estas características es recibido por el EP. La acción llama un Web Service ofrecido por Hammurabi para completar la información necesaria para la tarea de ejecución de caso de pruebas, incluyendo los requerimientos asociados y una descripción detallada del caso de prueba. Finalmente, la acción crea una tarea de ejecución de caso de prueba en Cronos para que el usuario registre sus tiempos.

Otra regla ECA es declarada en el CP que ofrece servicios a Hammurabi, para eventos lógicos generados de observaciones, con tipo <*executionTask, process.testing*> y con un parámetro con nombre *executionId*. La acción solicita el identificador del proyecto, del caso de pruebas y del usuario responsable de la ejecución del caso de prueba usando servicios que provee Hammurabi a través de Web Services. La acción incluye como parámetros del evento lógico *projectId, caseId* y *userId* y lo notifica al DEM.

6.3 Interacción de los Componentes

La figura 6.8 muestra un diagrama de secuencia con la interacción abstracta de los componentes de Eleggua. El usuario llama un método en la aplicación, el CP que ofrece servicios a esta aplicación intercepta la ejecución y envía el evento al componente productor del DEM local. El componente productor persiste el evento y lo notifica al componente despachador del DEM, posiblemente remoto, que persiste el evento y lo notifica a todos los componentes consumidores remotos. El componente consumidor inicia una transacción por cada aplicación suscriptora al tipo de evento; esta transacción envía el evento al CP que ejecuta la regla ECA asociada, esta usualmente efectúa cambios de estado sobre otra aplicación.

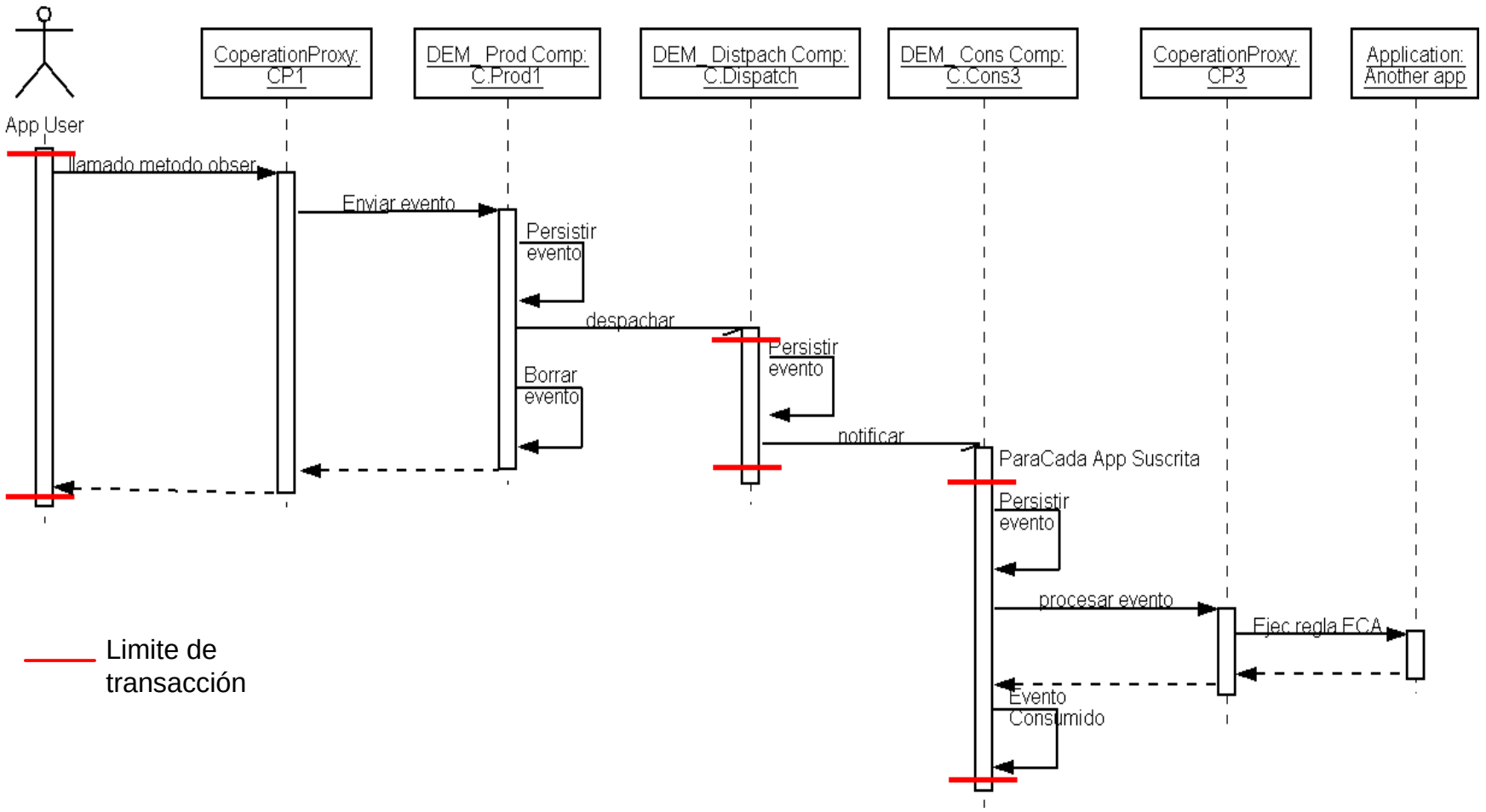

Figura 6.8 Interacción de los componentes

6.4 Componente Monitor y Administrador de Eleggua

Un componente de monitoreo y administración ha sido desarrollado como parte de Eleggua. Este es un componente basado en tecnología JMX que ofrece servicios para administración monitoreo de:

- Aplicaciones: El componente administra aplicaciones externas conectadas a Eleggua. La notificación de eventos a una aplicación puede ser detenida durante ejecución; los eventos recibidos para la aplicación durante este tiempo son enviados al reiniciar la notificación
- Tipos de eventos: Ofrece servicios para listar, crear o borrar tipos de eventos
- Eventos: Provee servicios para crear trazas de eventos y búsquedas de eventos con varios filtros
- Reglas ECA: Ofrece servicios para activar y desactivar reglas ECA durante ejecución
- Suscripciones: Ofrece servicios para creación y eliminación de suscripciones
- Configuración de propiedades de componentes: Provee servicios de búsqueda y modificación de propiedades de los componentes

Este componente, adicionalmente, genera alarmas para el administrador del proceso en caso de errores durante la ejecución. La compensación es manual; si un error ocurre durante el escenario de ejemplo, por ejemplo, si no hay un desarrollador trabajando en el proyecto del caso de prueba, el administrador recibe una alarma e informa al líder del proyecto que tome acciones correspondientes.

7 Implementación

Este capitulo presenta los detalles de implementación de la infraestructura presentada anteriormente. Primero se detalla el Middleware de Eventos Distribuidos y los detalles implementación de los sub-componentes. Después se profundiza en la implementación del Proxy de Cooperación. Para cerrar el capítulo se presentan los detalles de la distribución de los componentes y algunos detalles de la implementación del monitor de la infraestructura.

7.1 Middleware de Eventos Distribuidos (DEM)

El DEM está compuesto de tres tipos de sub-componentes distribuidos. El componente Despachador es el nodo central de comunicación entre los otros componentes. La comunicación entre los componentes es distribuida. La figura 7.1 presenta los componentes principales del DEM y su interacción.

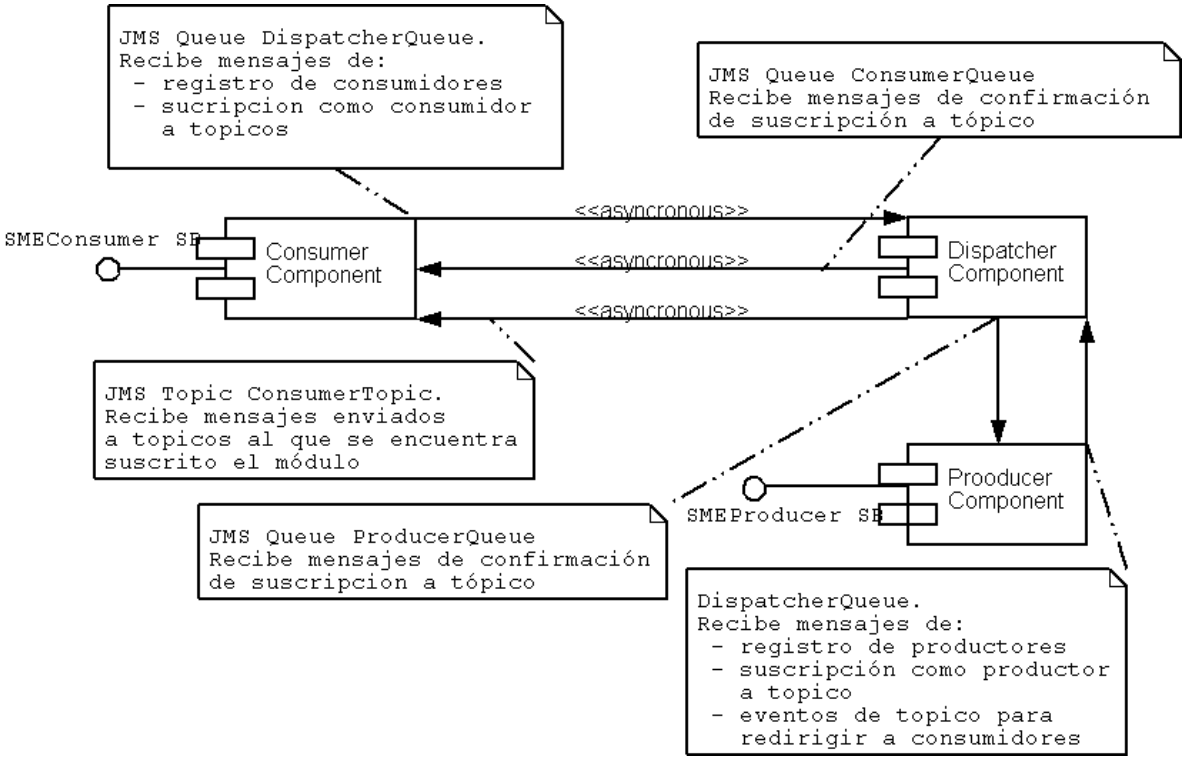

Figura 7.1 Diagrama de Componentes del DEM

El componente despachador es implementado como un componente central con varios componentes consumidores y productores comunicándose a través de tecnología JMS. Conexiones punto a punto son establecidas entre los componentes para registro de aplicaciones y suscripción, y para mensajes administrativos como cancelación de tópicos. Una conexión publicador/suscriptor es establecida para notificación de eventos; el despachador publica en esta conexión y todos los componentes consumidores reciben las notificaciones. Todos los eventos son envueltos en mensajes JMS incluyendo eventos lógicos y eventos administrativos. El proveedor de JMS es JBossMQ [15].
Los componentes del DEM están implementados usando EJBs de sesión (SBeans) sin estado, entidad (EBeans) y message driven (MDBeans).

7.1.1 Componente Despachador

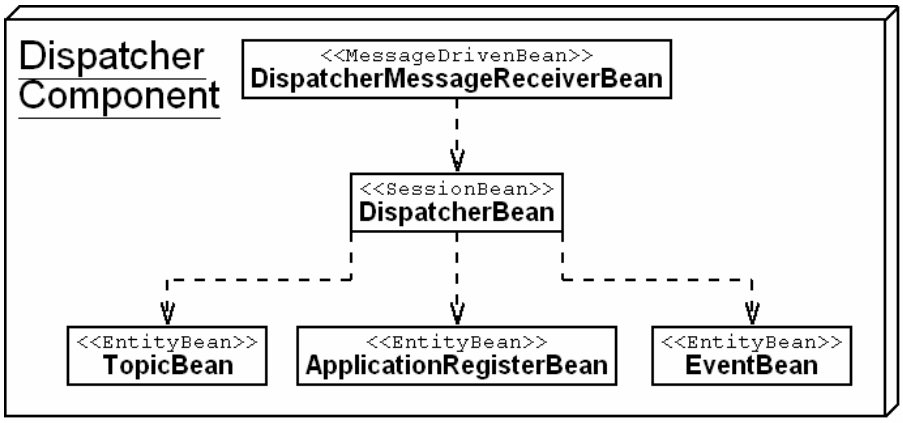

Figura 7.2 EJBs del Componente Despachador

El componente despachador se encarga de recibir mensajes JMS con solicitudes y distribuir eventos y respuestas a solicitudes a los componentes consumidor y productor. Los principales EJBs del componente y sus dependencias son presentados en la figura 7.2.

