# Estrategia dirigida por modelo para el gobierno SOA

# A model-driven governance analysis tool for SOA-based systems

**Diana Angélica Cruz Ortega. M.Sc. & Darío Ernesto Correal Torres. Ph.D**
Departamento de Sistemas y Computación, Universidad de Los Andes, Colombia
*da.cruz135@uniandes.edu.co, dcorreal@uniandes.edu.co*

Recibido para revisión 01 de octubre de 2010, aceptado 28 de junio de 2011, versión final 30 de junio de 2011

***Resumen*— Actualmente es común ver como muchas organizaciones utilizan arquitecturas orientadas a servicios (SOA) para dar soporte a la automatización de sus procesos de negocio. Los servicios ofrecen la flexibilidad requerida por las organizaciones modernas, en las que el cambio y la continua evolución son características primordiales. Esta evolución requiere la toma de decisiones arquitecturales las cuales deben garantizar y controlar la evolución coherente de la SOA. Para esto es necesario definir e implementar un modelo de gobierno SOA. En esta propuesta, presentamos una estrategia basada en la Ingeniería Dirigida por Modelos (MDE), para apoyar la gobernabilidad de la arquitectura, permitiendo definir políticas de gobierno SOA y evaluar si estas políticas se cumplen o no en la arquitectura de solución.**

***Palabras Clave*— SOA, Gobierno, MDE.**

***Abstract*— It is now commonly seen as many organizations use service-oriented architecture (SOA) to support the automation of business processes. The services offer the flexibility required by modern organizations in which continuous change and evolution are key characteristics. This development requires architectural decisions which should secure and control the coherent evolution of the SOA. This requires defining and implementing an SOA governance model. In this proposal, we present a strategy based on Model Driven Engineering (MDE) in support of architecture governance, allowing the definition of SOA governance policies and assess whether these policies are met or not in the solution architecture.**

***Key words*— SOA, Governance, MDE.**

## I. INTRODUCCIÓN

Las arquitecturas orientadas a servicios (SOA)[9], se han convertido en el nuevo estándar de facto para el diseño y creación de servicios de negocio reutilizables que se pueden compartir en toda la empresa. Las SOA ofrecen una alta capacidad de reutilización de activos de software empresariales de una manera ágil y robusta, acompañada de la flexibilidad que les permite hacer frente al cambio y la interoperabilidad que facilita la integración de estos activos heterogéneos.Sin embargo, la complejidad de los procesos de negocio, su entorno cambiante y la rápida adopción y creación de servicios da como resultado una arquitectura de solución SOA altamente compleja, que sin un adecuado medio de control puede llegar rápidamente a ser inmanejable para los administradores de TI y los arquitectos SOA[2]. Dado que en las arquitecturas orientadas a servicios, la lógica de negocio que exponen sus servicios es compartida por distintos procesos de negocio, un cambio en un servicio tiene un impacto potencial en varios segmentos de negocio.

El problema que motiva nuestra investigación, se centra en el impacto de implementar un cambio en la arquitectura, ya que si este impacto no es evaluado antes de su implementación se pueden generar situaciones no deseadas o inesperadas. Este tipo de situación son por ejemplo: la implementación redundante de servicios lo que genera un crecimiento desordenado de la arquitectura y disminuye la posibilidad de reutilizar los servicios existentes [6], ó la interrupción de procesos de negocio cuando se realizan cambios en los servicios, sin considerar todos y cada uno de los procesos de negocio que hacen uso de este servicio [7]. Las consecuencias de este tipo de situaciones pueden ir desde el incumplimiento en los acuerdos de nivel de servicio, hasta el abandono de la implementación de la arquitectura por que se pierde la credibilidad en el paradigma orientado a servicios.El costo de implementar una decisión arquitectural que tiene un impacto negativo, se incrementa a medida que se avanza en el ciclo de vida de los servicios. Por esta razón se hace necesario contar con un modelo de gobierno SOA que permita administrar cada uno de los servicios, su ciclo de vida y como éstos se relacionan con sus consumidores en la ejecución de los procesos de negocio, para lograr una arquitectura sólida, confiable, segura y escalable.Pensando en apoyar a las organizaciones en hacer frente a esta problemática en la gobernabilidad de sus arquitecturas SOA, presentamos una estrategia dirigida por modelos (MDE), que apoya la definición de políticas de gobierno SOA y la evaluación de su cumplimiento en la arquitectura. Aplicar un enfoque dirigido por modelos, nos permite analizar el impacto de las decisiones arquitecturales

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

representando la arquitectura SOA y las políticas de gobierno, mediante modelos independientes de la tecnología involucrada en la implementación y ejecución de los servicios.

La organización de este artículo es la siguiente: en la siguiente sección presentamos los principales conceptos del SOA y Gobierno SOA, en la sección tres presentamos los conceptos principales de MDE que soportan esta estrategia, en la sección cuatro presentamos el metamodelo Archivol, el cual es base fundamental en el desarrollo nuestra estrategia, en la sección 5 exponemos nuestra estrategia y la ilustramos mediante un ejemplo. Finalmente, en la sección seis presentamos nuestras conclusiones y proponemos un trabajo futuro.

## II. SOA Y GOBIERNO SOA

En esta sección se presentan los principales conceptos de las arquitecturas orientadas a servicios y su gobierno, así como la situación actual de las propuestas para gobernar este tipo de arquitecturas.

### 2.1 Arquitecturas Orientadas por Servicios

En la actualidad es común ver como las organizaciones hacen uso de las arquitecturas orientadas por servicios (SOA) para implementar sus procesos de negocio. De acuerdo con [2], SOA es una combinación de: un modelo de negocio, una estrategia de TI y una propuesta arquitectural. SOA como modelo de negocio se basa en la oportunidad de ejecutar algunas funciones del negocio basado en *outsourcing*. Como estrategia de TI se basa en la idea de proveer servicios de tecnología y activos. Desde el punto de vista técnico una arquitectura orientadas a servicios es un paradigma para organizar y utilizar capacidades distribuidas que pueden estar bajo el control y gobierno de diferentes dominios [10].

Este paradigma se basa en la idea de que las entidades (personas y organizaciones) crean capacidades para dar solución a necesidades que enfrentan en la ejecución de sus procesos de negocio. Estas entidades pueden crear capacidades que solucionan las necesidades de otras y viceversa. Una *necesidad* es un requerimiento medible que se busca satisfacer, mientras que una *capacidad* es un recurso que puede ser usado para satisfacer determinada(s) necesidad(es) [10]. Por ejemplo, una organización puede tener la *necesidad* de conocer rápidamente los costos de los artículos que le suministra determinada empresa proveedora, por su parte la empresa proveedora puede ofrecerle un servicio que le brinda la *capacidad* de consultar esta información a través de una página de internet. Sin embargo, no existe necesariamente una correlación uno a uno entre las necesidades y capacidades; de manera que para satisfacer determinada necesidad puede ser necesario combinar múltiples capacidades, mientras que una capacidad puede abarcar más de una necesidad. Esto genera una gran complejidad, el gran aporte que ofrece SOA como propuesta arquitectural es que proporciona un marco bajo el cual es posible combinar las capacidades ofrecidas para hacer frente a las necesidades de un negocio promoviendo la reutilización e interoperabilidad de las capacidades [9].Tanto en la industria como en la academia se encuentran distintos enfoques para describir una SOA. En este trabajo hemos decidido usar como referencia el marco de modelamiento orientado a servicios SOMF (Service Oriented Modeling Framework)[3].SOMF propone dos tipos de modelos para describir las arquitectura SOA, un modelo estático que es el portafolio de servicios y modelos dinámicos que son los ecosistemas SOA, estos modelos se describen a continuación.

**Portafolio de Servicios**: es un catálogo que describe los servicios requeridos por la organización para ejecutar sus procesos de negocio. A través del portafolio de servicios es posible: establecer la clasificación de los servicios, determinar la información requerida para conocer y administrar el servicio y administrar el ciclo de vida de los servicios.

