AtOOmix: un Lenguaje Extensible de Reglas de Negocios

Sebastián Ortiz-Chamorro[1,2], Nathalie Aquino[1], Roberto Rubin[1,2], Luca Cernuzzi[1]

[1]DEI-Universidad Católica Nuestra Señora de la Asunción, Paraguay.
Campus Universitario, Barrio Santa Ana
{sortiz, naquino, rrubin, lcernuzz}@uca.edu.py
http://www.uc.edu.py
[2]LIFIA, Universidad Nacional de La Plata, Argentina.
http://lifia.info.unlp.edu.ar/

Resumen. La gran variedad existente de reglas de negocios ha llevado a la aparición de múltiples lenguajes y herramientas relacionadas. El desarrollo de un lenguaje general de reglas de negocios que integre las funcionalidades y características de los lenguajes existentes del área no es una tarea sencilla, conlleva varios problemas conceptuales y de integración y aparece hoy en día como una posibilidad lejana. Como alternativa a la posibilidad de crear un lenguaje único o general de reglas de negocios, hemos desarrollado AtOOmix, un lenguaje creado usando XML que ofrece como base una serie de tipos de datos y operaciones clásicas propias de muchos lenguajes de programación, ya implementadas en un intérprete. Como característica distintiva, AtOOmix provee además potentes mecanismos para crear nuevos lenguajes de reglas de negocios a través de la extensión y combinación de sus componentes.

Palabras Clave: Business Rules, End-User Development, Domain-Specific Languages, Object-Oriented Programming.

1 Introducción

Las reglas de negocios se refieren a las políticas, procedimientos y definiciones que gobiernan la manera en que una organización opera e interactúa con las personas y con otras organizaciones [20,19,23]. En este sentido, ubicamos a las reglas de negocios dentro del área de programación de usuario final (*end-user development*) [12,13].

Existe una gran diversidad de tipos de reglas de negocios, que pueden ser clasificadas en tres categorías fundamentales: reglas de integridad, de derivación y de reacción [23]. Esta diversidad ha motivado la aparición de varios lenguajes y herramientas relacionados a ellas, entre los que podemos citar a BRBeans [8], ILOG JRules [10], Jess [6] y RuleML [4].

Debido a la amplia variedad de lenguajes presentes en el área, se ha planteado la problemática que implicaría el desarrollo de un lenguaje que integre todas las funcionalidades y características de los distintos lenguajes de reglas de negocios existentes [23]. Ésta sería una tarea complicada de conceptualización e integración [23].

Por otro lado, la naturaleza amplia de las reglas de negocios hace necesaria, en algunos casos, la creación de formas de expresión específicas al problema que se intenta solucionar.

Creemos que para resolver este problema, antes que buscar soluciones generales, vale la pena desarrollar soluciones flexibles, con potentes mecanismos de adaptabilidad. A este efecto, hemos desarrollado un lenguaje de reglas de negocios básico, que provee un conjunto

de tipos de datos y de operaciones comunes en varios lenguajes de programación, pero que como característica distintiva, provee mecanismos para crear nuevos lenguajes a partir de la extensión y combinación de sus componentes. Nuestra hipótesis es que al problema de la diversidad de reglas de negocios es más fácil atacarlo con la utilización de herramientas flexibles que permitan crear rápidamente lenguajes de domino específico antes que intentar crear un lenguaje que contemple todos los casos posibles.

Si a esto añadimos el requerimiento de que el ambiente de desarrollo asociado deba ser utilizado por usuarios finales, la simple creación de lenguajes basados puramente en caracteres no es recomendable, puesto que fuerza al usuario a aprender "la obscura sintaxis y convenciones del vocabulario del lenguaje" [3].

En consecuencia, a lo largo de éste trabajo, asumiremos que los sistemas con los que estamos lidiando requieren la creación de un lenguaje de dominio específico y también la construcción de un ambiente de desarrollo con prestaciones avanzadas, posiblemente combinando elementos de programación gráfica o hipertextos. Excluimos los casos en los cuales se decide crear lenguajes de dominio específico basado puramente en caracteres.