El principal medio de comunicación con los otros componentes es DispatcherMessageReceiverBean, este es un MDBean que se encarga de recibir las solicitudes y los eventos de los otros componentes. Al recibir un mensaje JMS el MDBean dependiendo del tipo de mensaje pasa la solicitud al DispatcherBean.

El DispatcherBean es un SBean que contiene toda la lógica de negocio del componente. Este SBean ofrece servicios para registro de tipos de eventos, registro de aplicaciones, creación y cancelación de suscripciones y envío de eventos. El SBean se apoya en tres EBeans que se encargan de persistir los conceptos principales que maneja el componente.

El TopicBean administra los tipos de eventos, el ApplicationRegisterBean administra los registros de aplicaciones y las suscripciones, finalmente, el EventBean administra los eventos. El componente recibe solicitudes de suscripción de los componentes productor y consumidor, al recibir la solicitud se persiste la suscripción y se envía respuesta en caso de ser componente consumidor.

Al recibir un evento lógico a despachar de un componente productor se revisa que la aplicación productora esté suscrita al tipo de eventos, en este caso se persiste el evento y se envía por el canal publicar/suscribir a todos los componentes consumidores.

7.1.2 Componente Consumidor

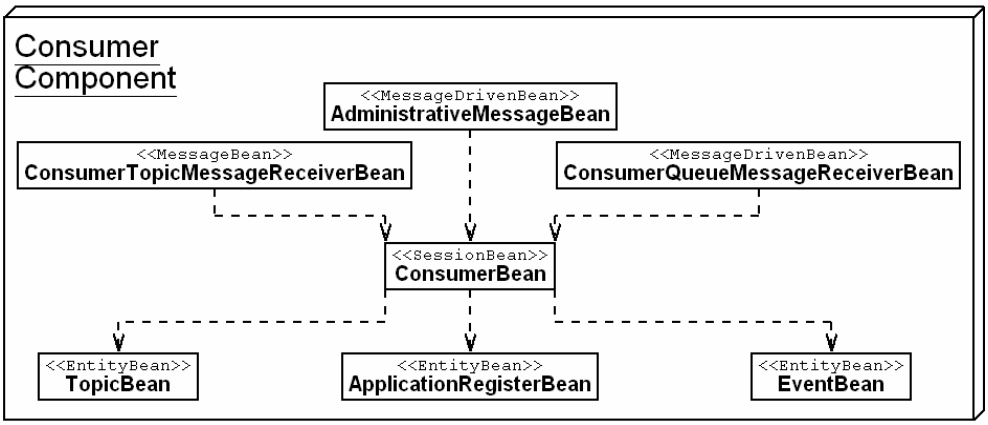

Figura 7.3 EJBs del Componente Consumidor

Este componente es el principal punto de interacción de una aplicación para consumir eventos del DEM. Los principales EJBs del componente y sus dependencias son presentados en la figura 7.3. El componente tiene un SBean principal que ofrece servicios de negocio a la aplicación externa y servicios locales a los MDBeans receptores de mensajes JMS.

El componente cuenta con 3 MDBeans con responsabilidades diferentes. El ConsumerTopicMessageReceiverBean es un MDBean de tipo publicar/suscribir que recibe los eventos lógicos enviados por el componente despachador. El ConsumerQueueMessageReceiverBean es un MDBean que recibe mensajes de respuesta a solicitudes hechas al componente despachador. Finalmente, el AdministrativeMessageBean MDBeans se encarga de recibir eventos administrativos como la eliminación de un tipo de eventos.

El componente además cuenta con EBeans iguales a los del componente despachador para la persistencia local de tipos de eventos, registros de aplicaciones y suscripciones, y eventos lógicos.

Al solicitar una suscripción el componente despacha una solicitud a través de un mensaje JMS al componente despachador. El componente registra las suscripciones solamente al recibir respuesta a la solicitud de suscripción por parte del componente despachador.

El registro de aplicaciones identifica a una aplicación, sus suscripciones y los eventos que recibe. Adicionalmente este registro puede incluir opcionalmente una clase manejadora de eventos (IEventHandler).

Al recibir un evento notificado por el ConsumerTopicMessageReceiverBean se persiste una copia del evento por cada aplicación consumidora. Si la aplicación consumidora declaró un manejador de eventos el evento es pasado a este manejador. Los eventos son marcados como consumidos al ser pasados a este manejador o cuando sean explícitamente solicitados a través del ConsumerBean.

7.1.3 Componente Productor

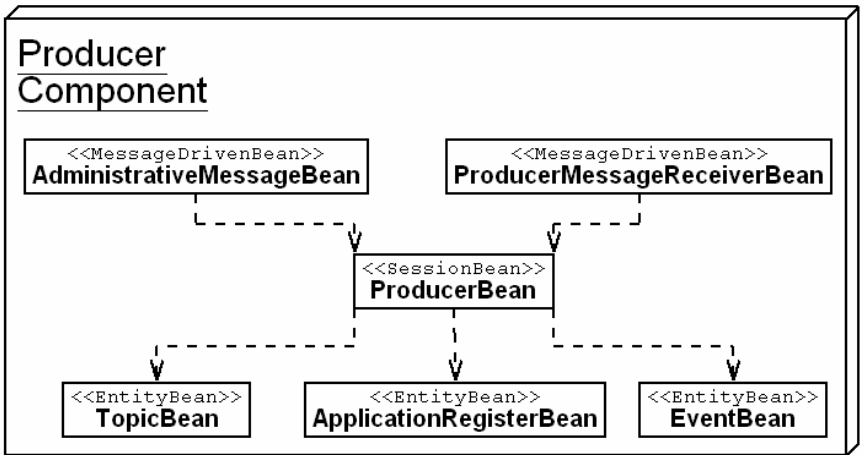

Figura 7.4 EJBs del Componente Productor

Este componente ofrece servicios para producción de eventos en el DEM. Los principales EJBs del componente y sus dependencias son presentados en la figura 7.4. El componente ofrece sus servicios a través de un SBean, el ProducerBean, este ofrece servicios de negocio a la aplicación externa y servicios locales a los MDBeans receptores de mensajes JMS.

El componente cuenta con dos MDBeans para recepción de mensajes JMS, ambos son de comunicación punto a punto. El ProducerMessageReceiverBean recibe respuestas a solicitudes hechas al componente despachador como suscripción a tópicos y registro de aplicaciones. El AdministrativeMessageBean recibe mensajes administrativos como la eliminación de un tipo de eventos.

Al igual que los otros dos componentes este tiene una serie de EBeans que administran la persistencia de los objetos localmente. El comportamiento del TopicBean y del ApplicationRegisterBean es igual al del componente consumidor. El EventBean es usado en este componente para persistir eventos en caso de fallas o caídas en el sistema. La lógica para recuperación de fallos aún no ha sido implementada, sin embargo este mecanismo permite que los datos no se pierdan y así poder mantener la consistencia del proceso. Este EJB almacena los eventos antes de ser enviados al DEM, en caso de que el envío sea exitoso el evento es eliminado. Esto permite ofrecer servicios para discriminar que eventos fueron producidos y no pudieron ser enviados.

7.2 Proxy De Cooperación (CP)

El CP es el componente que intermedia la interacción entre el DEM y las aplicaciones externas. El componente esta dividido en tres sub-componentes. El CP administra y ejecuta las reglas ECA y de observación.

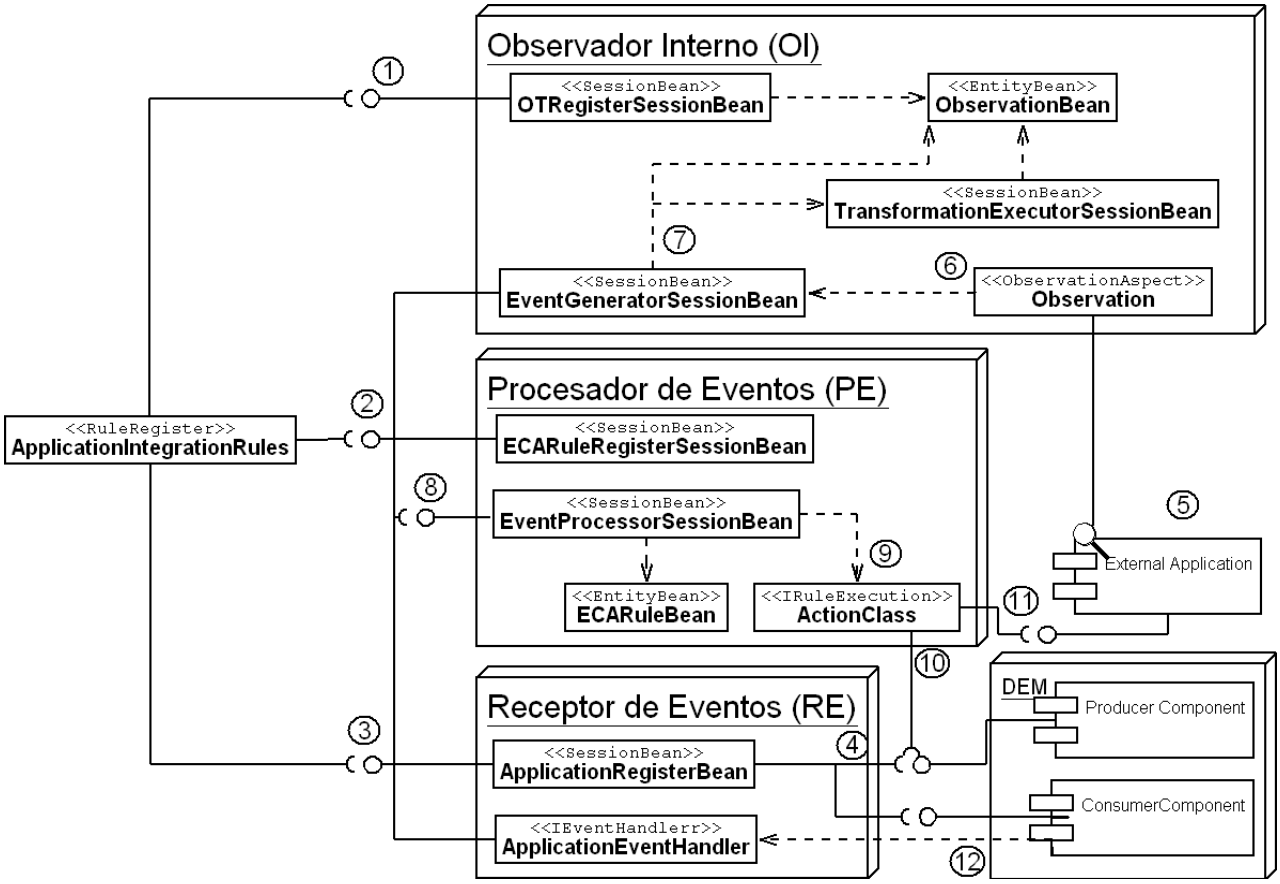

Figura 7.5 Diagrama de componentes del CP

La figura 7.5 presenta un detalle de los sub-componentes del CP y su interacción con otros componentes de Eleggua. Los puntos numerados corresponden a las interacciones entre los componentes que serán presentadas a continuación para explicar los detalles de implementación de los sub-componentes. Los sub-componentes del CP están implementados en una plataforma J2EE usando EJBs de sesión (SBeans) y entidad (EBeans). La figura 7.5 muestra los EJBs de los componentes Observador Interno (OI), Procesador de Eventos (PE) y Receptor de Eventos (RE).

7.2.1 Observador Interno (OI)

El Observador Interno ofrece servicios para registro y ejecución de reglas de observación. La figura 14 en el punto 1 muestra un SBean llamado OTRegisterSessionBean para el registro de reglas de observación; el registro de estas reglas es persistido por el EBean ObservationBean.

La observación durante ejecución (Fig. 7.5 punto 5) es hecha por medio de intercepción de aspectos usando AspectJ; esta tecnología impone restricciones sobre el tipo de plataforma que se puede observar, esta y otras restricciones son discutidas posteriormente. Adicionalmente, esta

tecnología impone como restricción que las clases deben ser entrelazadas usando las fuentes .java antes de ejecución. El componente automatiza este proceso a través de un build de Ant; este entrelaza, regenera el código y hace deploy de las aplicaciones externas.