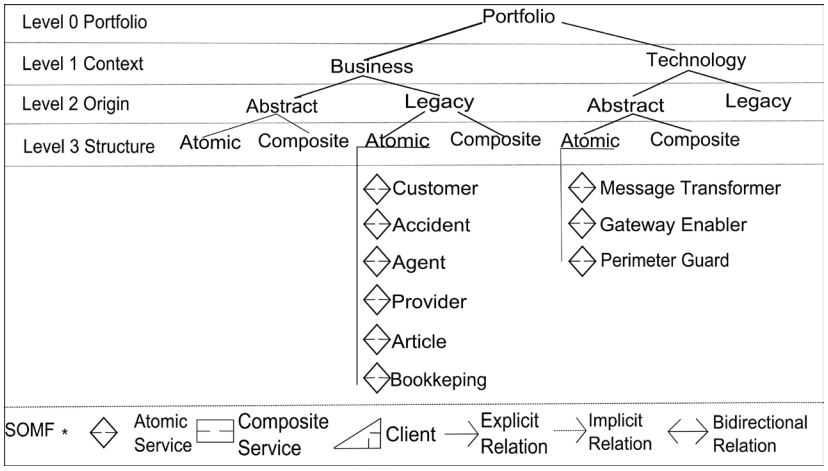


**Figura 1.** Portafolio de Servicios

Sin una identificación clara, se dificulta determinar la capacidad que tiene un servicio para responder a determinadas necesidades de la organización, por esta razón SOMF propone clasificar los servicios por contexto, origen y estructura, esta clasificación se presenta con una estructura arborescente como la presentada en la figura 1. En el primer nivel del árbol los

servicios son clasificados por contexto funcional ya sea de negocio o tecnológico. En el segundo nivel, cada nodo es dividido a su vez en abstracto o legado. El término abstracto indica que los servicios aún no han sido implementados pero hacen parte del portafolio de servicios y el término legado indica que el servicio ya está implementado en la organización. El tercer nivel clasifica los servicios en atómicos o compuestos. Un servicio *atómico* es una pieza de software que es indivisible. Un servicio *compuesto*, significa que se compone de otros servicios ya sean atómicos o compuestos. Los servicios se componen para ofrecer capacidades más complejas.

**Ecosistema SOA:** las arquitecturas orientadas a servicios se representan como *ecosistemas* ya que al igual que en un ecosistema biológico, una arquitectura SOA no puede entenderse como una simple descomposición de sus partes y subsistemas, puesto que es necesario observar muchas y complejas interacciones entre las partes[9]. La figura 2 presenta un ecosistema de servicios, utilizando la notación propuesta en SOMF [3].

Esta vista de un ecosistema describe la orquestación de los servicios en un escenario de implementación de un proceso de negocio. Cada ecosistema está compuesto de elementos llamados *Zona*, los cuales definen agrupaciones lógicas de consumidores, servicios y proveedores . La zona de servicios es utilizada para expresar el intercambio de mensajes entre servicios del modelo de portafolio o entre un servicio y sus consumidores y las condiciones en las cuales es ejecutado el servicio. Los consumidores se representan por medio de triángulos y las relaciones de intercambio de mensajes se representan por medio de flechas. La direccionalidad de las flechas representa el tipo de intercambio de mensajes.

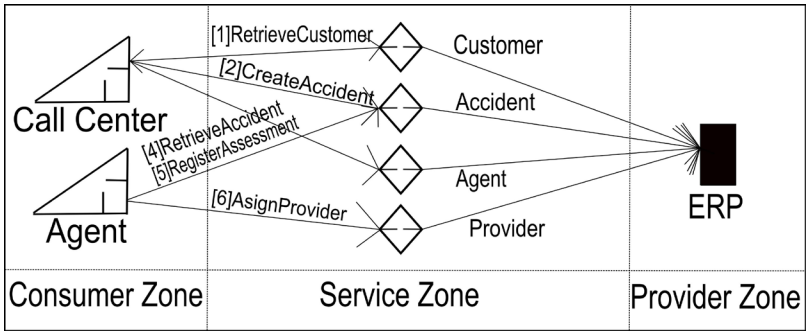


**Figura 2.** Ecosistema SOA

## 2.2 Gobierno SOA

Gobierno en un contexto general es el proceso de tomar las decisiones apropiadas en beneficio de los interesados en estas decisiones u opciones[2]. En el contexto SOA, podemos decir que el gobierno SOA es el proceso de asegurar que todos los intereses de los participantes en la arquitectura son tenidos en cuenta en la planeación, diseño y ejecución de la SOA de una organización. En este sentido el gobierno SOA hace referencia a la organización, procesos, procedimientos, políticas y métricas requeridas para administrar una SOA exitosamente, entendiendo como exitosa una SOA que conoce y promueve los objetivos del negocio todo el tiempo[2].

En el proceso de gobernar una arquitectura SOA se definen, implementan y ejecutan decisiones de los interesados en la arquitectura. Estas decisiones deben ser modeladas dentro de un marco que asegure que la organización tiene una estrategia SOA adecuada, alineada con los objetivos de negocio y que esta estrategia se ejecuta de acuerdo con las directrices y restricciones definidas por medio de políticas SOA.

Las claves del gobierno SOA son: La definición de políticas de gobierno SOA y su gestión para garantizar su cumplimiento. Un modelo de gobierno SOA robusto debe definir políticas claras y aplicables y la forma de poner en práctica los procesos de gobierno SOA a través de todo el ciclo de vida de la SOA dentro de la organización. Estas políticas deben dirigirse a garantizar que los servicios se identifiquen, diseñen e implementen de tal forma que sean reusables e interoperables.El análisis del impacto de los cambios en la arquitectura para garantizar que su evolución sea coherente con las necesidades de la organización. Este análisis debe realizarse: durante la planeación y el diseño de las decisiones arquitecturales para garantizar que se están tomando las decisiones adecuadas, durante su implementación para garantizar que se están desarrollando de la manera correcta y durante su ejecución para asegurarse de que estas decisiones producen el comportamiento deseado y cumplen con las políticas de gobierno que rigen la arquitectura.

## 2.3 Situación actual del Gobierno SOA

Para apoyar los procesos de gobierno SOA, los arquitectos de TI y administradores de la plataforma deben contar con instrumentos que faciliten el análisis de las decisiones que afectan la arquitectura. Sin embargo, las prácticas y herramientas de gobierno SOA se encuentran en una etapa temprana de su evolución.

En la actualidad los mecanismos para definir y asegurar el cumplimiento de las políticas de gobierno SOA, son en su mayoría manuales. Las políticas de gobierno SOA, son en muchos casos, expresadas a través de documentos no estructurados que se dan a conocer a algunos de los arquitectos

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

para que sean tenidos en cuenta en el diseño e implementación de los nuevos servicios pero que no son verificadas a través de procesos o mecanismos estrictos y disciplinados. Según una encuesta desarrollada por Oracle [5] en la que participaron más de 500 empresas, se encontró que 9 de cada 10 compañías no se sienten satisfechas con sus metodologías de gobierno SOA. El 85% de estas empresas implementa controles y procedimientos manuales para implementar el gobierno en las fases de diseño de sus decisiones arquitecturales, el 56% admite que por lo menos la mitad de sus artefactos y servicios no son revisados antes de pasar a producción y consideran que los controles manuales son insuficientes e ineficientes dada la complejidad de este tipo de arquitecturas. No obstante, para el 49% de las organizaciones, el modelo de gobierno es un factor crítico puesto que sin un modelo de gobierno la implementación de SOA fallará. Para el 44% el gobierno SOA es importante y requerido mientras que para el 7% este es factor que carece de importancia.Esta encuesta también reveló que para el 85% de estas organizaciones, es importante contar con un modelo de gobierno SOA en las etapas de planeación y diseño de decisiones arquitectónicas para prevenir problemas que de detectarse en las etapas de implementación y ejecución de las SOA pueden generar costos mayores en su corrección. Sin embargo, aunque para estas organizaciones implementar un modelo de gobierno SOA a través de todo el ciclo de vida de la arquitectura y de cada uno de los servicios es un factor clave de éxito, la mayoría de estas organizaciones se encuentra en fases de planificación y exploración de alternativas de gobierno SOA. Para el 59% de estas organizaciones uno de los factores claves deseables en el gobierno SOA es tener mecanismos para analizar el impacto de los cambios (o cambios potenciales) y entender cómo las modificaciones de los servicios individualmente impactarán el resto de la arquitectura. Teniendo en cuenta que uno de los grandes retos de las arquitecturas SOA es hacer frente al cambio en las organizaciones, el gobierno SOA debe contar con mecanismos que faciliten adaptarse a estos cambios en la arquitectura. En la actualidad, la tecnología y las normas de gobierno de SOA, y en particular la aplicación de políticas, son relativamente inmaduros [2]. Las siguientes son algunas de las especificaciones que existen para SOA basadas en servicios web: WS-Policy Framework, WS-MetadataExchange, WS-Addressing, WS-MessageDelivery y Web Services Policy Language. Estas especificaciones emergentes fundamentalmente se basan en las normas establecidas para los servicios Web como SOAP, WSDL, UDDI, XML y XML Schema. Estas propuestas presentan las siguientes limitaciones:La definición de la política está acoplada al contrato del servicio, esto produce redundancia en la definición de las políticas y dificulta su administración. Este tipo de propuesta permite la definición de políticas que se limitan a la arquitectura del servicio, dejando de lado los niveles superiores de la arquitectura, es decir, la composición de los servicios, la administración del portafolio de servicios y el nivel de la arquitectura empresarial. La verificación de estas políticas se realiza en tipo de ejecución, por esta razón se dirigen a detectar y monitorear el cumplimiento de las políticas más que a prevenir los problemas asociados a su no cumplimiento.