AtOOmix fue creado usando XML. Las características claves por las que seleccionamos XML son: i) XML es extensible, pues permite definir marcas propias y estructurar la información de una manera apropiada a través de herramientas de fácil manejo y amplia utilización como el XML Schema [2,21]; ii) XML es una recomendación de la W3C ampliamente utilizada; iii) la amplia difusión de XML facilitaría la adopción de AtOOmix y el proceso de creación de lenguajes extendidos; iv) por último, los lenguajes de reglas de negocios en XML permiten que las reglas sean expresadas como unidades modulares e independientes y también permiten publicarlas y compartirlas entre diferentes sistemas. En un contexto más amplio, esto podría facilitar las interacciones comerciales, *business-to-customer* (B2C) y *business-to-business* (B2B) en la Web [23].

El resto del trabajo se estructura de la siguiente forma: en la sección 2 se presenta AtOOmix, poniendo particular énfasis en las extensiones que propone. Por motivos de espacio, se dará una explicación más detallada de las partes del lenguaje que consideramos más distintivas e innovadoras, como los mecanismos de extensión sintáctica. Para una definición detallada del lenguaje, su semántica y su implementación de referencia ver [1]. En la sección 3 se realiza una discusión acerca de AtOOmix y una comparación con otros trabajos relacionados. Finalmente, la sección 4 contiene las conclusiones y trabajos futuros.

2 AtOOmix

El diseño y desarrollo de AtOOmix en su forma básica se ha centrado en la creación e implementación de expresiones. Las expresiones son composiciones de operadores, funciones y/o métodos que se evalúan y devuelven algún valor; las mismas constituyen los bloques sintácticos básicos a partir de los cuales se construyen enunciados y, a veces, hasta programas completos [17]. Hemos restringido el desarrollo de AtOOmix puramente a un lenguaje de expresiones pues todos los lenguajes de reglas de negocios encontrados en la literatura especializada ([6,9,11,4] entre otros) utilizan expresiones, y éstas definen, en gran medida, sus funcionalidades, sintaxis y semántica.

Si bien algunos tipos de reglas de negocios se pueden expresar con AtOOmix en su forma básica, la intención es que el lenguaje sea utilizado como una plataforma de construcción de lenguajes de reglas de negocios. Lo importante en una plataforma de este estilo es brindar bloques de construcción que resulten útiles, así como una serie de mecanismos para extenderlos y combinarlos.

AtOOmix incorpora ciertas características de orientación a objetos y su intérprete ha sido implementado utilizando el lenguaje de programación Java [7].

2.1 Implementación y Modelo de Ejecución

Para posibilitar su utilización como un componente, el intérprete de AtOOmix provee una API (*Application Programming Interface*) sencilla que permite invocar las reglas de negocios en un punto específico de la aplicación principal en tiempo de ejecución. También se proveen mecanismos para validar AtOOmix en su forma básica y extendida, utilizando el XML Schema correspondiente. Es recomendado, pero no obligatorio que las reglas de negocios escritas en AtOOmix sean validadas antes de ser evaluadas.

AtOOmix es un lenguaje interpretado. Cada marca de primer nivel es procesada, una a la vez por el intérprete. Hay ciertas construcciones, como por ejemplo, las declaraciones de clases y de variables y los cambios de contextos, que alteran el estado del intérprete e influyen en la interpretación de las expresiones subsiguientes.

AtOOmix provee además de mecanismos de manejo de excepciones.

2.2 Sintaxis, Gramática y Semántica

AtOOmix ha sido creado usando XML. Por lo tanto, su sintaxis está sujeta a las reglas establecidas en la especificación de XML [5]. Su gramática ha sido definida utilizando XML Schema [2,21].

2.3 Tipos de Datos y Clases

Los tipos de datos primitivos de AtOOmix son `string`, `boolean` y `decimal`.

Los literales de los tipos de datos primitivos se representan con elementos XML cuyos nombres son `string`, `boolean` y `decimal`. Se tiene también un literal que representa el valor nulo. Los siguientes son ejemplos:

```
<string>AtOOmix</string>
<boolean>true</boolean>
<decimal>1.01</decimal>
<null/>
```

Por otra parte, AtOOmix provee mecanismos para definir nuevos tipos de datos y sus operadores asociados.