Durante ejecución la clase de Observación es llamada por la intercepción de servicios; esta solicita servicios del EventGeneratorSessionBean para obtener los parámetros de la observación (Fig. 7.5 punto 6), la observación llena los valores de estos parámetros y pasa la observación al SBean.

El EventGeneratorSessionBean recibe la observación y se la pasa al TransformationExecutorSessionBean, este transforma la observación en un evento lógico (Fig. 7.5 punto 7) y lo devuelve al EventGeneratorSessionBean. Finalmente este SBean pasa el evento generado al Procesador de Eventos (PE) (Fig. 7.5 punto 8).

7.2.2 Procesador de Eventos (PE)

El procesador de eventos ofrece servicios de registro de reglas ECA para procesamiento de eventos (Fig. 7.5 punto 2) a través del ECARuleRegisterSessionBean. Este SBean usa servicios del EBean ECARuleBean para persistencia de las reglas. El registro de una regla ECA incluye el tipo de evento, y una clase de acción que implemente IRuleExecution. Esta interfaz expone dos métodos: checkCondition() y execute(); checkCondition recibe como parámetro un evento lógico y retorna verdadero o falso, execute recibe como parámetro un evento lógico, este método se encarga del procesamiento del evento. Este esquema permite definir condiciones y acciones arbitrarias sobre los eventos a procesar. Adicionalmente a estos parámetros se especifica si los eventos lógicos a procesar son observados localmente o notificados. Cuando se registra una regla ECA para un evento notificado el componente se comunica con el componente Receptor de Eventos para que se suscriba al tipo de eventos.

El componente recibe, via push, eventos para procesar del componente OI y RE a través del EventProcessorSessionBean (Fig. 7.5 punto 8), los eventos recibidos desde el OI son caracterizados como observados, mientras los recibidos del RE son notificados. Esta diferenciación permite que los eventos observados localmente puedan ser notificados a la infraestructura pero procesados en caso de ser notificados de vuelta hacia el componente que los produjo.

Al recibir un evento el componente busca todas las reglas para el tipo de eventos del evento lógico usando servicios del ECARuleBean. Posteriormente busca por introspección la clase de acción; al procesar un evento el EventProcessorSessionBean pasa el evento hacia la clase de acción (Fig. 7.5 punto 9). El SBean primero ejecuta el método checkCondition de la clase de acción, en caso de retornar verdadero llama el método execute(). Las acciones pueden modificar el estado de una aplicación a través de su API (Fig. 7.5 punto 11), enviar eventos al DEM (Fig. 7.5 punto 10) y solicitar servicios por Web Services o a través del API de otras aplicaciones.

7.2.3 Receptor de Eventos (RE)

El componente Receptor de Eventos (RE) tiene como responsabilidad administrar los registros de aplicaciones y sus suscripciones y la recepción de notificaciones del DEM. Este componente cuenta con un SBean que ofrece los servicios de registro de aplicaciones y suscripciones: ApplicationRegisterBean. Este EJB es usado para suscripción a tipos de eventos que son producidos por las acciones (Fig. 7.5 punto 3). Adicionalmente el componente PE usa servicios de este SBean para registrarse a tipos de eventos de reglas ECA para eventos notificados. El componente suscribe para estos eventos producidos usando los servicios del componente productor del DEM y para eventos notificados usando los servicios del componente consumidor del DEM (Fig. 7.5 punto 4).

Este componente adicionalmente incluye las clases manejadoras de eventos que implementan IEventHandler. Una clase de estas existe por cada aplicación que consume al menos un tipo de eventos. La clase manejadora de eventos es registrada por el SBean del componente al registrar una aplicación. Al recibir notificaciones para un tipo de eventos de interés la clase de acción es llamada y el evento es pasado (Fig. 7.5 punto 12). La clase manejadora de eventos pasa el evento al Procesador de Eventos (Fig. 7.5 punto 8).

7.3 Deployment de Eleggua

Eleggua es una infraestructura de componentes distribuidos; la comunicación entre los componentes distribuidos, detallada anteriormente, es usando tecnología JMS. Eleggua está construido sobre una plataforma J2EE, específicamente el contenedor de aplicaciones JBoss 3.2.3. El siguiente diagrama presenta la distribución de los componentes de Eleggua en varios nodos JBoss distribuidos. Este esquema es una sola de las posibilidades, no es estrictamente necesario que los componentes se encuentren distribuidos y un Proxy de Cooperación puede ofrecer servicios a varias aplicaciones. Esta arquitectura permite un nivel bueno de escalabilidad.

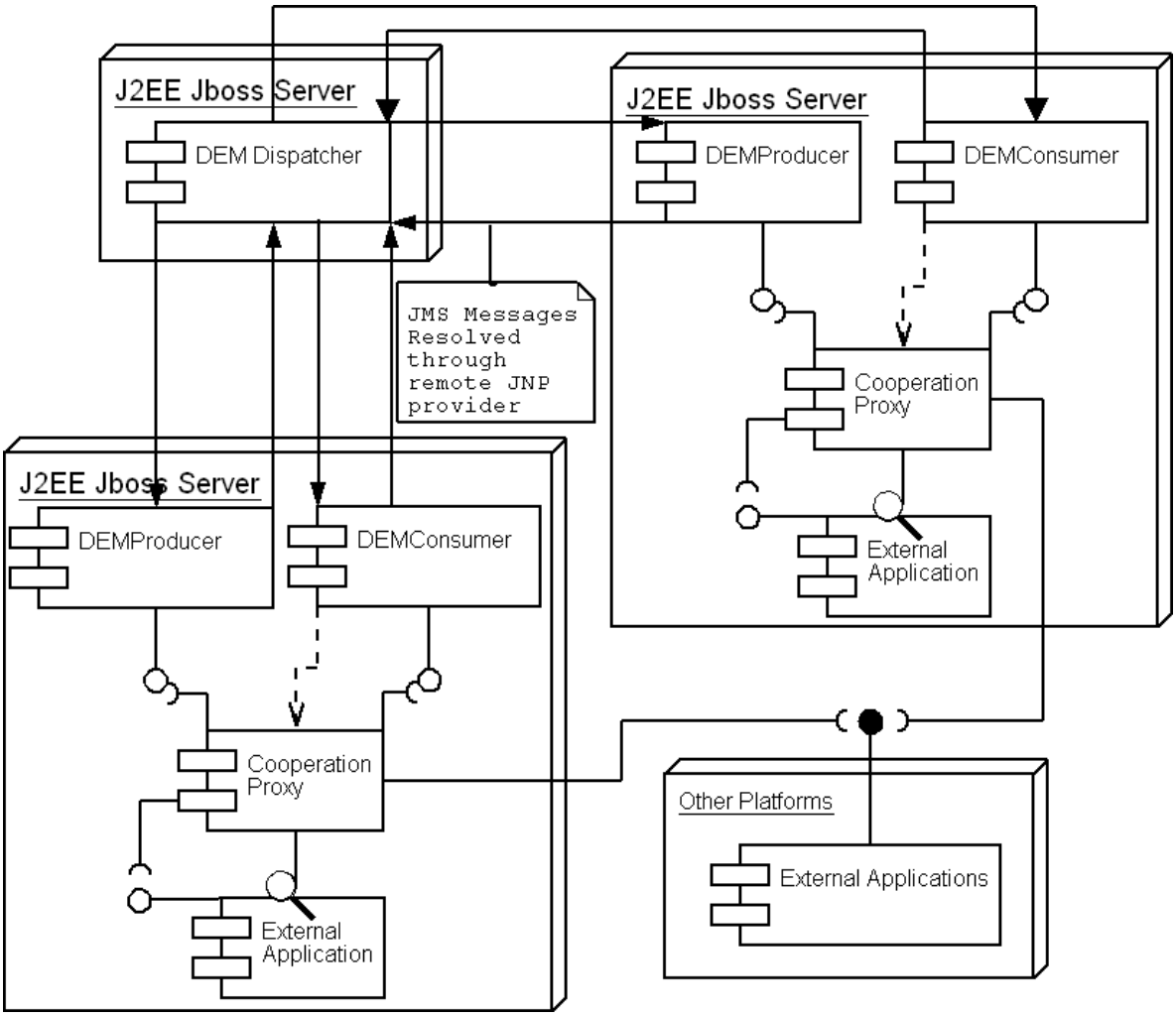

Figura 7.6 diagrama de nodos de distribución de Eleggua

Los componentes del DEM están distribuidos en varios nodos JBoss. El componente despachador es único y varios componentes productor y consumidor del DEM lo acceden remotamente. El CP debe estar en el mismo nodo que los componentes productor y consumidor del DEM y la aplicación externa a la que ofrece servicios. Un CP observa aplicaciones locales y usa sus servicios ofrecidos por APIs de SBeans. Finalmente otras aplicaciones en plataformas diferentes tienen son accedidas a través de Web Services o a través de extensiones del CP que permitan integración a nivel de datos.

7.4 Instrumentación Eleggua

La figura 7.7 presenta la instrumentación de los componentes de Eleggua para el monitor JMX. La consola de monitoreo está actualmente en proceso de diseño Esta sección describe los EJBs Managed Beans (MBeans) que instrumentan los servicios de administración y monitoreo y su relación con los componentes de Eleggua. La interfaz de acceso a estos servicios para el administrador del proceso está en diseño. Sin embargo estos servicios son accesibles a través de una consola estándar de JMX. A continuación se detalla los servicios de cada MBean.

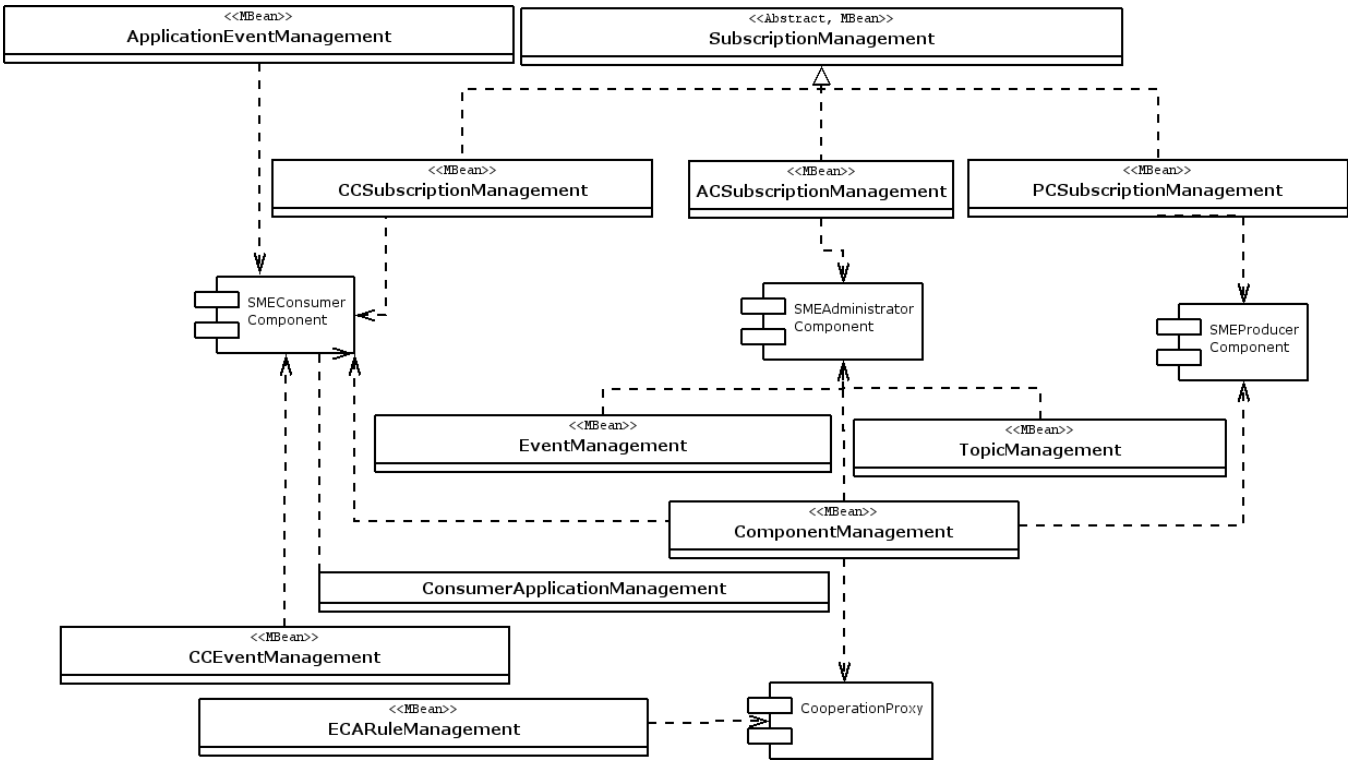

Figura 7.7 diagrama de instrumentación de Eleggua

7.4.1 TopicManagement MBean

Este MBean ofrece servicios simples de administración de tópicos. Se comunica directamente con el SBean DispacherBean del componente despachador. Este MBean ofrece los siguientes servicios:

- getTopics busca todos los tópicos en el sistema devuelve una colección de TopicData
- getTopicsByContext busca todos los tópicos dado un contexto

- createTopic crea un nuevo tópico en el sistema dado su nombre y contexto
- deleteTopic borra un tópico del sistema. Este servicio adicionalmente elimina todas las suscripciones al tópico eliminado en todos los componentes.