## III. DESARROLLO DIRIGIDO POR MODELOS

La propuesta que presentaremos en la sección 4 para apoyar la gobernabilidad de las arquitecturas SOA se basa fundamentalmente en el desarrollo dirigido por modelos, en esta sección presentamos los principales conceptos que hemos tenido en cuenta en el desarrollo de este proyecto, especialmente las transformaciones basadas en grafos. El desarrollo dirigido por modelos es una metodología para describir y manipular sistemas y procesos complejos. Este enfoque trata de resolver esta complejidad describiendo los sistemas y procesos a través de la definición de modelos que se usan para la implementación y análisis del sistema. Mediante la definición de modelos se facilita la manipulación de sistemas grandes y complejos, ya que es posible simplificar y expresar estructura y comportamiento de un sistema y administrar y analizar las distintas dependencias entre los elementos.

A continuación se presenta los principales conceptos del desarrollo dirigido por modelos.

***Modelo.*** Un modelo es un conjunto de sentencias que describen un sistema y su entorno, esta descripción puede incluir el comportamiento deseado del sistema. Por ejemplo, un modelo puede describir una arquitectura de software así como su evolución esperada.

***Metamodelo***. Un metamodelo es un modelo que sirve para describir otro modelo. Dicho de otra manera un metamodelo es una abstracción de alto nivel que permite delimitar el dominio específico de un problema definiendo los elementos que se pueden modelar, sus relaciones y las restricciones que aplican en la construcción de estos modelos. De esta manera es posible generar modelos de un dominio específico limitándose a estos elementos y a las reglas que condicionan la forma de relacionarse. La relación que existe entre un modelo y el metamodelo que lo describe es una relación de *conformidad*, es decir, que el modelo es *conforme a* el metamodelo.

***Transformaciones.*** Uno de los mecanismos más importantes que ofrece el desarrollo dirigido por modelos son las transformaciones. Una transformación es el proceso en el cual un modelo es convertido a otro modelo. Este proceso permite integrar, generar, refinar y abstraer modelos. Los elementos necesarios para ejecutar una transformación son: un modelo de entrada, el metamodelo que lo describe, así como el metamodelo que describe el modelo esperado y las condiciones que relacionan los elementos de entrada con los elementos en que se transformarán y las relaciones que existirán entre los elementos de salida; estas condiciones se conocen como reglas de transformación y se definen a través de un lenguaje de transformación.

### 3.1 Transformaciones basadas en grafos y EMFT

Estas transformaciones se basan en la idea de que un modelo se puede interpretar como un grafo con características semánticas adicionales, como la herencia y las relaciones de multiplicidad, así como información adicional como atributos de los elementos y relaciones. De esta manera las transformaciones entre modelos se pueden definir usando conceptos de comparación de grafos. Para esto se define el modelo como un grafo objetivo y las condiciones de búsqueda se definen a través de un grafo llamado patrón. Tanto el grafo objetivo como el grafo patrón son modelos conformes al mismo metamodelo. Mediante algoritmos de comparación se define si el grafo patrón es o no un subgrafo del grafo objetivo. En la figura 3 presenta un ejemplo de grafos donde el modelo superior es el grafo objetivo y modelo de la parte inferior es el grafo patrón, las flechas que relacionan estos elementos nos permite verificar gráficamente que el modelo patrón es un submodelo del modelo objetivo. Este algoritmo es conocido como: *subgraph matching*, *graph pattern matching*, ó *subgraph isomorphism problem*.Las reglas de transformación basadas en grafos se definen especificando una combinación de: un grafo LHS (left-hand-side) que define las precondiciones que se deben cumplir en el grafo objetivo para aplicar la regla. Un grafo RHS (right-land-side) que describe el resultado o post-condición de la regla. Uno o más grafos(s) NAC (Negative Application Conditions) que define condiciones que deben *no cumplirse* para permitir la aplicación de la regla, y *mappings* entre los elementos de estos grafos.

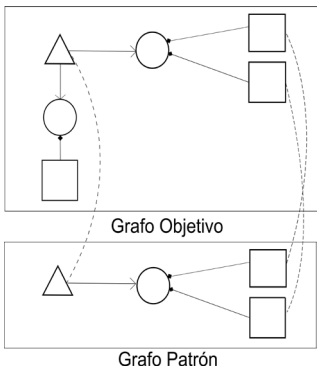


**Figura 3.** Ejemplo de comparación de grafos

Este tipo de transformaciones, nos permite buscar un conjunto de elementos en un modelo EMF, de acuerdo con el grafo LHS y sobre-escribirlo (agrega, edita, borra o mantiene elementos) comparando los elementos del grafo LHS frente al grafo RHS y los enlaces entre sus elementos, de manera que si un elemento se encuentra en el grafo LHS pero no así en el grafo RHS, este elemento deberá ser eliminado. Por el contrario si un elemento se encuentra en el grafo RHS y no en el grafo LHS este elemento deberá ser agregado al modelo de salida.La figura 4 presenta un ejemplo de regla de transformación basada en grafos. En la parte superior se presenta el modelo de entrada, en la parte intermedia se presenta la regla de transformación y en la parte inferior se presenta el modelo que resulta después de ejecutar la regla. Al comparar los elementos del modelo LHS con los elementos del modelo RHS de la regla se puede determinar que si en el modelo objetivo se encuentra un subgrafo similar al modelo LHS, a este grafo se le deberá borrar el elemento cuadrado. Las flechas que relacionan los elementos del grafo LHS con los elementos del modelo de entrada indican que en este sector del modelo objetivo se ha encontrado un subgrafo igual al grafo LHS. La flecha que relaciona el círculo del modelo RHS con el modelo de salida, indica que este elemento de la arquitectura ha sido modificado de acuerdo con las condiciones del modelo RHS y por esta razón ya no presenta el elemento cuadrado. El diseño de estas reglas de transformación se concentra en *qué elementos* del modelo EMF deseamos buscar y *qué operaciones* deseamos realizar sobre estos elementos, más que en cómo realizar estas operaciones.En este trabajo hemos usado el framework de Eclipse EMF Tiger [4], el cual permite la definición y ejecución de este tipo de transformaciones. Estas reglas pueden ser transformadas a reglas para AGG [8] que es un motor de transformaciones basados en grafos que ejecuta la transformación y devuelve el resultado en un modelo EMF. También es posible generar código java para lograr una ejecución más eficiente de la transformación.

## IV. ARCHIVOL

La estrategia que se presenta en este artículo, se basa en la utilización del metamodelo *Archivol*. Este metamodelo ha sido construido en el proyecto de investigación Moosas[1], y apoya diversos proyectos de investigación orientados a realizar descripción, descubrimiento, análisis y evaluación de soluciones arquitectónicas en diferentes dominios de aplicación, incluidas las SOA. Estructuralmente, este metamodelo está dividido en varios módulos. En esta sección se presentan los elementos del metamodelo que se utilizaron para el diseño e implementación de esta propuesta.

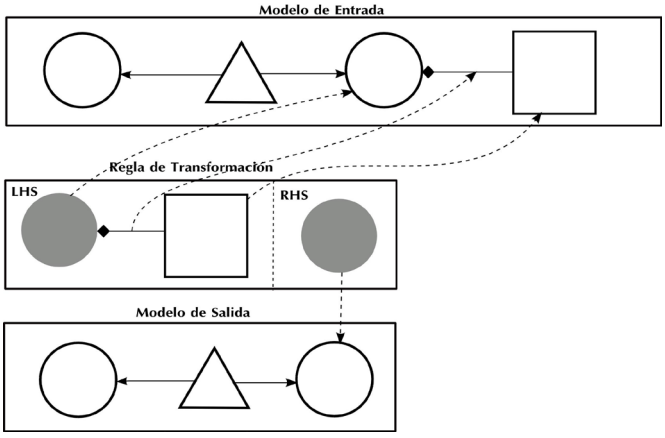


**Figura 4.** Ejemplo de aplicación de Regla de transformación

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

### 4.1 Módulo Architectural Commons

Este módulo define los elementos transversales requeridos en la descripción de arquitecturas de solución. La figura 5 presenta los elementos que conforman este módulo:

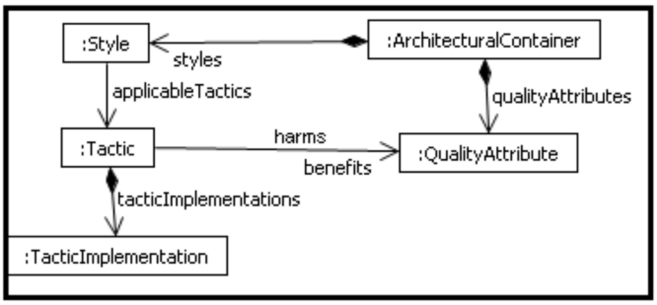


**Figura 5.** Módulo ArchitecturalCommons de *Archivol*

*ArchitecturalContainer:*este elemento es el contenedor de todos los elementos que componen y describen una arquitectura de software.