Para poder validar y luego ejecutar reglas que involucren objetos, AtOOmix requiere previamente de una asociación con un espacio de objetos. Esto se realiza a través de la declaración de clases. Es importante señalar que AtOOmix no provee mecanismos para la implementación de clases; la declaración de una clase en AtOOmix asocia la clase declarada con una clase en el espacio de objetos dado (brindado por la JVM en éste caso).

Ejemplos de declaración de clases:

```
<class type="AtOOmix.ejemplos.facultad.Persona" label="Persona">
     <cnstr>
            <arg name="nombre" type="string"/>
            <arg name="apellido" type="string"/>
     </cnstr>
     <mt name="imprimir"/>
</class>
```

```
<class type="AtOOmix.ejemplos.facultad.Carrera" label="Carrera">
     <cnstr>
            <arg name="nombre" type="string"/>
            <arg name="cantidadSemestres" type="decimal"/>
     </cnstr>
     <mt name="getSemestres" type="decimal"/>
</class>
```

Ejemplo de declaración de una variable e instanciación de una clase:

```
<let name="vPersona" type="Persona"/>
<set name="vPersona">
     <new type="AtOOmix.ejemplos.facultad.Alumno">
            <string>Gustavo Gabriel</string>
            <string>Frutos Pereira</string>
            <decimal>41188</decimal>
            <new type="Carrera">
                   <string>Ingeniería Informática</string>
                   <decimal>10</decimal>
            </new>
     </new>
</set>
```

Ejemplo de invocación de un método:

```
<send name="imprimir">
     <get name="vPersona"/>
</send>
```

Ejemplo de Casting:

```
<cast type="Alumno">
     <get name="vPersona"/>
</cast>
```

2.4 Variables

En AtOOmix, todas las variables deben ser declaradas antes de ser utilizadas. Luego se les pueden asignar tipos de datos compatibles y se las puede utilizar en expresiones. Ejemplos:

```
<let name="matricula" type="decimal"/>
<set name="matricula">
     <decimal>41180</decimal>
</set>
<get name="matricula"/>
```

2.5 Funciones

AtOOmix provee funciones aritméticas, lógicas, relacionales, de cadenas de caracteres, y un operador condicional `if`. Además AtOOmix provee extensibilidad de procedimientos [18,16] ya que permite que los usuarios definan funciones nuevas a partir de las predefinidas o a través de la programación. AtOOmix soporta la declaración y utilización de funciones sobrecargadas (con lo que se da soporte a la extensibilidad por sobrecarga de operadores [18]) y recursivas. Ejemplos de definición y llamado a funciones:

```
<fn name=”promedioAlumno” type="decimal">
     <arg name="totalNotas" type="decimal"/>
     <arg name="totalMaterias" type="decimal"/>
     <def>AtOOmix.ejemplos.facultad.Alumno.promedio</def>
</fn>
```

```
<call name="promedioAlumno">
     <decimal>241</decimal>
     <decimal>57</decimal>
</call>
```

2.6 Extensibilidad Sintáctica

Como una de las características centrales del lenguaje, AtOOmix incorpora un mecanismo de extensibilidad sintáctica que permite definir nuevos elementos XML y enlazarlos con clases que los interpreten.

El elemento `<xlang name="…" evaluator="…" ruleevaluator="…"/>` es utilizado para declarar un lenguaje XML con el que AtOOmix será extendido. El atributo `name` indica el nombre del elemento raíz del nuevo lenguaje XML. El atributo `evaluator` indica el nombre de la clase Java, pública, encargada de interpretar al nuevo lenguaje. Y el atributo `ruleevaluator` indica si el intérprete del nuevo lenguaje puede o no acceder al intérprete de AtOOmix.

La clase encargada de interpretar el nuevo lenguaje deberá implementar la interfaz `AtOOmix.evaluator.XEvaluator`, la cual define dos métodos: `getType` y `evaluate`.

El método `getType` retorna un `String` que representa el nombre del tipo de dato retornado luego de la evaluación del nuevo lenguaje; éste debe corresponder a una clase válida en AtOOmix.