7.4.2 ApplicationEventManagement MBean

Este MBean ofrece servicios simples de administración de eventos de aplicaciones. Se comunica directamente con el SBean ConsumerBean del componente CC. Este MBean ofrece los siguientes servicios:

- stopEventNotification (byTopic): detiene la notificación de todos los eventos a una aplicación (de un tópico
- restartEventNotification (byTopic): reinicia la notificación de todos los eventos a una aplicación (de un tópico).
- blockEventNotification: permite el bloque de la notificación de un evento a una aplicación (p.e. si está detenido el envío de eventos de un tópico para que al reiniciar no sea enviado cierto evento).

7.4.3 ComponentManagement MBean

Este MBean ofrece servicios simples de administración de Componentes. Este MBean interactúa con todos los componentes. Este MBean interactúa con la clase PropertiesReader que provee servicios de acceso a propiedades para todos los componentes. El MBean permite establecer nuevas propiedades, eliminar propiedades, modificarlas y listarlas.

7.4.4 ConsumerApplicationManagement MBean

Ofrece servicios para listar y modificar configuración de las aplicaciones atendidas por un componente consumidor del DEM. Este MBean interactúa directamente con el EBean de ApplicationRegistration. Este MBean ofrece los siguientes servicios:

- getApplications devuelve una colección de ApplicationRegistrationData con la información de todas las aplicaciones registradas localmente.
- getApplication: devuelve la información completa de una aplicación
- setApplicationEventHandler: recibe como parámetro adicionalmente el identificador de la aplicación (String). Modifica la clase que recibe eventos para una aplicación este servicio es solicitado directamente al EBean de ApplicationRegistration.

7.4.5 ProducerApplicationManagement MBean

Ofrece servicios para listar las aplicaciones atendidas por un componente productor del DEM. Este MBean interactúa directamente con el EBean de ApplicationRegistration. Este MBean ofrece los siguientes servicios:

- getApplications devuelve una colección de ApplicationRegistrationData con la información de todas las aplicaciones registradas localmente.
- getApplication: devuelve la información completa de una aplicación

7.4.6 ECARuleManagement Mbean

Este MBean ofrece servicios de administración de reglas ECA en un componente CP. Interactúa directamente con el EBean ECARuleBean del CP. Este MBean ofrece los siguientes servicios:

- getRulesByApplication: Devuelve todas las reglas ECA para una aplicación
- disableECARule: Desactiva una regla ECA, los eventos recibidos son ignorados por la regla y descartados
- enableECARule: Activa una regla ECA, los eventos recibidos son procesados.

7.4.7 EventManagement MBean

Este MBean ofrece servicios de consulta de eventos del DEM. Se comunica directamente con el SBean DispatcherBean. Ofrece servicios para búsqueda de eventos por identificador único, por tópico, por contexto, por productor y en un rango de fechas.

7.4.8 CCEventManagement MBean

Este MBean ofrece como único servicio devolver todas las copias locales de un evento. Junto con los servicios de EventManagementBean permite hacer una traza completa de un evento en toda la infraestructura.

7.4.9 SubscriptionManagement MBeans

Estos MBeans permiten administrar las suscripciones de aplicaciones en la infraestructura. Se implementan con tres MBeans que interactúan con cada componente del DEM independientemente. Todos ofrecen los mismos servicios, pero su dependencia de varios componentes implica que debe haber varios MBeans que implementen los servicios. Estos servicios interactúan con el SBean principal de cada componente. Este MBean ofrece los siguientes servicios:

- getSubscriptions (ByApplication, Type, Topic): devuelve una lista de las suscripciones en el sistema (por aplicación, tipo: productor/consumidor, por tópico)
- createSubscription: crea una nueva suscripción (propaga en el sistema)
- deleteSubscription: elimina una suscripción (propaga en el sistema)

7.5 Restricciones

Algunas decisiones de diseño y tecnologías imponen ciertas restricciones sobre la implementación de Eleggua. Adicionalmente, algunos factores relevantes para toda plataforma de integración son considerados restricciones pues aún no están implementados y no hacen parte de la tesis debido al alcance previsto. Entre estos están las restricciones de la plataforma para observación por aspectos que solo permite observar aplicaciones en código java. Adicionalmente la sincronización de los mensajes pasados entre los componentes del DEM no incluye consideraciones adicionales a las previstas por JMS, por esta razón se podría dar el caso de que un componente recibiera mensajes en un orden diferente a como fueron creados. Tercero, el sistema no es completamente tolerante a fallas, este aspecto actualmente está en desarrollo y es crítico para el uso de la infraestructura en un ambiente real. Finalmente, otra restricción impuesta por JMS para la comunicación es que esta necesita acceso a puertos que usualmente están restringidos por un firewall.

8 Validación de resultados

8.1 Contexto

El proyecto fue validado en Heisohn Software House, una empresa de software que en la actualidad afronta los problemas del desarrollo de software globalizado. La validación fue hecha dentro del contexto del proyecto Testing en Ambientes Globalizados con el soporte de la Universidad de los Andes y el Instituto Colombiano para el Desarrollo de la Ciencia y la Tecnología Francisco José de Caldas" - COLCIENCIAS.

El proyecto tiene como objetivo definir e implementar un proceso de pruebas en esta empresa; el proyecto incluye la construcción de nuevas herramientas y su integración con herramientas legado. A continuación se presenta una parte del proceso definido y las herramientas que son usadas en el proceso. En las secciones posteriores se presentan las reglas de integración y los detalles de implementación del piloto y los resultados del piloto.

8.1.1 Proceso

Se seleccionó el proceso de pruebas de software distribuidas definido para el proyecto “Testing en Ambientes Globalizados” para ser definido e integrar las aplicaciones que dan soporte a través de Eleggua . Las reglas de negocio que describen este proyecto están divididas en los subprocesos de este proceso de pruebas. La figura 8.1 presenta el proceso.

Diseño del plan de pruebas

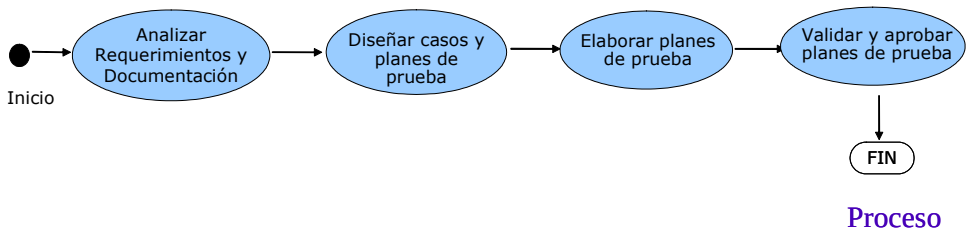

Figura 8.1 proceso de diseño de plan de pruebas

Ejecución Plan de Pruebas

Este proceso corresponde a las actividades necesarias para ejecutar un plan de pruebas. Estas actividades incluyen preparar el ambiente de pruebas, ejecutar los planes y en caso de no aprobar algún plan pasar al proceso de registro y corrección de defectos. El proceso es presentado a continuación en la figura 8.2.

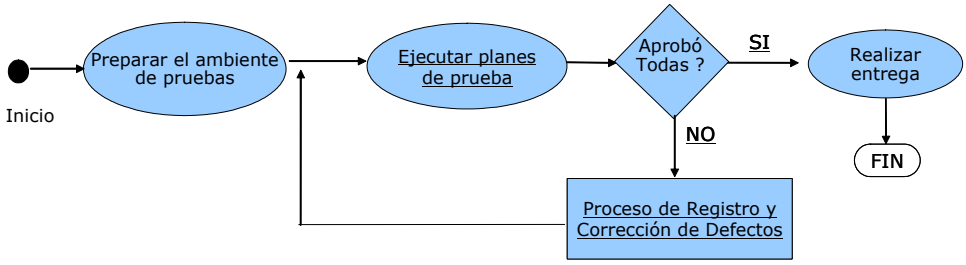

Figura 8.2 proceso de ejecución de plan de pruebas

Registro y Corrección de Defectos

Este proceso representa el registro y corrección de defectos encontrados en una tarea de ejecución de caso de pruebas rechazada. El proceso es presentado a continuación en la figura 8.3.

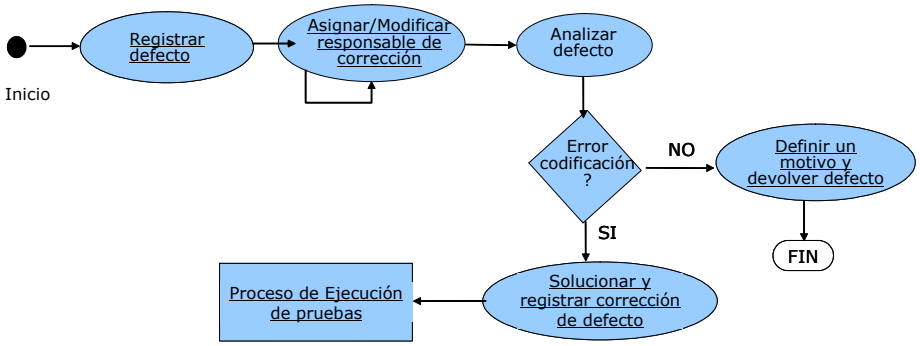

Figura 8.3 proceso de registro y corrección de defectos

8.1.2 Aplicaciones

El proceso presentado cuenta con el soporte de varias herramientas. Hammurabi es usada en el proceso para crear planes de prueba y registrar reportes de defectos. Permite administración y consulta de los planes de prueba. Los defectos y mejoras encontrados durante las diferentes ejecuciones son registrados en Hammurabi, además permite administrar de los defectos encontrados durante las ejecuciones de los planes de prueba. Hammurabi es una aplicación J2EE que fue desarrollada anteriormente a la iniciación de este proyecto. Los servicios observados y llamados por las acciones son ofrecidos por APIs de SBeans.

Cronos es una aplicación de agenda y rastreo del proceso; provee servicios para registrar tareas planeadas y registrar tiempo gastado en las tareas. La planificación para la corrección de defectos, una vez terminado el reporte de defectos es responsabilidad de esta herramienta. Permite crear tareas asociadas a cada una las actividades de ejecución de planes de prueba y corrección de defectos. Cronos ofrece servicios a un usuario para visualizar las actividades y tareas asignadas para la corrección de los defectos. Cronos es una aplicación J2EE desarrollada en el contexto del proyecto. Los servicios observados y llamados por las acciones son ofrecidos por APIs de SBeans.

Vernue, una aplicación de control global de proyectos es usada para calcular pagos de empleados con base al tiempo trabajado en las actividades de los proyectos. Esta herramienta le facilita al responsable del proyecto conocer el estado global, el estado de las pruebas y determinar tiempos de finalización y valor ganado dentro del cronograma del proyecto. Vernue es una aplicación legado desarrollada en AS/400. La integración con Vernue es hecha a nivel de datos. El estado de

los datos de Vernue relevante para la interacción es accedido directamente a la base de datos a través de un componente simple de registro y verificación periódica de datos desarrollada en el contexto del proyecto en una plataforma J2EE.

8.2 Reglas de negocio de la integración

El proceso anterior presenta una serie de actividades que son soportadas por las herramientas. Es necesario integrar las aplicaciones para los subprocesos presentados y adicionalmente para el registro de tiempos de actividades y registro de tiempos de actividades no planeadas. Esto es necesario para calcular reportes de tiempos sobre el proceso de pruebas.