*Style:* expresa los estilos arquitecturales que aplican en la descripción de las arquitecturas de software. Un estilo arquitectónico es una especialización de los elementos y relaciones, así como con un conjunto de restricciones sobre cómo pueden ser utilizados..

*Tactic:* son decisiones que se aplican para promover algunos atributos deseados en la arquitectura o para prevenir situaciones no deseadas. Las Tácticas arquitecturales pueden beneficiar algunos atributos de calidad mientras perjudica a otros.

*TacticImplementation:*describe formas de implementación de tácticas arquitecturales.

*QualityAttribute:* los atributos de calidad son las características deseables en la arquitectura expresada.

### 4.2 Módulo Candidate Architecture

Este módulo permite describir una arquitectura candidata, la cual puede ser expresada desde diferentes puntos de vista y puede ser evaluada. Los elementos que conforman este módulo son:

*CandidateArchitecture:* describe una arquitectura propuesta para lograr determinados requerimientos y escenarios de calidad. Esta arquitectura candidata puede ser expresada a través de un conjunto de puntos de vista, contiene uno o más elementos arquitecturales y aplica para cero o más decisiones arquitecturales.

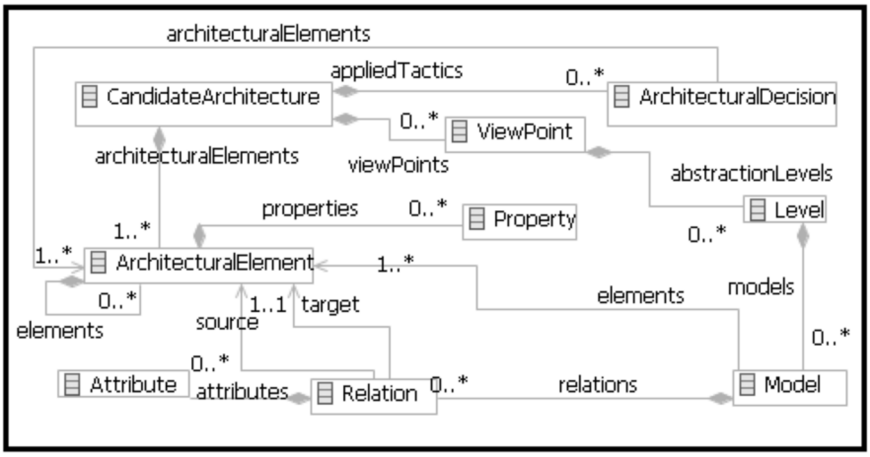


**Figura 6.** Módulo CandidateArchitecture de Archivol

*ViewPoint:* un punto de vista es una especificación de las convenciones usadas para construir y usar vistas de una arquitectura.[9]. El punto de vista es identificado por un nombre y tiene relacionado un tipo de vista. Cada punto de vista puede contener cero o más niveles.

*Level:* permite clasificar los distintos modelos de acuerdo a su nivel de abstracción. Un nivel es identificado por un nombre, tiene una descripción y puede definir cero o más modelos arquitecturales.

*Model:* un modelo arquitectural es una abstracción o representación de algún aspecto de la arquitectura. Cada modelo es desarrollado usando los métodos y convenciones asociados al punto de vista al que pertenece. [10]. Estos modelos tienen asociado un tipo que puede tener el valor dinámico o estático. Los modelos dinámicos representan aspectos del comportamiento de la arquitectura mientras que los modelos estáticos representan aspectos estructurales de la arquitectura. Cada modelo puede ser definido por una notación como UML y es identificado por un nombre. Un modelo tiene relación con los elementos arquitecturales de la arquitectura candidata y a partir de estos elementos define un conjunto de relaciones.

*ArchitecturalElement*: los elementos arquitecturales son los componentes que conforman la arquitectura, por ejemplo para el caso de SOA, un servicio es definido a través de un elemento arquitectural.

*Property:* las propiedades definen las características de los elementos arquitecturales.

*Relation:* define la forma como los elementos arquitecturales se relacionan entre sí, estas relaciones tienen cero o más atributos. Una relación tiene como origen y destino un elemento arquitectural, la relación es identificada por un nombre, tiene un tipo, una direccionalidad y permite ser visible u oculta para el modelo.

*ArchitecturalDecision:* este elemento agrupa todos los elementos arquitecturales involucrados en la aplicación de un táctica arquitectural.

### 4.3 Módulo Expressions

La figura 7 presenta los elementos que conforman este módulo. Este módulo permite definir distintos tipos de condiciones utilizados en metodologías de evaluación y análisis de arquitecturas de software. En este caso particular permite describir la forma de implementación de una táctica arquitectural. La figura 7 presenta los elementos que conforman este módulo. El diseño de este módulo está inspirado en las bases de la lógica de predicados a través de la cual es posible expresar relaciones entre objetos así como sus atributos. Cada predicado está representado por el elemento *expression*, las cuales contiene uno o más operandos que pueden ser términos ó valores. Cada objeto se representa con el elemento *term*. La relación existente entre dos o más términos o entre un término y sus atributos, se expresa mediante el elemento *operator* mientras que los valores de los atributos se representan por el elemento *SingleValue* el cual contiene el nombre y el valor del atributo.

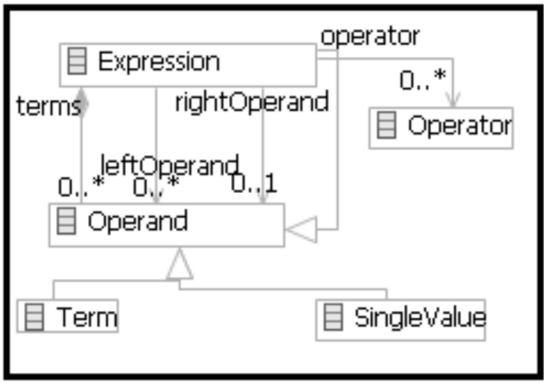


**Figura 7.** Módulo Expressions en Archivol

### 4.4 Módulo Architectural Evaluation

Este módulo permite expresar los resultados de aplicar metodologías de evaluación sobre las arquitecturas candidatas. La figura 8 presenta los elementos que conforman este módulo. Una arquitectura candidata definida en el módulo *CandidateArchitecture* puede tener distintas evaluaciones representadas por el elemento *ArchitecturalEvaluation*. Cada evaluación arquitectural está compuesta por un conjunto de salidas de evaluación representada por el elemento *EvaluationOutput*, estas salidas de evaluación pueden ser de tipo cuantitativo o cualitativo, esto se representa mediante los elementos *QuantitativeOutput* y *QualititativeOutput* respectivamente. Las salidas de tipo cualitativo implican razonamientos o conclusiones acerca de los elementos que son evaluados, mientras las salidas de tipo cuantitativo representan medidas aplicadas a los elementos arquitecturales. Cada salida de evaluación referencia los elementos arquitecturales que han sido evaluados.

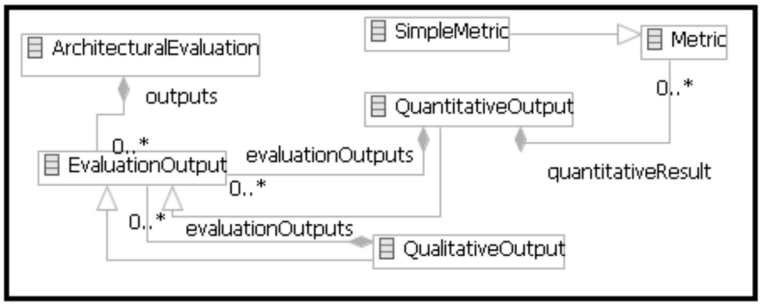


**Figura 8.** Módulo ArchitecturalEvaluation en Archivol

## V. ESTRATEGIA DE GOBIERNO SOA DIRIGIDA POR MODELOS

En esta sección presentamos nuestra estrategia para apoyar los procesos de gobierno de las arquitecturas orientadas a servicios. En la primera parte ofrecemos una visión general de la solución, y en la segunda parte ilustramos la implementación y los pasos que se deben seguir para la ejecución de esta estrategia a través de un ejemplo.

### 5.1 Descripción general de la propuesta de solución

Esta propuesta busca apoyar los procesos de gobierno de una arquitectura SOA, ofreciendo una herramienta que facilita el análisis del impacto de los cambios (o cambios potenciales) para entender cómo una modificación en algún punto de la arquitectura puede afectar el resto de la misma. Para esto se basa en el gobierno SOA dirigido por políticas[2]. Como se mencionó en la sección 2, mediante la definición de políticas se busca determinar las situaciones que se desean promover o evitar en la arquitectura, de manera que esta evaluación pretende identificar si la arquitectura de solución cumple o no las políticas definidas.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

En esta estrategia, vemos la *gobernabilidad* como un atributo deseable de la arquitectura como el grado de cumplimiento de las políticas de gobierno. En general, la evaluación de la arquitectura debería permitirnos identificar problemas potenciales lo más temprano posible en el ciclo de vida de la decisión arquitectural; en este caso antes de su implementación.