La figura 8.4 muestra un listado de algunas de las reglas ECA definidas para el proceso. Estas serán referenciadas y explicadas en las siguientes secciones.

Cons	Regla	Aplicación	Módulo	Tipo	Tópico	Contexto
1	Reg Resp Caso de Prueba	Arquimedes	Test Case	Productor	assignCases	process.testing
2	Rechazar Caso de Prueba	Arquimedes	Test Case	Productor	testCaseRejected	process.testing
3	Cambio Resp Defecto	Arquimedes	Test Case	Productor	changeDefectResponsible	process.testing
4	Reg Resp Caso de Prueba	Cronos	Test Case	Consumidor	assignCases	process.testing
5	Rechazar Caso de Prueba	Cronos	Test Case	Consumidor	testCaseRejected	process.testing
6	Cambio Resp Defecto	Cronos	Test Case	Consumidor	changeDefectResponsible	process.testing
7	Cerrar tarea Ejec. Caso de Prueba	Cronos	Test Case	Productor	closeCaseExecActivity	process.testing
8	Registrar tiempo Ejec Caso de pru	Arquimedes	Test Case	Consumidor	closeCaseExecActivity	process.testing
9	Resigrar tiempo de tarea	Vernue	Test Case	Consumidor	closeCaseExecActivity	process.testing
10	Tasklog Tarea Planeada	Cronos	Tasklog	Productor	logRequirement	process.tasklog
11	Tasklog Tarea No Plan	Cronos	Tasklog	Productor	logNotPlanned	process.tasklog
12	Tasklog Tarea Servicio	Cronos	Tasklog	Productor	logServiceRequest	process.tasklog
13	Tasklog Tarea Planeada	Vernue	Tasklog	Consumidor	tasklogCreate	process.tasklog
14	Tasklog Tarea No Plan	Vernue	Tasklog	Consumidor	notPlannedTasklogCreate	process.tasklog
15	Tasklog Tarea Servicio	Vernue	Tasklog	Consumidor	serviceRequestTasklogCrea	process.tasklog

Figura 8.4 Tabla de reglas de integración para el proceso piloto

A continuación se presenta en detalle la integración para estos subprocesos. Primero se presenta un diagrama para cada sub-proceso con la descripción del proceso, las reglas de integración enunciadas y las aplicaciones que afectan. Posteriormente se detallan las reglas de cada subproceso relacionándolas con la tabla 8.4.

8.2.1 Tarea de ejecución de planes de prueba

La figura 8.5 presenta el grupo de actividades que hacen parte del proceso de ejecución de planes de pruebas. En estas el usuario que hace el plan de pruebas interactúa con Hammurabi, mientras que el usuario que registra la ejecución interactúa con Cronos.

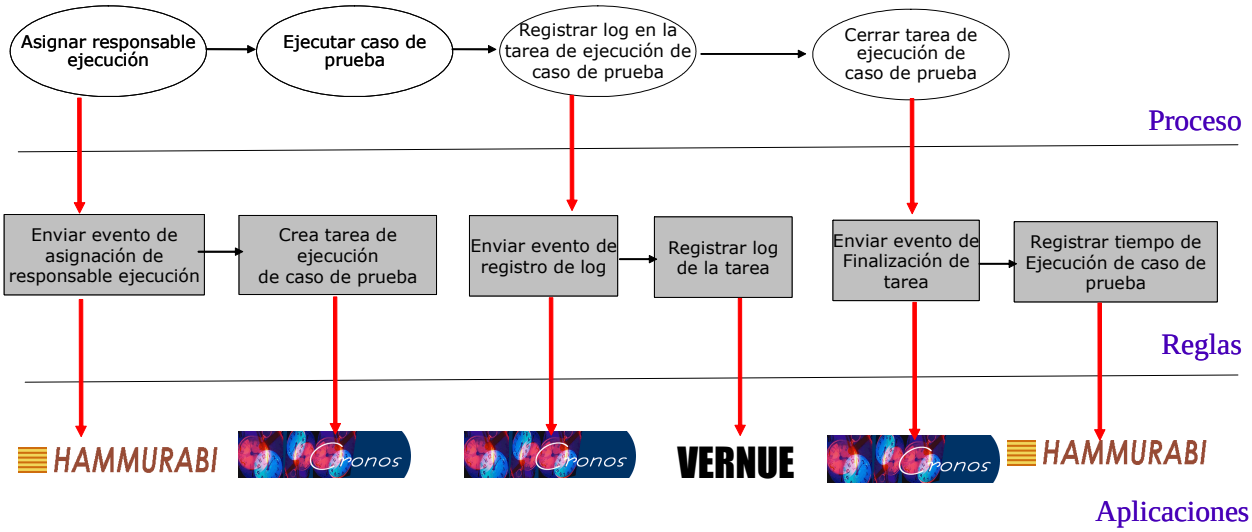

Figura 8.5 sub-proceso de tarea de ejecución de planes de prueba

- Enviar evento de asignación de responsable: Se observa el servicio de asignación de responsable de ejecución, la observación genera un evento con el identificador del usuario, del proyecto y de la ejecución. La regla ECA 1 Registrar Responsable de Caso de Prueba para Hammurabi se encarga de procesar este evento. Esta regla produce un evento con toda la información necesaria para la creación de la tarea para el tipo de eventos <*assignCases, process.testing*>.

- Crear tarea de ejecución de caso de prueba: Se recibe la notificación de un evento del tipo <*assignCases, process.testing*> en el CP que ofrece servicios a Cronos. Este evento es procesado por la regla ECA 4 Registrar Responsable de Caso de Prueba para Cronos. Esta regla ejecuta servicios del API de Cronos para crear una nueva tarea de tipo ejecución de caso de prueba.

- Enviar evento de registro de *log*: Se observa en Cronos el servicio de registro de tiempo, esta observación genera un evento con el identificador del proyecto y del registro. Este evento es procesado por la regla ECA 10 Tasklog Tarea Planeada para Cronos. La acción completa información sobre la tarea y envía un evento del tipo <*logRequirement, process.tasklog*>.

- Registrar *log* de tarea: Se recibe la notificación de un evento del tipo <*logRequirement, process.tasklog*> en el CP que ofrece servicios a Vernue. Este evento es procesado por la regla ECA 10 Tasklog Tarea Planeada que crea la tarea y registra su tiempo en Vernue.

- Enviar evento de Finalización de Tarea: El usuario registra la finalización de la tarea de ejecución en Cronos. Eleggua observa este servicio y genera un evento con la identificación de la tarea. La regla ECA 7 es activada por este evento; esta adiciona información de tiempo de la tarea y de la ejecución asociada a la tarea y envía un evento del tipo <*closeCaseExecActivity, process.tasklog* >.

- Registrar tiempo de ejecución de caso de prueba: Se recibe la notificación de un evento del tipo <*closeCaseExecActivity, process.tasklog* > en el CP que ofrece servicios a Hammurabi. Este evento activa la regla ECA 8. Esta regla registra el tiempo en la ejecución del caso de prueba local.

8.2.2 Registrar defecto

Las figuras 8.6 y 8.7 presentan el grupo de actividades que hacen parte del proceso de registro y corrección de defectos. En estas el usuario que registra una ejecución rechazada en Hammurabi debe asignar un responsable para la corrección. Este responsable puede ser modificado mientras el defecto siga abierto.

a. Registrar Responsable de Corrección

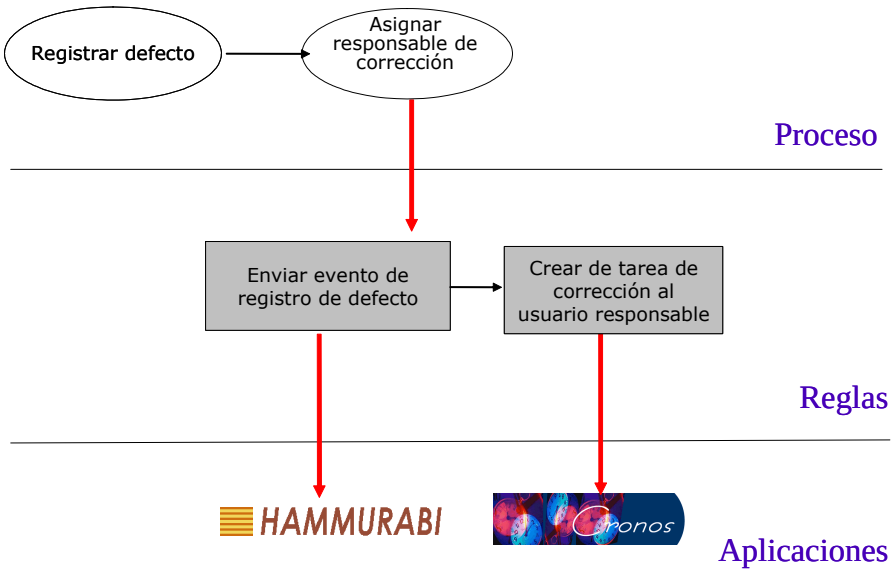

Figura 8.6 proceso de registro y asignación de defectos

- Enviar evento de registro de defecto: El usuario registra el defecto y asigna el responsable de la corrección en Hammurabi. Eleggua observa este servicio y genera un evento con la información del defecto y el responsable. La regla ECA 2 es activada por este evento y envía un evento del tipo <*testCaseRejected, processTesting*>.

- Crear tarea de corrección al usuario responsable: Se recibe la notificación de un evento del tipo < *testCaseRejected, processTesting* > en el CP que ofrece servicios a Cronos. Este evento activa la regla ECA 5 que crea una nueva tarea para el responsable del defecto con la información del defecto. Esta información es pedida por Web Services.

b. Modificar Responsable de Corrección

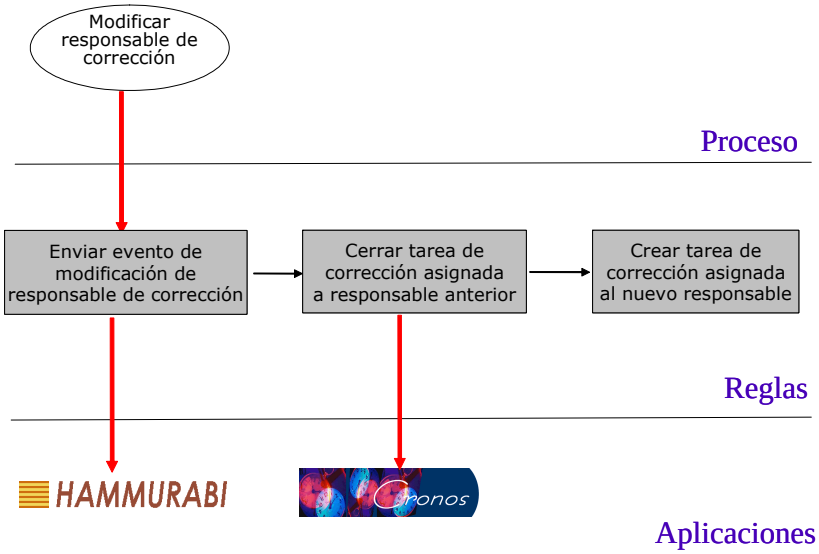

Figura 8.7 proceso de modificación de responsable de corrección

- Enviar evento de modificación de responsable de corrección: El usuario modifica el responsable de la corrección de un defecto en Hammurabi. Eleggua observa este servicio y genera un evento con el identificador del defecto y el responsable. La regla ECA 3 es activada por este evento y envía un evento del tipo <*changeDefectResponsible*, *processTesting*>.

- Cerrar tarea de corrección asignada a responsable anterior y crearla para el nuevo responsable: Se recibe la notificación de un evento del tipo <*changeDefectResponsible*, *processTesting* > en el CP que ofrece servicios a Cronos. Este evento activa la regla ECA 6 que cierra la tarea de corrección asignada al responsable anterior y crea una nueva tarea para el nuevo responsable.

8.2.3 Registrar Tiempos de Actividades Planeadas

La figura 8.8 presenta las actividades necesarias para el seguimiento y administración de todos los procesos incluyendo el de pruebas. El usuario debe ingresar a

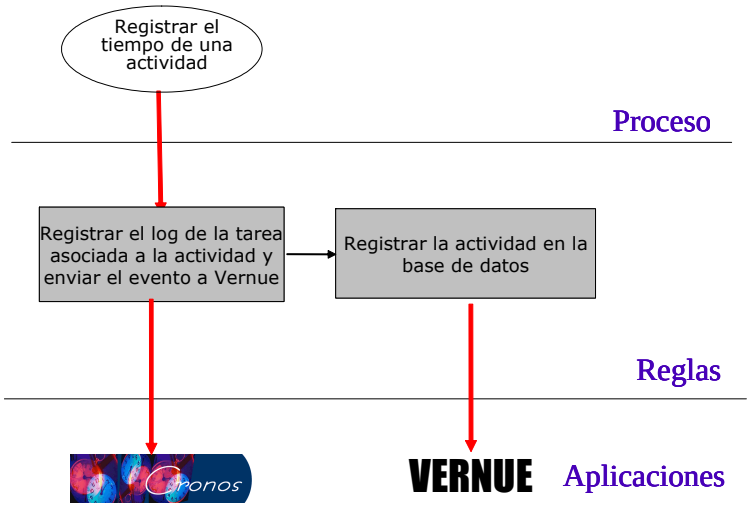

Figura 8.8 proceso de registro de tiempos de actividades planeadas

- Registrar el log de tarea asociada a la actividad: El servicio de registro de tiempo es observado por Eleggua. Esta genera un evento que es procesado por la regla ECA 10 o ECA 11 dependiendo del tipo de tarea en que se registre el tiempo. Las reglas completan información dependiendo del tipo de tarea y envían un evento de tipo *<logRequirement, proces.tasklog>* y *<logNotPlanned, proces.tasklog>* respectivamente.

- Registrar la actividad en Vernue: El CP que ofrece servicios a Vernue recibe notificaciones de los eventos con tipos *<logRequirement, proces.tasklog>* y *<logNotPlanned, proces.tasklog>*. Estos eventos activan las reglas ECA 13 y 14 respectivamente. Estas reglas ingresan la información del evento a la base de datos de Vernue directamente.

8.3 Piloto

La figura 8.9 muestra el proceso seguido en la ejecución del piloto para probar la infraestructura. La estructura de las siguientes secciones sigue este proceso.

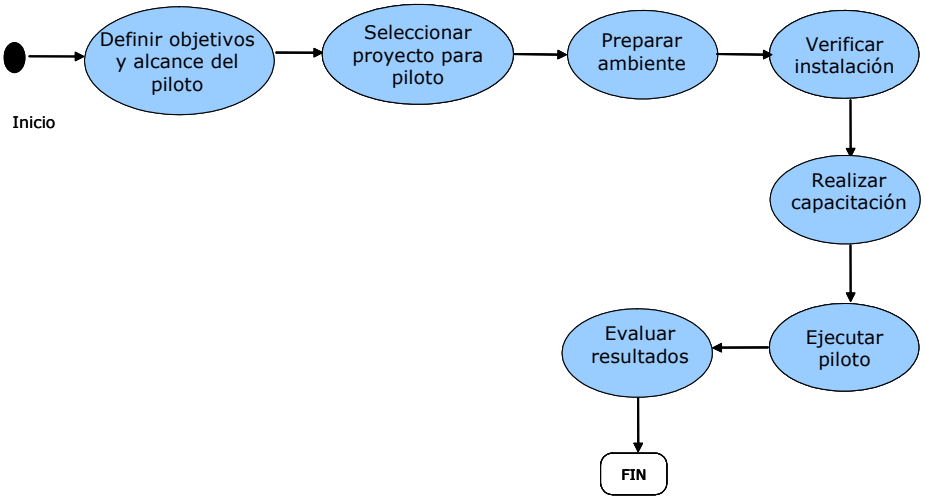

Figura 8.9 proceso del piloto

8.3.1 Objetivos Específicos Piloto

- Evaluar la funcionalidad de la infraestructura de integración. Para esto se debe evaluar que tipo de errores se presentaron, la disponibilidad de la infraestructura, y la suficiencia de la información proporcionada por la infraestructura.

- Validar que efectivamente se reduce la necesidad de contar con un coordinador del proceso. Para este objetivo se pretende medir si fue necesario que los usuarios se comunicaran constantemente, si los usuarios buscaron apoyo constante en el líder para resolver dudas. Adicionalmente se evalúa la calidad en la ejecución del proceso.

- Validar que realmente las herramientas son útiles para los usuarios y no generan una carga adicional al proceso. Para este objetivo se debe considerar central que tanta duplicación hubo en las tareas realizadas. Es necesario establecer si las herramientas agilizan el tiempo de realización del proceso, proporcionan la información suficiente y si fue necesario generar nuevas tareas adicionales debido al uso de las herramientas.

- Evaluar la correctitud y pertinencia del proceso de pruebas globalizado. Este objetivo es relacionado a la definición del proceso y de las reglas de integración. Se pretende evaluar si hubo tareas definidas en el proceso que no se realizaron, si se realizaron tareas adicionales a las definidas en el proceso y cuál fue la productividad de los usuarios.

- Validar que la infraestructura si es apropiada para la integración de procesos GSD. Evaluar el proceso general del desarrollo, si el tiempo de desarrollo y pruebas es bajo y las modificaciones necesarias a aplicaciones mínimas. Evaluar los beneficios en general de la infraestructura para procesos GSD. El cumplimiento de este objetivo depende de todos los anteriores.

8.3.2 Ambiente

Se seleccionó un proyecto de desarrollo en la casa de software para la ejecución del piloto. Este proyecto consiste de un líder de planeación, un conjunto de desarrolladores, y un grupo de usuarios encargados de las pruebas. Este proyecto es pertinente en el contexto pues el grupo de desarrolladores y de usuarios se encuentra distribuido.

La figura 8.10 presenta el ambiente de distribución en que se ejecutó el piloto. Se decidió usar dos locaciones remotas para las aplicaciones para probar la distribución de la infraestructura. En la locación de Heinsohn se instalaron las herramientas Hammurabi y Vernue, y adicionalmente un componente productor y consumidor del DEM y el CP.

En la locación de la Universidad de los Andes se instaló cronos y todos los componentes de Eleggua: los tres sub- componentes del DEM, el CP y el monitor.

El usuario tiene acceso por Web a las tres herramientas (flechas rojas) y la comunicación entre las aplicaciones es a través de eventos (flechas verdes) y es mediada por Eleggua.

Fue necesario solicitar acceso a puertos diferentes al 80 en los fierwalls entre los servidores para poder establecer la conexión entre los componentes.

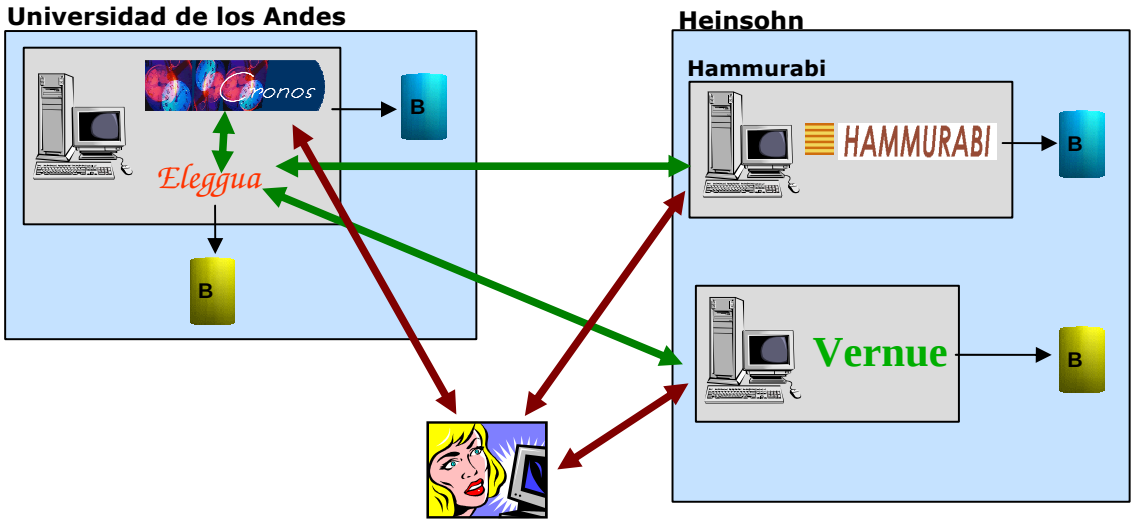

Figura 8.10 ambiente de ejecución del piloto

8.3.3 Pruebas

Se desarrolló una plataforma simple de pruebas de ejecución de acciones. La plataforma se basa en un decorador de pruebas de junit que lee de archivos los datos para generar eventos y decora las pruebas con eventos leídos. Las clases de prueba para cada caso ejecutan la acción a través del componente procesador de eventos del CP y las aserciones revisan el estado de la aplicación. Para probar la generación de eventos se usa la plataforma de pruebas para acciones que producen eventos y se hace aserción sobre la recepción del evento en Eleggua.

Eleggua también cuenta con un esquema de pruebas de las reglas de observación sobre las aplicaciones basado en junit. Las observaciones llaman servicios de una clase que genera logs al

iniciar y terminar la intercepción. Una clase de pruebas por cada observación llama el servicio a interceptar y hace una aserción sobre la creación del Log. Un último componente lee los logs generados a archivos y responde búsquedas sobre los logs de observaciones.

8.3.4 Documentación

Se entrego a los usuarios un manual de instalación y administración, y un manual de usuario.

El manual de instalación y administración detalla todas las configuraciones y pasos necesarios para la instalación de Eleggua, este documento adicionalmente específica los pasos para creación de módulos de integración. Un módulo de integración es un conjunto de reglas de observación y ECA para una aplicación

El manual de usuario incluye la descripción de la herramienta Cronos. Este describe la interacción con la aplicación que incluye el control de acceso, los bloques de información, la administración del sistema y la descripción de los varios roles: responsable de proyecto, líder de planeación e ingeniero de desarrollo.

El proyecto incluye además una infraestructura para la documentación técnica. Esta infraestructura está basada en XML para definir los contenidos de la documentación y XSL para generar la presentación en forma de una página Web.

La figura 8.11 presenta la interfaz de la documentación técnica del proyecto

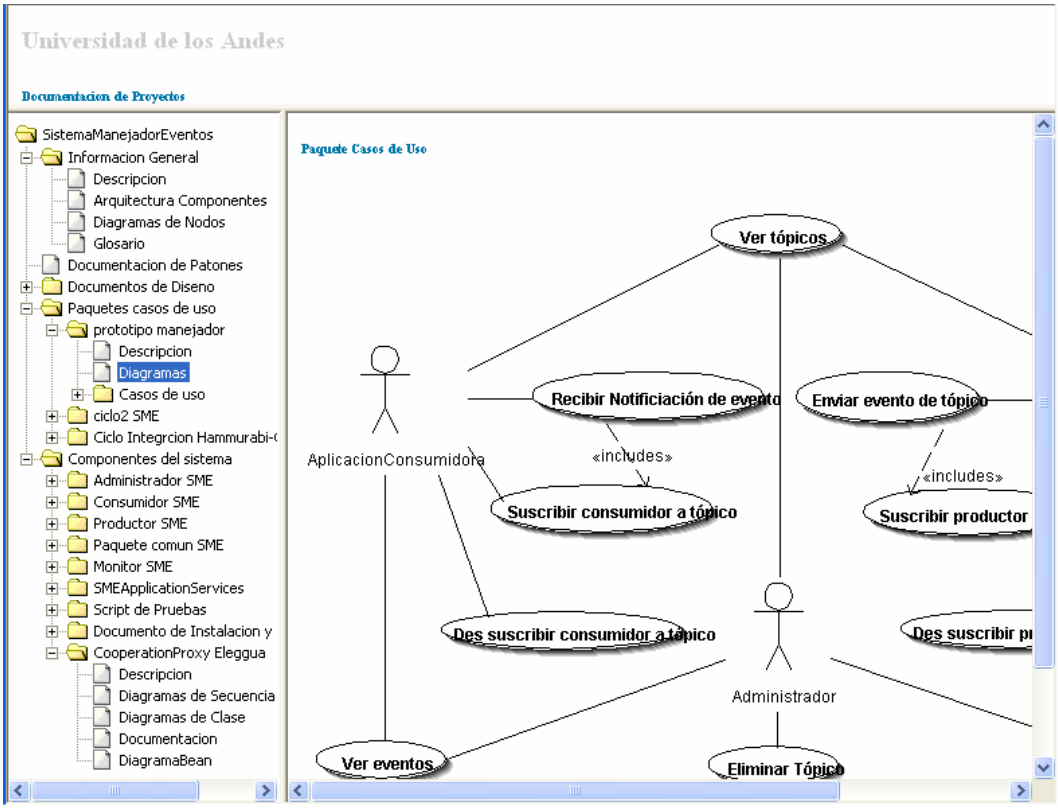

Figura 8.11 interfaz de la documentación técnica

8.3.5 Capacitación

El proyecto necesita capacitación para el uso de las nuevas herramientas y de la integración. Hammurabi y Vernue han sido ya usadas por los miembros del proyecto anteriormente. Cronos es una herramienta nueva que no ha sido usada anteriormente en producción. El uso de esta herramienta se divide en roles (tipos de usuarios), es necesario hacer capacitación para los roles de líder de planeación e ingeniero de desarrollo.