Nuestra estrategia se divide en tres etapas principales. La primera etapa es la descripción de la arquitectura de solución que se desea evaluar. Esta descripción se realiza a través de un lenguaje de dominio específico llamado *DSL-SOA*. Con este lenguaje es posible describir un punto de vista de la arquitectura, compuesto por un modelo estático que corresponde con el portafolio de servicios y uno o varios modelos dinámicos que corresponden a cada uno de los ecosistemas que hacen parte de la arquitectura. La definición de la arquitectura por medio de un lenguaje de dominio específico permite describir las decisiones arquitecturales que se desean aplicar, para estudiarlos al nivel de estos modelos antes de iniciar su implementación.

La segunda etapa consiste en describir las políticas de gobierno que se desean implementar en la arquitectura. Para esto se hace uso del lenguaje de dominio específico llamado *govSOA*. Las políticas de gobierno pueden aplicarse tanto al portafolio de servicios como a cada uno de los ecosistemas.

En el caso del portafolio de servicios, se busca garantizar que los servicios evolucionen de una manera controlada durante todo su ciclo de vida. Para esto se definen políticas dirigidas a controlar: la identificación y clasificación de los servicios, la descripción y estandarización de sus contratos e interfaces y la composición/descomposición de los servicios.

En el caso de los ecosistemas, se busca controlar los escenarios de orquestación en los que los servicios son ejecutados para dar cumplimiento a los procesos de negocio, es decir cómo interactúan con sus potenciales consumidores, ya sean clientes u otros servicios.

Una de las principales fuentes de políticas son los patrones de diseño puesto que están dirigidos a prevenir situaciones problemáticas comunes. Sin embargo, las políticas de gobierno se definen de manera particular según las necesidades de cada organización y deben evolucionar con la arquitectura.

Una vez definidas la arquitectura de solución y las políticas de gobierno, es posible ejecutar la tercera etapa que corresponde con el procedimiento de evaluación. El primer paso es identificar en qué puntos de la arquitectura debe aplicarse la política, para luego evaluar si estos elementos cumplen o no las restricciones definidas en la política. Para esto se realiza un proceso de comparación basado en el algoritmo *subgraph matching* descrito en la sección 3. El modelo de la arquitectura de solución hace las veces de grafo objetivo y cada política hace las veces de grafo patrón.

El producto de esta evaluación, es un árbol de resultados compuesto por tres niveles.

El primer nivel relaciona la política que se está evaluando y da un resultado general que indica si la política se cumplió o no, es decir que este resultado es *TRUE* si la política se cumplió todas las veces que se debió cumplir ó *FALSE* si la política dejó de cumplirse por lo menos una vez. Adicionalmente brinda información de resumen como cuántas veces se evaluó la política, cuántas veces el resultado de la evaluación es positivo y cuántas veces es negativo.

En el segundo nivel se encuentran los resultados de cada una de las veces que la política se evaluó. En este nivel se referencian cada uno de los elementos de la arquitectura que fueron evaluados y se da un resultado que indica si estos elementos cumplen o no la política. Es decir que este resultado es *TRUE* si se cumplen todas y cada una de las restricciones que impone la política y *FALSE* si al menos una de las restricciones no se cumple.

En el tercer nivel se da información detallada sobre el cumplimiento de cada una de las restricciones que impone la política. Este nivel da como resultado *TRUE* si la condición se cumple y *FALSE* en caso contrario. En este nivel se referencia los elementos arquitecturales, propiedades o relaciones evaluados y la condición que se está aplicando. (Ejemplo: la fecha de vencimiento del servicio debe ser mayor que hoy.) Este procedimiento de evaluación puede ser aplicado en momentos claves en el gobierno de una arquitectura, algunos de estos momentos son: Antes de implementar una decisión arquitectural, para validar si esta decisión cumple las políticas que gobiernan esta arquitectura, previniendo así la implementación de decisiones arquitecturales que puedan generar situaciones no deseadas o los riesgos que la implementación de las políticas puedan mitigar. Periódicamente para validar si las políticas de gobierno vigentes se están cumpliendo en la arquitectura en un momento dado. Antes de implementar una nueva política de gobierno para identificar los elementos que se ven involucrados y obtener información que le permita al arquitecto deducir el esfuerzo requerido en la implementación de esta política y posibles impactos negativos que puedan generarse.

### *5.2 Ejemplo de Aplicación de la estrategia*

A continuación ilustramos cada etapa de esta estrategia y presentamos las herramientas diseñadas y su aplicación con un ejemplo.Vamos a suponer una organización que ofrece pólizas de aseguramiento para siniestro de automóviles. El proceso de negocio *Registrar Siniestro*, a través del cual se atienden los siniestros de los clientes garantizando la atención de los accidentes de vehículos que tienen pólizas con la compañía, inicia con el registro del siniestro y las partes involucradas y finaliza con la asignación del taller que reparará el vehículo. La figura 9, presenta las actividades que conforman este proceso, estas actividades son:

*Registrar Siniestro:* el proceso comienza cuando el servicio de atención recibe una llamada para informar que ha ocurrido un siniestro, en este momento se registran los datos básicos del cliente,

el lugar del siniestro y los detalles que el cliente suministre.

*Asignar Representante Móvil:* en seguida se asigna y notifica a un representante móvil en el área del siniestro para que asista al cliente en el lugar donde ha ocurrido.

*Registrar Información detallada del Siniestro:* una vez el representante móvil llega al lugar de los hechos recopila la información detallada del siniestro (datos del vehículo(s) implicado(s), detalles técnicos del Siniestro, entre otros) y realiza el registro.

*Registrar Evaluación preliminar de daños:* el representante realiza una evaluación de los daños y registra los datos de las reparaciones que considera se deben realizar.

*Asignar Taller:* el representante asigna un taller para que realice la reparación del vehículo y notifica al cliente.

### 5.3 Etapa uno: Descripción de la Arquitectura

Como ya hemos mencionado, la primera parte en la ejecución de la estrategia es expresar tanto el portafolio de servicios, como cada uno de los ecosistemas que se implementan en la organización. Para esto se ha diseñado un lenguaje de dominio específico llamado *DSL-SOA*.

Las figuras 1y 2 presentadas en la sección 2 corresponde con el portafolio de servicios y el ecosistema SOA requerido para implementar el proceso de negocio *Registrar Siniestro*, respectivamente.A continuación veremos cómo esta arquitectura se describe a través de lenguaje de dominio específico *DSL-SOA*. La figura 10 presenta una visión condensada de una arquitectura. En esta figura se destacan los principales segmentos en la definición de la arquitectura.

La definición de la arquitectura inicia en la línea uno con la palabra reservada *Architecture*. En la línea 2 se presenta el segmento de definición de parámetros generales que aplican a la arquitectura, este segmento inicia con la palabra reservada *Parameters*. Ejemplo de parámetros de la arquitectura son: los tipos de datos válidos ó los estados en el ciclo de vida del servicio, estos parámetros son adaptables a las necesidades de cada arquitectura particular.

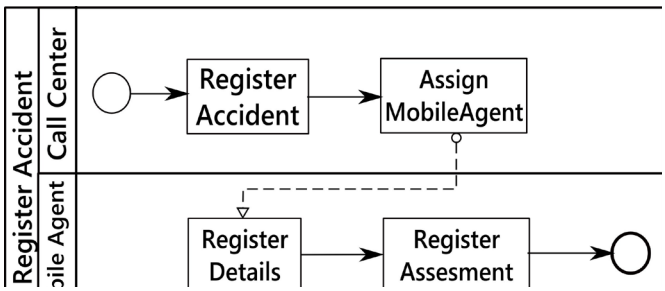


**Figura 9.** Proceso de negocio *Registrar Siniestro*

El siguiente segmento es el punto de vista SOA, este inicia en la línea 19 con la palabra reservada *ViewPoint*, este segmento está conformado por dos subsegmentos que corresponden a dos modelos dentro de la arquitectura, el primer modelo es el portafolio de servicios, el cual es un modelo estático que contiene todos los servicios que conforman la arquitectura, la definición de este subsegmento inicia en la línea 20 con la palabra reservada *Portfolio* y finaliza en la línea 49 con la expresión *End Portfolio*. En el siguiente subsegmento se presentan los modelos dinámicos del punto de vista SOA, estos modelos corresponden con los ecosistemas SOA que conforman la arquitectura. La definición de cada ecosistema inicia con la palabra reservada *Ecosystem* y finaliza con la expresión *End Ecosystem*.

Estructura.SOA

```
 1 Architecture
 2   Parameters
19   ViewPoint SOA
20     Portfolio
21      Atomic Services [
22       Service Customer [business,visible,legacy,active]
49     End Portfolio
50     Ecosystem RegisterAccident
51       Consumer Zone [CallCenter]
52       ServicesZone[Customer]
53       ProvidersZone[ERP]
54     End Ecosystem
55   End ViewPoint
56   Consumer CallCenter
57   Provider ERP [active,source = commercial]
157 End Architecture
```

**Figura 10.** Principales segmentos en el DSL *DSL-SOA*

A continuación presentamos algunos ejemplos que ilustran el cuerpo de estos segmentos.La figura 11 ilustra la definición de un servicio atómico usando el lenguaje de dominio específico *DSL-SOA*. El ejemplo corresponde con la definición del servicio *Customer*, el cual hace parte del portafolio de servicios. La definición de este servicio inicia en la línea 22 con la palabra reservada *Service* seguida por el nombre del servicio, en esta caso *Customer*. En la misma línea se presentan algunas propiedades del servicio que son obligatorias y que permiten clasificar al servicio en el portafolio de servicios. La primera propiedad indica el contexto de aplicación del servicio, en este caso se trata de un servicio de negocio lo que se indica mediante el parámetro *business*, la siguiente propiedad indica si este servicio es visible o no directamente para los consumidores de la arquitectura, la siguiente propiedad indica el estado de implementación del servicio, en este caso se trata de un servicio legado lo que se indica con el parámetro *legacy* y la siguiente propiedad indica el estado del servicio en el ciclo de vida, en este caso el servicio se encuentra activo. Entre las líneas 23 y 34 se definen los mensajes que intercambia el servicio para ejecutar sus operaciones. En este caso se presentan dos mensajes de solicitud de operación identificados con la palabra reservada *requestmessage*, la información que se requiere para ejecutar está operación se identifica con la palabra reservada *input*. El primer mensaje llamado *createCustomer* solicita la creación de un cliente, este mensaje requiere un identificador y nombre del cliente, el siguiente mensaje llamado *retrieveCustomer* solicita la información de un cliente, este mensaje requiere un identificador del cliente. Los mensajes de respuesta se identifican con la palabra reservada *responseMessages*, estos mensajes contienen la información que retornan las operaciones que ejecuta el servicio, esta información se identifica con la palabra reservada *output*. En este caso el mensaje llamado *responseCreateCustomer* da como respuesta si la operación se realizó satisfactoriamente o no, por su parte el mensaje *retrieveResponseCustomer* retorna el nombre y el identificador del cliente. Entre las líneas 35 y 39 se define la interfaz del servicio, la cual se identifica con la palabra reservada *interface* y las operaciones que ofrece las cuales se identifican con la palabra reservada *Operation*, a cada operación se le asignan los mensajes de tipo *requestmessage* y *responsemessage* correspondiente. Entre las líneas 40 y 42 se definen los puntos de acceso a través de los cuales se exponen las operaciones que el servicio ofrece, estos se identifican con la palabra reservada *EndPoints*. En la línea 4 se inicia la definición del contrato del servicio indicando algunas propiedades no funcionales del servicio como la fecha de expiración y finalmente se encuentra la expresión *End Service* para indicar el final de la definición de este servicio. En la figura 12 se ilustra la descripción de un ecosistema con el DSL *SOA*. Este ejemplo corresponde con un ecosistema que implementa un proceso de negocio llamado *Registrar Siniestro*. En la línea 218 se inicia la identificación del ecosistema con la palabra reservada *Ecosystem* seguida por el nombre del ecosistema.

El ecosistema esta divido en tres zonas, la primera es la zona de consumidores identificada con la palabra reservada *CustomerZone* en la línea 218, la cual contiene cada uno de los consumidores que participan en el ecosistema, en este caso *Call Center* y *Agent*. La segunda es la zona de servicios identificada con la palabra reservada *ServiceZone* en la línea 227, esta zona relaciona los servicios definidos en el portafolio de servicios y que se ejecutan en este ecosistema, para este caso contiene los servicios *Customer, Accident, Agent y Providers*. La tercera es la zona de proveedores en la cual se agrupan los sistemas que ofrecen las funcionalidades que exponen los servicios, en este caso las funcionalidades son ofrecidas por el sistema *ERP*.

Architecture.SOA

```
22  Service Customer [business,visible,legacy,active]
23    requestMessages
24      createCustomer
25        input idCustomer[INTEGER,required,one]
26        input nameCustomer [INTEGER,required,one];
27      retrieveCustomer
28        input idCustomer[INTEGER,required,one];
29    responseMessages
30      createResponseCustomer
31        output sucess [BOOLEAN,one];
32      retreiveResponseCustomer
33        output idCustomer[INTEGER,one]
34        output nameCustomer [INTEGER,one];
35    interface ICustomer
36      Operations{
37        createCustomer  input = createCustomer  output = createResponseCustomer
38        retrieveCustomer    input = retrieveCustomer output =  retreiveResponseCustomer
39       }
40    endPoints
41      EPCreateCustomer   binding {interface = ICustomer, operation = ICustomer.createCustomer};
42      EPRetrieveCustomer binding {interface = ICustomer, operation = ICustomer.retrieveCustomer}
43    contract SLACustomer
44      properties state ["Active":STRING,one];
45                 expirationDate ["'01/10/2020'":DATE,one];
46      End Service
```

**Figura 11.** Definición de un servicio atómico usando el DSL *SOA*

Entre la zona de consumidores y la zona de servicios se encuentran las conexiones que se pueden establecer entre estos, esta sección se identifica con la expresión *Connections Customer to Service*. Entre la zona de servicios y proveedores se encuentran las conexiones que se pueden establecer entre los servicios y entre los servicios y los proveedores. La ubicación de estos servicios se identifican con las expresiones *Connections Service to Service* y*Connections Service to Provider* respectivamente. Cada conexión se define entre los símbolos: *[ ]*, al lado izquierdo se ubica el elemento origen seguido por el tipo de relación y el elemento destino.

Al finalizar esta etapa se cuenta con el modelo de la arquitectura, el cual debe reflejar las decisiones arquitecturales que se desean evaluar, este modelo es conforme al metamodelo que soporta el DSL *DSL-SOA*.

### *5.4 Etapa dos: Descripción de las Políticas de Gobierno SOA*

SegurosLosAlpes.SOA

```
Ecosystem RegisterSinister
  Consumer Zone [CallCenter, Agent]
    Connections Customer to Services
      [CallCenter"->"Customer.EPRetrieveCustomer]
      [CallCenter"<->"Accident.EPCreateAccident]
      [CallCenter"<->"Agent.EPAssignAccident]
      [Agent"<->"Accident.EPRetrieveAccident]
      [Agent"<->"Accident.EPRegisterAssessment]
      [Agent"<->"Accident.EPAssignProvider]
  ServicesZone[Customer, Accident, Agent, Providers]
  Connections Service to Service
  Connections Service to Provider
    [Customer.EPRetrieveCustomer"<->"ERP.retrieveCustomer
       uses (MessageTransformer)]
    [Accident.EPCreateAccident"<->"ERP.createAccident]
    [Agent.EPAssignAccident"<->"ERP.assignAccident]
    [Accident.EPRetrieveAccident"<->"ERP.retrieveAccident]
  ProvidersZone[ERP]
End Ecosystem
```

**Figura 12.** Definición de un ecosistema usando el DSL *SOA*

En el etapa dos, el arquitecto debe expresar la(s) política(s) que gobiernan la arquitectura. Para esto cuenta con en el DSL *govSOA*. La manera de expresar el portafolio de servicios y los ecosistemas presenta similitud frente a la manera de expresar las políticas que les aplican respectivamente, puesto que cada política representa un patrón válido en la arquitectura.

Para definir las políticas de gobierno es necesario seguir los siguientes pasos:

1. Identificar si la política aplica al ecosistema o al portafolio de servicio.
2. Identificar las partes de la arquitectura que se involucran en el cumplimiento de la política, es decir los elementos, sus propiedades y las relaciones que deben existir entre estos elementos. Estas partes de la arquitectura forman el *contexto* de aplicación de la política, es decir, las situaciones en las que se espera que la política se cumpla.
3. Identificar qué condiciones *deben* cumplir los elementos que se involucran en el cumplimiento de la política. Estas condiciones se conocen como *restricciones*. Estas restricciones pueden definir valores o rangos de valores que deben cumplir las propiedades de los elementos o relaciones que pueden existir entre los elementos.

Siguiendo con nuestro ejemplo, consideremos que esta arquitectura se ha visto afectada por el uso no autorizado y mal intencionado de servicios, lo que ha permitido revelar información confidencial de clientes de la compañía. Esto ocurre a través de los servicios que se ofrecen a los representantes móviles por lo que son considerados consumidores externos no confiables. Los arquitectos administradores de la arquitectura desean implementar alguna solución que evite este tipo de ataques. Por esta razón piensan establecer como política de gobierno la implementación del patrón *Service Perimeter Guard*, propuesto en [6]. Este patrón se ilustra en la figura 13, y sugiere usar un servicio intermediario como único punto de contacto seguro para cualquier consumidor externo considerado no confiable que necesite interactuar con los servicios internos.