La capacitación para los usuarios del proyecto se ha dividido en dos fases:

Fase 1: Entrenamiento inicial al Líder de Planeación

Capacitación al usuario que asumirá el rol de Líder de Planeación para que conozca el proceso de planeación utilizado en Cronos y tenga la capacidad de realizar la planeación de actividades en el proyecto.

Fase 2: Presentación del piloto y entrenamiento general a usuarios

Presentación del piloto y sus objetivos.

Capacitación a los usuarios que asumirán el rol de Ingeniero de Desarrollo, para que conozcan la forma como deben realizar el registro de actividades utilizando Cronos.

8.4 Resultados

Los resultados del piloto son agrupados por los objetivos planteados previamente.

- **Evaluar la funcionalidad de la infraestructura de integración.**

El desarrollo y pruebas del sistema identificaron algunos errores menores en la infraestructura. Se considera esto positivo pues fue un proceso en el que se hicieron pruebas completas del uso de los componentes principales de la infraestructura.

Los errores principales actualmente son de degradación de los recursos después varios días de uso de la infraestructura (Ver anexo 2). El manejo de transacciones manuales y el acceso a la base de datos han presentado algunos errores que están siendo evaluados en este momento. En la mayoría de los casos la solución a los errores es reiniciar el contenedor. A pesar de esto la disponibilidad de la infraestructura ha sido alta y debido a que no es una aplicación de tarea crítica no se ha perdido información en los momentos en que no ha estado disponible.

- **Validar que efectivamente se reduce la necesidad de contar con un coordinador del proceso.**

El soporte dado a los desarrolladores por las herramientas y la infraestructura es apropiado; sin embargo el piloto no fue apropiado para medir la reducción de la necesidad del coordinador pues todos los usuarios estaban ubicados en la misma oficina. Próximamente un usuario trabajará remotamente y a partir de esta experiencia pretendemos recolectar más datos para medir este objetivo con mayor precisión.

- **Validar que realmente las herramientas son útiles para los usuarios y no generan una carga adicional al proceso.**

La carga se redujo sobre los desarrolladores al ingresar tiempos; sin embargo, la ocurrencia de errores en la primera semana llevo a que una vez ingresados los tiempos en Cronos los usuarios tuvieran que revisar en Vernue que el log si se creo correctamente.

La usabilidad de las herramientas fue un factor relevante en los comentarios de los usuarios. La mayoría de las solicitudes de cambio buscaban mejorar este factor. El usuario planeador tuvo bastantes quejas sobre el tiempo necesario para hacer la planeación de las tareas en Cronos. Esto nos ha recalcado la necesidad de integrar al proceso una herramienta que facilite la planeación

- **Evaluar la correctitud y pertinencia del proceso de pruebas globalizado.**

Algunas de las reglas de integración fueron modificadas durante el piloto debido a ajustes menores en el proceso. Estos ajustes buscan que el proceso pueda institucionalizarse con el apoyo de los usuarios. En particular las convenciones sobre nombramiento y conversión de identificaciones de conceptos como el de requerimientos fueron revisados durante el piloto.

Cabe notar que el proceso de levantamiento de las reglas de negocio no fue sencillo. Los directores de proyecto con que nos reunimos inicialmente ofrecieron una visión abstracta del proceso; sin embargo, la mezcla entre conceptos propios de los procesos de negocios y los conceptos de las aplicaciones causaron problemas. En particular levantar información sobre los detalles del comportamiento de las herramientas legado en la empresa fue un proceso complejo.

Un importante resultado del piloto es que consideramos que es necesario que las aplicaciones a integrar estén bien documentadas para facilitar el proceso de desarrollo de las reglas.

- **Validar que la infraestructura si es apropiada para la integración de procesos GSD.**

La integración de aplicaciones legado fue un proceso interesante. Un desarrollador de la empresa construyo un módulo intermedio que permitiera que la aplicación legado compartiera datos con un CP desarrollado particularmente para la aplicación.

El esfuerzo necesario para el desarrollo de las reglas de integración ha bajado, aún mas, se considera que la curva de aprendizaje para el desarrollador es baja y que rápidamente pueden integrarse aplicaciones sin tener mucho conocimiento de las aplicaciones a integrar.

El tiempo de pruebas fue muy largo y el esfuerzo fue alto. La infraestructura de pruebas tuvo que ser implementada para esta necesidad pues inicialmente no se identificó esto como un problema crítico. Una vez la infraestructura de pruebas se creó el problema se redujo y se considera que es crítico en integraciones futuras definir primero las pruebas y posteriormente las reglas de integración.

Finalmente el piloto ha mostrado que la infraestructura propuesta si ayuda a dar soporte a los procesos GSD. Esto es por varias razones: primero, porque ofrece un solo punto de entrada a las aplicaciones. Segundo, la infraestructura garantiza la propagación automática de la información basado en las reglas de negocio definidas. Tercero, a pesar de que el proceso de levantamiento y definición de las reglas es tedioso una vez el proceso ha sido integrado es menos propenso a errores. Por último, las herramientas integradas proveen una visión integral del estado de su trabajo y de sus tareas pendientes y al líder de proyecto ofrece visión completa del trabajo, lo que evita el exceso de comunicación cara a cara.

9 Conclusiones y Trabajo Futuro

9.1 Resumen

En este trabajo hemos presentado Eleggua, una infraestructura para integración y cooperación de aplicaciones. Primero se presentaron algunas arquitecturas y tecnologías similares a la nuestra y que sirvieron como base para nuestro trabajo. Posteriormente se presentó el modelo de Eleggua que permite integrar aplicaciones legado a través de un sistema de notificación por eventos y de representantes de aplicación que intermedian la relación entre Eleggua y las aplicaciones. A continuación se presentó el modelo de implementación de Eleggua sus los componentes principales: DEM y CPs. Finalmente se presentó el proyecto piloto en el que se probó y validó la infraestructura.

El enfoque principal en el diseño fue el soporte a la cooperación y la integración de aplicaciones, y mantener la consistencia de la información. Las restricciones principales fueron impuestas por los sistemas legados; nuestro objetivo era ofrecer una integración laxa y bajo acoplamiento de aplicaciones en un contexto distribuido. Nuestro enfoque se centra en:

- La interacción entre las aplicaciones y la infraestructura basada en eventos,
- La comunicación basada en eventos y el uso de Web Services como mecanismo de solicitud/respuesta;
- Reglas ECA y de observación de aplicaciones para describir la cooperación entre las aplicaciones.

Eleggua fue validado en un proyecto piloto en un contexto real GSD para dar soporte a un proceso distribuido de administración de pruebas y defectos de software. Consideramos que la validación demuestra que la infraestructura si da soporte a los procesos GSD pues junto con las herramientas y las reglas de negocio implementadas ayuda a los desarrolladores a controlar las actividades, sincronizar el trabajo y comunicar los resultados.

9.2 Aportes

El aporte principal de este proyecto está en reunir una serie de estrategias de integración, arquitecturas y tecnologías para construir una infraestructura de integración que satisface unos requerimientos particulares identificados. El proyecto retoma el trabajo de varios autores en el área de sistemas de notificación basados en eventos y lo extiende a través del uso de tecnologías que faciliten la integración del sistema con otras aplicaciones.

Es relevante notar que no conocemos de otros proyectos que usen las estrategias planteadas para la interacción con las aplicaciones. En particular cabe notar que la observación no intrusiva de aplicaciones para generación de eventos y el uso de reglas ECA para describir la integración en un sistema EAI es novedoso.

Otro aporte relevante es la experiencia en la definición y transformación de un proceso de negocio hacia una infraestructura específica. Esto, junto con el proyecto piloto, valida la pertinencia de los problemas y necesidades que satisface la infraestructura.

Cabe notar que un aporte notable del proyecto es que efectivamente resuelve un problema real que enfrentan las empresas actualmente. Consideramos que uno de resultados del piloto es un aporte relevante del proyecto. Este es que el proceso de pruebas de integración definido para reducir el esfuerzo necesario en la definición de reglas. Actualmente estamos trabajando en extender este esquema para hacerlo genérico a otras infraestructuras.

9.3 Trabajo Futuro

Algunos aspectos de la infraestructura aún están por desarrollar y mejorar; estos son planteados como trabajo futuro.

Definir un lenguaje de alto nivel posiblemente en términos de un profile de UML que permita describir los procesos de negocio, las herramientas y las reglas ECA y de observación. El lenguaje debe incluir la definición de transformaciones de un modelo de reglas de negocio a un modelo específico implementado por Eleggua. Este lenguaje y las transformaciones facilitarían a los desarrolladores y usuarios de negocio la definición y evolución de las reglas de negocio que definen un proceso GSD

Diseñar una plataforma de simulación de reglas de negocio que permita validar un proceso antes de implementar la integración para una plataforma específica como Eleggua. Esto podría hacerse a través de UML ejecutable, esta tecnología permite definir y simular modelos. Esta plataforma debería también proveer servicios para transformar un proceso definido en el lenguaje de alto nivel mencionado previamente.

Estos dos factores permitirían modelar y simular un proceso de negocio sin necesidad de transformarlo y detallarlo para su ejecución en una plataforma específica como Eleggua. Adicionalmente, el proceso de pruebas de un proceso sería más eficiente pues el proceso sería validado en una fase temprana de la integración.

Otro factor crítico que es trabajo futuro del proyecto es la construcción de una consola de administración que aproveche al máximo los servicios de administración ya instrumentados. Este componente debe permitir, adicionalmente a la administración de la infraestructura, la administración, pruebas y recuperación de fallos del proceso.

Finalmente algunos aspectos tecnológicos y de extensión de los servicios actuales de Eleggua son considerados trabajo futuro. Primero, es necesario implementar un mecanismo efectivo de comunicación para conectar varios componentes despachadores del DEM que cumpla con los requerimientos impuestos por una red de gran escala como Internet. Adicionalmente es deseable diseñar e implementar un componente observador de patrones para el Proxy de Cooperación. Este componente permitirá definir reglas ECA que sean activadas por la ocurrencia de un patrón de eventos particular. El componente tiene como responsabilidad administrar, detectar y notificar la ocurrencia de patrones de eventos y la composición de eventos. Finalmente, se quiere sobreponer las restricciones impuestas por la tecnología AspectJ que hace que el deployment de una nueva regla de observación requiera acceso directo al código fuente de una aplicación. Adicionalmente, es deseable encontrar un mecanismo de intercepción de aspectos que sea independiente de plataforma.

10 Referencias

[1] Joonsoo Bae, Hyerim Bae, Suk-Ho Kang, and Yeongho Kim. Automatic control of workflow processes using eca rules. IEEE Transactions on knowledge and data engineering 16 (8), 2004.

[2] Antonio Carzaniga, David Rosenblum, and Alexander Wolf. Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems, 19(3):332– 383, 2001.

[3] Módulo V. Descripción del Proyecto GSD Uniandes

[4] Gianpaolo Cugola, Elisabetta di Nitto, and Alfonso Fuggeta. The jedi event-based infrastructure and its application to the development of the opss wfms. IEEE Transactions on Software Engineering, 27(9), 2001, 2001.

[5] Andreas Geppert, Mikael Berndtsson, Daniel Lieuwen, and Claudia Roncancio. Performance evaluation of object-oriented active dbms using the beast benchmark. TAPOS Intl. journal, ed K. Lieberherr and R.Zicari, Vol 4(8), pub. John Wiley and Sons, Inc, 1998, 1998.