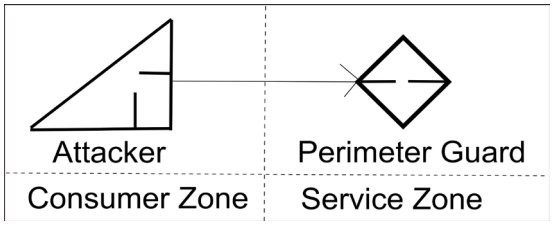


**Figura 13.** Patrón *Service Perimeter Guard*

Sin embargo, la decisión arquitectural de implementar esta política tiene un impacto en la arquitectura de solución,

añadiendo complejidad y afectando el rendimiento, ya que establece una capa de procesamiento intermediario para todas las conexiones de consumidores externos. Por esta razón, es necesario evaluar el impacto de aplicar esta decisión arquitectural y el esfuerzo requerido en su implementación, antes de iniciar su ejecución para evitar invertir recursos en una política que podría tener un impacto negativo ó requerir esfuerzos exagerados frente al beneficio obtenido. Adicionalmente, es necesario contar con herramientas que les permitan garantizar que esta política se acatará en la implementación de decisiones que modifiquen la arquitectura posteriormente.

La figura 14, ilustra la definición de la política *Perimeter Guard* usando el lenguaje de dominio específico *govSOA*.

Las siguientes son las consideraciones que permiten definir esta política.

1. Dado que esta política afecta las interacciones entre los servicios y sus consumidores, podemos decir que aplica a los ecosistemas. Esto se indica en la línea 53 con la expresión *In Ecosystem* seguida por el nombre que identifica la política en este caso *Perimeter Guard*.
2. El contexto de aplicación de esta política define que debe aplicarse en ecosistemas en los cuales se ofrecen servicios a consumidores considerados no confiables. Por lo que se deben identificar aquellos ecosistemas en donde en la zona de consumidores participa un consumidor, cuyo valor de la propiedad *reliability* es: *unreliable*. Esta condición se expresa en la línea 55. Adicionalmente este consumidor debe tener una conexión a un servicio que se encuentra en la zona de servicios del ecosistema correspondiente, esta condición se expresa en la línea 58. Las condiciones de contexto se identifican en el DSL *govSOA* por medio de la palabra reservada *Context* al lado derecho de la condición. La definición de zonas y elementos en estas zonas son por defecto condiciones que definen el contexto de aplicación de la política.
3. Finalmente es necesario definir las restricciones que *deben* cumplir estos elementos, estas condiciones se identifican en el DSL *govSOA* por medio de la palabra reservada *Constraint* al lado derecho de la condición.

SegurosLosAlpes.SOA | government.govSOA

```
52  In Ecosystem PerimeterGuard
53    ConsumerZone [Consumer: consumer
54      conditions {
55        Property: reliability "=" "unreliable" : STRING[Context];
56      }
57    ]
58    Connections Customer to Services [consumer "->"servicio.endpoint [Context]]
59    ServiceZone [
60      Atomic Services [ Service servicio
61        conditions {
62          Property:name "=" "PerimeterGuard":STRING[Constraint];
63        }
64        endPoints endpoint
65      ]
66    ]
67    Times: "MIN" "0";
68  End Policy
```

**Figura 14.** Definición de la política de gobierno *Perimeter Guard*

En este ejemplo se ve que en la línea 62 se expresa una condición de tipo *Constraint* que define que el nombre del servicio al cual se conecta el consumidor *debe ser igual a PerimeterGuard*.

### 5.5 Etapa tres: Evaluación de cumplimiento de las Políticas en la arquitectura

El propósito de la etapa tres es identificar si la arquitectura de solución descrita en la etapa uno cumple o no las políticas descritas en la etapa dos y en qué puntos de la arquitectura.

Al finalizar las etapas uno y dos, contamos con dos modelos, un modelo de la arquitectura SOA el cual es conforme al metamodelo que soporta el DSL *DLS-SOA*, y un modelo de políticas de gobierno el cual es conforme al metamodelo que soporta el DSL *govSOA*. Sin embargo en nuestra estrategia de evaluación es necesario que estos modelos se transformen a modelos conformes al metamodelo *Archivol* descrito en la sección 4.

*Transformación de la arquitectura SOA a una arquitectura candidata Archivol.*En este paso se realiza una transformación que permite expresar la arquitectura SOA que se ha definido en el etapa uno a un modelo conforme al metamodelo *Archivol* usando específicamente el módulo de Archivol *CandidateArchitecture*.

El portafolio de servicios se transforma a un modelo arquitectural de tipo estático y cada uno de los ecosistemas de la arquitectura candidata se transforman en modelos arquitecturales de tipo dinámico. Todas las partes de los servicios, los consumidores y las zonas que conforman los ecosistemas se transforman en *ArchitecturalElements*. Los atributos de estos elementos se transforman al elemento *Property*

y las relaciones existentes entre elementos arquitecturales se transforman al elemento *Relation*.

*Transformación de Políticas de Gobierno SOA a Tácticas conformes a Archivol.*En este paso se realiza una transformación que permite expresar las políticas de gobierno SOA como tácticas arquitecturales en un modelo conforme a Archivol puesto que son decisiones que se aplican para promover algunos atributos deseados en la arquitectura o para prevenir situaciones no deseadas. Esta transformación es una transformación de refinamiento que recibe como entrada el modelo conforme a Archivol generado en el paso anterior y el modelo que contiene las políticas de gobierno el cual es conforme al metamodelo que soporta el DSL *govSOA*. El resultado de este paso es un modelo conforme a Archivol que contiene además de la arquitectura candidata, las políticas de gobierno expresadas como tácticas arquitecturales. Como vimos en la etapa dos, cada política de gobierno es un conjunto de condiciones de contexto y restricciones, estas condiciones se expresan en Archivol mediante el módulo de *Expressions*.

*Generar instrumento de Evaluación de Gobierno SOA* De acuerdo con lo expuesto en los pasos anteriores se puede decir que: El objeto en evaluación es la arquitectura SOA candidata que se expresa a través del módulo *CandidateArchitecture* de *Archivol*, y que está compuesta por elementos arquitecturales, sus propiedades y sus relaciones, en este caso específicamente elementos del contexto SOA. El criterio de evaluación a aplicar, es el grado de cumplimiento de las políticas de gobierno definidas como tácticas arquitecturales y representadas como un conjunto de expresiones en el módulo *Expressions* de *Archivol*.

Ahora es necesario diseñar el instrumento con el cual realizaremos la evaluación. La figura 15 presenta desde una perspectiva gráfica la relación existente entre la arquitectura candidata y la política que se desea evaluar. Esta gráfica presenta en la parte superior la arquitectura candidata y en la parte inferior la política que deseamos evaluar, por medio de flechas punteadas representamos la relación existente entre sus elementos. Como podemos observar la relación existente entre la representación de la arquitectura y la política es similar a la presentada en la figura 3 de la sección 3, en donde se ilustra la comparación de grafos. En este caso la arquitectura hace las veces de modelo objetivo y la política actúa como modelo patrón.

Basados en esta comparación, es posible usar reglas basadas en grafos como instrumento de evaluación, usando cada una de las políticas como condición LHS, es decir, la precondición para buscar los elementos que deben cumplir la política y generando las salidas de evaluación correspondientes en el módulo *ArchitecturalEvaluation* de Archivol a través de las condiciones RHS o post-condiciones de estas reglas.

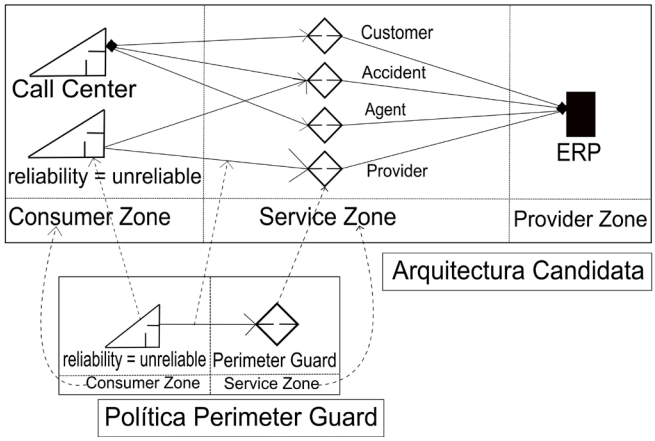


**Figura 15.** Definición de la política de gobierno *Perimeter Guard*

En este paso, se realiza una transformación que convierte las políticas que se han expresado mediante el módulo *Expressions* de *Archivol* en un conjunto de reglas de transformación EMT. En esta transformación se debe generar un conjunto de reglas que identifica qué elementos de la arquitectura deben cumplir la política y un segundo conjunto que verifica que estos elementos cumplan las restricciones definidas. Estas reglas de transformación son como tal el instrumento para realizar la evaluación.

*Ejecución de la Evaluación* Como lo indicamos en el paso anterior, ejecutar la transformación diseñada en el paso anterior permite registrar en el módulo de *ArchitecturalEvaluation* de Archivol las salidas de la evaluación que determinan si las políticas de gobierno de la arquitectura se cumplen o no y en qué puntos de la arquitectura. Después de ejecutar el proceso de evaluación se genera una transformación que permite visualizar un reporte de resultados en una tabla en formato html. En la figura 16, se representa el reporte generado al evaluar el grado de cumplimiento de la política *Perimeter Guard* en la arquitectura de solución de nuestro ejemplo. El primer bloque del reporte presenta el nombre de la política que se está evaluando, en la segunda línea se presenta la cantidad de veces que la condiciones de contexto se cumplen en la política, es decir la cantidad de veces que debió aplicarse esta política, en este caso 2 veces. La tercera línea indica la cantidad de veces

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

que la política se aplicó satisfactoriamente en este caso cero veces ya que es una política que aún no se ha implementado, y el reglón 4 indica la cantidad de veces que la política debió aplicarse pero no fue así, en este caso 2 veces. En seguida se encuentra un bloque de información por cada vez que la política fue evaluada. En el primer reglón de cada bloque se indica el ecosistema que fue evaluado y el resultado de esta evaluación. Este resultado es TRUE si todas las condiciones de *constraint* se cumplen y FALSE si por lo menos una de las condiciones de *constraint* no se cumple.

En este caso las dos veces que la política se evaluó se encontró que no se cumple la política. En seguida se presenta un subbloque que indica los elementos que fueron evaluados en esta ocurrencia. En la primera ocurrencia se evalúo la conexión existente entre el consumidor *Agent* y el servicio *Customer*, en la segunda ocurrencia se evaluó la conexión existente entre el consumidor *Agent* y el servicio *Provider*

| | | | | | |
|---|---|---|---|---|---|
| Policy: Perimeter Guard | | | | | |
| Occurrences: | | | | | 2 |
| Successfull Occurrences: | | | | | 0 |
| Unsuccessfull Occurrences: | | | | | 2 |
| In Ecosystem: | Register Sinister | | Applied: | | FALSE |
| Architectural Elements Evaluated | Agent (CONSUMER)<br>Accident (ATOMIC SERVICE)<br>Connection: (Agent -> Accident) | | | | |
| Details | Element | Property | Value | Expression | Result |
| | Accident | name | Accident | = Perimeter Guard | FALSE |
| In Ecosystem: | Register Sinister | | Applied: | | FALSE |
| Architectural Elements Evaluated | Agent (CONSUMER)<br>Accident (ATOMIC SERVICE)<br>Connection: (Agent -> Accident) | | | | |
| Details | Element | Property | Value | Expression | Result |
| | Accident | name | Accident | = Perimeter Guard | FALSE |

**Figura 16.** Reporte de Evaluación de Gobernabilidad

El siguiente subbloque presenta información acerca de cada una de las restricciones evaluadas, indicando que elemento fue evaluado, que propiedad se evaluó, cual es su valor actual, que condición se aplica en la evaluación y cuál es su resultado. En este caso se espera que el nombre del servicio sea *PerimeterGuard*, pero en ninguno de los dos casos se cumple la condición.

*Análisis de Resultados:* La interpretación de esta información apoya a los arquitectos responsables de esta arquitectura, en la toma de decisiones en cuanto a la pertinencia de implementar está política, puesto que le permite identificar los cambios que deben realizar en la arquitectura para asegurar su cumplimiento, así como los ecosistemas que se verán comprometidos. En este caso, el reporte indica que para que esta política se cumpla, es necesario modificar dos conexiones en el Ecosistema *RegisterSinister*, por lo que el consumidor *Agent* y todos los interesados en el proceso de negocio *Registrar Siniestro* se verán afectados. Con esta información, el arquitecto puede analizar si el aporte que brinda la aplicación de esta política en la arquitectura es justificable frente al esfuerzo requerido en su implementación y al impacto negativo que supone su implementación en los ecosistemas afectados, en este caso frente al incremento en el tipo de respuesta percibido por el consumidor *Agent* en la ejecución de las actividades de Consultar Accidentes y Registrar Proveedores.En este ejemplo, el arquitecto encargado de definir las estrategias de gobierno de una arquitectura SOA, desea identificar los elementos que se ven involucrados en la implementación de esta política y obtener información que le permita al arquitecto deducir el esfuerzo requerido en la implementación de esta política y posibles impactos negativos que puedan generarse tras su implementación. Cabe recalcar que el ejemplo aplicado para ilustrar nuestra metodología, es una versión simplificada de una SOA. En la práctica, las arquitecturas de solución SOA, apoyan procesos de negocio complejos y están compuestas de gran cantidad de servicios que son ejecutados a través de múltiples ecosistemas. Es en estos casos donde esta propuesta apoya de manera significativa la toma de decisiones en el gobierno de una arquitectura SOA.

## VI. CONCLUSIONES

Las organizaciones interesadas en la implementación de arquitecturas orientadas a servicios, son cada vez más conscientes de que el modelo de gobierno SOA es uno de los factores principales que determina el éxito o fracaso en su arquitectura SOA. Si no se cuenta con un modelo de gobierno SOA adecuado, la empresa puede no sólo perder la oportunidad de obtener todos los beneficios que este paradigma ofrece, sino que puede enfrentarse a dificultades en la ejecución de sus procesos de negocio. Debido a la interoperabilidad entre los sistemas y la alta reutilización de los servicios que ofrece SOA, una decisión que modifique el portafolio de servicios o los ecosistemas SOA tiene un impacto potencial en muchos segmentos de negocio. Cuando este impacto no es evaluado previamente, se pueden presentar consecuencias que pueden ir desde la interrupción de los procesos de negocio hasta el abandono en la implementación de la arquitectura SOA por pérdida de credibilidad en el paradigma. Sin embargo, aún no se cuentan con estándares ni herramientas aceptadas comúnmente por la industria para apoyar estos procesos de gobierno a nivel de todo el ciclo de vida de la arquitectura.

En este trabajo se presentó una estrategia que apoya la gobernabilidad de las arquitecturas SOA, a través de la definición y evaluación de políticas de gobierno. Esta evaluación determina si los elementos que conforman la arquitectura, es decir, sus componentes, propiedades y relaciones, cumplen

las políticas definidas. Para realizar esta evaluación se utiliza una estrategia dirigida por modelos que permite describir una arquitectura de servicios y sus políticas de gobierno, y mediante transformaciones basadas en grafos se ejecuta la evaluación. El resultado de la evaluación se expresa mediante un modelo, que ofrece información detallada acerca de qué elementos cumplen o no las políticas evaluadas.

Aplicar una estrategia dirigida por modelos permite realizar esta evaluación a nivel de los modelos antes de iniciar la implementación de decisiones arquitecturales previniendo así los riesgos asociados a la decisión arquitectural en estudio. Adicionalmente, utilizar modelos permite la aplicación de este procedimiento de evaluación independientemente de la tecnología involucrada en la implementación de la arquitectura. Este proceso de evaluación debe ejecutarse antes de implementar una decisión arquitectural, para determinar si cumple con las políticas de gobierno de esta arquitectura y periódicamente para garantizar que la arquitectura se mantiene conforme a estas políticas.

## REFERENCIAS

[1] Universidad de Los Andes. MOdel Oriented Software ArchitectureS (MOOSAS).
[2] Michael Bell. *Executive's Guide Architecture to Service-Oriente*. John Wiley and Sons, Inc., 2006.
[3] Michael Bell. *Service Oriented Modeling: Service Analysis, Design and Architecture*. John Wiley and Sons, Inc., 2008.
[4] Biermann, Enrico and Ermel, Claudia and Taentzer, Gabriele. *Tiger EMF Model Transformation Framework EMT*. TU Berlin EMT Project Team, 1 edition, 2007.
[5] EbizQ. 2008 SOA Governance Survey Report. Executive Report, Oracle, 2008.
[6] Thomas Erl. *SOA Design Patterns*. 1 edition, 2009.
[7] Thomas Erl. *SOA: principles of service design*. 2008.
[8] Ermel, C. and Rudolf, M. and Taentzer, G. The AGG approach: language and environment. :551-603, 1999.
[9] OASIS. Reference Architecture for Service Oriented Architecture Version 1.0. Technical report, 2008.
[10] OASIS. Reference Model for Service Oriented Architecture. Technical report, 1, "OASIS Organization for the Advancement of Structured Information Standards", 2006.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663