[6] Andreas Geppert and Dimitrios Tomboros. Event-based distributed workflow execution with eve. In Proc. IFIP Int'l Conf. on Distributed Systems Platforms and Open Distributed Processing (Mideware'98), Lake District, England, 1998.

[7] Mockus Audris & Weiss David. "Globalization by chunking: A quantitative approach". IEEE Software Marzo 2001.

[8] David Hilbert and David Redmiles. An approach to large-scale collection of application usage data over the internet. In Proceedings of the 20th International Conference on Software Engineering, 1998.

[9] Gerti Kappel, Stefan Rausch-Schott, and Werner Retschitzegger. A framework for workflow management systems based on objects, rules and roles. In Bulletin of the Technical Committee on Data Engineering, March 1995,18(1), pp. 11-18., 1995.

[10] Balachander Krishnamurthy and David Rosenblum. Yeast: A general purpose event-action system. IEEE Transactions on Software Engineering, vol. 21, pp. 845-857, 1995.

[11] Mohagheghi P. "Global Software Development: Issues, Solutions, Challenges". University of Science and Technology, Trondheim, Norway. Trial Lecture 2004.

[12] David Rosenblum and Alexander Wolf. A design framework for internet-scale event observation and notification. Proceedings of the6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering, 1997.

[13] Roberto Silva, Cleidson de Souza, and David Redmiles. The design of a configurable, extensible and dynamic notification service. In In Proceedings of the 2nd International Workshop on Distributed Event-Based Systems (DEBS'03), June 2003.

[14] Jonathan Cook. "Internet -based Software Engineering Enables and Requires Event-Based Management Tools". 3rd Workshop on Software Engineering over the Internet. 2002.

[15] JBOSSMQ team. http://www.jboss.org/wiki/Wiki.jsp?page=JBossMQ. last visited on 2005-01-15.

[16] Cleideson De Souza, Santhosi Basaveswara and David Redmiles. "Supporting Global Software Development with Event Notification Servers". Workshop on Global Software Development. 2002

[17] Herbsleb James & Mockus Audris. “Formulation and Preliminary Test of an empirical Theory of Coordination in Software Engineering”.ESEC/FSE'03 Septiembre 2003.

[18] Dimitrios Tomboros and Andreas Geppert. Building extensible workflow systems using an event-based infrastructure. In Proc. 12th Int’l Conf. On Advanced Information Systems Engineering, Stockholm, Sweden, 2000.

[19]. A. Limongiello, R. Melen, M. Roccuzzo, V. Trecordi, J. Wojtowicz, “An Experimental Open Architecture to Support Multimedia Services Based on CORBA, Java and WWW Technologies”, IS&N '97, Cernobbio (Como), Italy, 27-29 May 1997.

[20] Jennifer Widom and Stefano Ceri. Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, 1996.

[21] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, AZ, October 2001.

[22] Systinet Corporation. Web Services: A practical Introduction to SOAP Web Services. http://www.systinen.com. Consultado el 10/04/2005

[23] Bill Buzbee and Stepthen DuBravac. Building in Application Manageability, a developer perspective.

[24] Sun Microsystems. Java Management Extensions White Paper. Dynamic Management for the Service Age.

[25] Justin Murray. Design Patterns for JMX and Application Manageability. October 2004

[26] Peter Pietzuch. Hermes: A Scalable Event-Based Middleware. A dissertation submitted for the degree of Doctor of Philosophy. Queens' College, University of Cambridge. February 2004.

11 Anexos

11.1 Especificación de Eleggua (Elementos del Modelo)

1.1. Tópico

Definición

Es un concepto de dominio común para un grupo de aplicaciones. Estos conceptos están basados en acuerdos para nombrar e identificar conceptos comunes de dominio que se traslapan en varias aplicaciones.

Un tópico <topic> esta representado por una cadena de caracteres alfanuméricos, empezando por una letra.

Ejemplo: “assignCases” (Tópico que representa el proceso de asignación de casos de pruebas para que sean ejecutados por un usuario).

1.2. Contexto

Definición

Un contexto <context> representa un dominio lógico de eventos. Un contexto se expresa como una cadena de caracteres alfanuméricos, empezando por una letra.

Ejemplo: “process.testing”. Bajo este contexto se agrupan lógicamente todos los eventos que se generan alrededor de la ejecución del proceso de pruebas.

1.3. Tipo de Evento

Definición

Un tipo de evento <eventType> se define a través de un contexto y de un tópico. A través de los tipos de eventos se caracterizan los eventos lógicos <logicalEvent> y las reglas ECA. <ECARule>.

Las suscripciones para recibir notificación sobre la ocurrencia de eventos de interés se realiza a través de <eventType>Estructura

Nombre	Tipo	Descripción
topic	<topic>	Nombre que identifica al tópico
context	<context>	Nombre que identifica al contexto.

1.4. Evento Lógico

Definición

Un evento lógico <logicalEvent> representa la ocurrencia de un evento de interés en una aplicación un evento lógico está caracterizado por un tipo de eventos <eventType>, un productor

<application> y un conjunto de parámetros <parameter>. Los eventos lógicos son los parámetros de entrada durante ejecución de las acciones <actions> de una regla ECA < ECARule>. Adicionalmente son el resultado de la ejecución de una observación <observation>.

Estructura

Nombre	Tipo	Descripción
eventType	<eventType>	El tipo de eventos de este evento lógico
producer	<application>	La aplicación que creo el evento
parameters*	<parameter>	Lista de parámetros del evento

1.5. Parámetro de evento

Definición

Un parámetro <parameter> caracteriza un parámetro de un evento lógico <logicalEvent> con un nombre y tipo

Estructura

Nombre	Tipo	Descripción
name	String	El nombre del parámetro
value	Object	El valor del parámetro

1.6. Registro de Aplicación

Definición

Un registro de aplicación identifica de manera única una aplicación que consume <consumer> o produce <producer> eventos. Esta identificación es usada para suscripciones <subscription> y para identificación de eventos producidos (en caso de ser <producer> o para notificación e identificación de eventos para consumidores (en caso de ser <consumer>.

Estructura

Nombre	Tipo	Descripción
applicationId	String	El identificador único de la aplicación
eventHandler	String	Opcional. El nombre completo de una clase que implemente IEventHandler para aplicaciones de tipo <consumer> para push de eventos

1.7. Suscripción

Definición

Una suscripción específica el interés de una aplicación en un tipo de evento <eventType> puede ser de dos tipos productor <producer> o consumidor <consumer>. Es necesario que haya una suscripción declarada para un tipo de eventos <eventType> para que una aplicación <application> pueda producir eventos <logicalEvent> del tipo dado.

Estructura

Nombre	Tipo	Descripción
application	<application>	La aplicación que se suscribe al evento
eventType	<eventType>	El tipo de eventos lógicos que serán producidos o consumidos
type	String	El tipo de suscripción para producir o consumir eventos

1.8. Regla de Observación

Definición

<observation> describe una observación sobre la ejecución de una aplicación <application>. Durante ejecución se intercepta la ejecución de la aplicación <application> y se genera un evento lógico <logicalEvent> para un tipo de eventos caracterizado <eventType>.

Es caracterizada por un método a interceptar y un conjunto de parámetros de observación.

Estructura

Nombre	Tipo	Descripción
eventType	<eventType>	Tipo de eventos que produce esta observación
application	<application>	La aplicación que es observada (la que quedará como productora del evento)
method	<interceptedMethod>	Método a observar e interceptar en la aplicación
parameters*	<observationParameter>	La lista de parámetros del evento generado

1.9. Método Interceptado

Definición

<interceptedMethod> Caracteriza un método a interceptar. Esto incluye la signatura del método para interceptar y un conjunto de parámetros del método. Un método interceptado <interceptedMethod> hace parte de una observación <observation>.

Estructura

Nombre	Tipo	Descripción
className	String	Clase (Session Bean) a observar
signature	String	Encabezado del método a observar
methodParameter*	<methodParameter>	Parámetros del método

1.10. Parámetro de método

Definición

Un parámetro de método <methodParameter> es usado como fuente de un parámetro de observación y pertenece a un método < interceptedMethod>. Son caracterizados por el nombre el tipo del parámetro y la posición dentro del encabezado del método al que pertenece.

Estructura

Nombre	Tipo	Descripción
name	String	El nombre del parámetro
type	String	El tipo del parámetro
position	Integer	La posición en el encabezado del método del parámetro

1.11. Parámetro de Observación

Definición

Un Parámetro de Observación <observationParameter> caracteriza un parámetro de una regla de observación <observation>. Los eventos lógicos <logicalEvent> producidos por la regla tienen un parámetro <parameter> que corresponde a este parámetro de observación <observationParameter>.

Estructura

Nombre	Tipo	Descripción
name	String	El nombre con el que quedará el parámetro en el evento lógico generado por la observación
methodParameter	<methodParameter>	El parámetro del método con el que se establecerá el valor del parámetro del evento lógico generado
type	String	El tipo del parámetro
getterMethod	String	Opcional. Nombre de un método a invocar sobre el <methodParameter>. En caso de estar presente el valor del parámetro en el evento lógico generado corresponde el retorno de este llamado

1.12. Regla ECA (Evento-Condición- Acción)

Definición

Una regla ECA <ECARule> describe una regla que recibe eventos de un tipo <eventType> y si pasa la condición <condition> ejecuta la acción <action> dada.

Estructura

Nombre	Tipo	Descripción
application	<application>	La regla que afecta este evento
eventType	<eventType>	El tipo de eventos que activan la regla
condition	<condition>	La condición que evalúa el evento lógico <logicalEvent> que activa la regla
action	<action>	La acción a ejecutar una vez activada la regla
source	String	Observation/Notification denota que la regla procesa eventos generados por una observación <observation> o por una notificación dada una suscripción <subscription> de tipo consumidor.

1.13. Condición

Definición

La condición <condition> evalúa si un evento lógico <logicalEvent> pasa una regla ECA <ECARule> a la que pertenece esta condición. Evalúa la existencia de parámetros en el evento lógico. La condición pasa si todos los parámetros están presentes.

Estructura

Nombre	Tipo	Descripción
parameters*	<parameter>	Lista de parámetros necesarios en el evento lógico <logicalEvent> para que la condición se pase.

1.14. Acción

Definición

<action> es La acción a ejecutar si un evento lógico <logicalEvent> pasa la condición de una regla ECA <ECARule>. Las acciónes son caracterizadas por la clase que implementa IRuleExecution que recibe como parámetro de su método execute() un evento lógico <logicalEvent> que se encargan de procesarlo.

Estructura

Nombre	Tipo	Descripción
actionClass	String	El nombre completo de la clase que implementa IRuleExecution que procesa el evento lógico <logicalEvent>

11.2 Errores Piloto Eleggua

Los errores más comunes que se han presentado en la infraestructura a lo largo del piloto son los siguientes:

Error	***Disponibilidad infraestructura***
Después de cierto tiempo de estar funcionando, en la infraestructura empiezan a salir errores con las transacciones.	Cuando esta situación se presenta (se ha presentado en pocas ocasiones) se empiezan a generar inconsistencias en las aplicaciones y es necesario reiniciar el servidor. Una vez se reinicia el servidor nuevamente la infraestructura funciona adecuadamente.
Se presenta constantemente un error de java.lang.nullPointerException sin causa aparente en el método finderByPrimaryKey en la clase ObservationBean.	Por lo general la infraestructura sigue funcionando adecuadamente a pesar del error y no es necesario reiniciar el servidor.
Se presenta un error en la clase SMEDBLocator indicando que no es posible establecer una conexión con la Base de Datos. Sin embargo la base de datos se encuentra disponible. Adicionalmente al consultar por la consola JMX de Jboss también es posible localizar el SMEDs.	Es necesario reiniciar el servidor de aplicaciones y nuevamente la infraestructura sigue funcionando bien.
SQL Exception a causa del tamaño de los mensajes de error, en ocasiones son demasiado extensos y no se pueden manejar. Este problema se generaba por el campo BLOB que se manejaba en la tabla SME_EVENT. Este error ya se solucionó pero es necesario realizar las actualizaciones pertinentes en el escenario del piloto.	Por lo general la infraestructura sigue funcionando adecuadamente a pesar del error y no es necesario reiniciar el servidor.

Otros errores que se han presentado en el envío y recepción de eventos son producto de errores en la implementación de las reglas de integración de las aplicaciones participantes.