El método `evaluate` es el encargado de evaluar el nuevo lenguaje. Cuando es llamado, se le pasa como parámetro el elemento raíz del nuevo lenguaje. Luego de la evaluación, el método debe retornar un objeto Java, de tipo `Object`, el cual contendrá el resultado de la evaluación. El objeto retornado debe ser una instancia de la clase (o de cualquiera de sus subclases) especificada en el método `getType`.

Si es que el nuevo lenguaje utiliza como componente a AtOOmix, deberá de alguna forma poder acceder a su intérprete. Para ello, el atributo `ruleevaluator` deberá contener el valor `yes`, de tal forma que a la hora de invocar al intérprete del nuevo lenguaje, el constructor de la clase del intérprete reciba como parámetro al objeto `RuleEvaluator`, el cual posee métodos para evaluar los elementos nativos de AtOOmix.

Para ejemplificar este mecanismo de extensibilidad, considérese que se ha definido un lenguaje XML para la representación de fechas, y que cada fecha tiene una estructura similar a la siguiente:

```
<fecha>
     <dia>...</dia>
     <mes>...</mes>
     <anho>...</anho>
</fecha>
```

Considérese también que `AtOOmix.XLang.FechaEvaluator` es la clase encargada de interpretar el lenguaje XML de representación de fechas, y que el resultado de evaluar un elemento `<fecha>` es un objeto del tipo `AtOOmix.ejemplos.fecha.Fecha`.

Luego, en AtOOmix se puede tener lo siguiente:

```
<let name="vDia" type="decimal"/>
<set name="vDia">
     <decimal>3</decimal>
</set>
<let name="vMes" type="decimal"/>
<set name="vMes">
     <decimal>12</decimal>
</set>
```

```
<let name=”vAnho” type=”decimal”/>
<set name=”vAnho”>
     <decimal>1978</decimal>
</set>
<class type=”AtOOmix.ejemplos.fecha.Fecha” label=”fecha”/>
<xlang name="fecha" evaluator="AtOOmix.XLang.FechaEvaluator"
     ruleevaluator="yes"/>
<let name="vFecha2" type="fecha"/>
<set name="vFecha2">
     <fecha>
            <dia><get name="vDia"/></dia>
            <mes><get name="vMes"/></mes>
            <anho><get name="vAnho"/></anho>
     </fecha>
</set>
```

En el ejemplo, la parte que ilustra la extensibilidad sintáctica de AtOOmix, se ve en el elemento `<fecha>` que está como contenido del elemento `<set>`, en el cual, a una variable se le asigna el resultado de la evaluación del elemento `<fecha>`.

Por otra parte, el lenguaje de representación de fechas ha precisado de expresiones básicas (expresiones que no utilizan extensiones sintácticas) para definir los valores correspondientes al día, mes y año. En el ejemplo se puede ver que los elementos `<dia>`, `<mes>` y `<anho>` tienen como contenido expresiones de AtOOmix. Esto ilustra la forma en que los lenguajes extendidos pueden utilizar a su vez componentes básicos de AtOOmix para evaluar sub-expresiones.

En el ejemplo, `<get name="vDia"/>` es una expresión directamente evaluada por el intérprete de AtOOmix, y su resultado (el objeto de tipo `decimal`) es retornado al evaluador de `<fecha>` para su utilización.

3 Discusión acerca de AtOOmix

AtOOmix está basado en XML y ofrece cuatro tipos de extensibilidad: i) de tipos de datos [18,16]; ii) de procedimientos [18,16], iii) de sobrecarga de operadores [18], y iv) sintáctica.

La extensibilidad de tipos de datos, en combinación con la extensibilidad sintáctica, constituyen el principal aporte de AtOOmix, pues brindan un mecanismo elegante y general de representación de expresiones para reglas de negocios a medida, específicas para dominios determinados de problemas que las requieran.

La especificación detallada de AtOOmix así como su implementación de referencia, que ha sido programada en Java, proveen una sólida definición del comportamiento del lenguaje [1].

Las expresiones del lenguaje AtOOmix son suficientes para representar una amplia e importante gama de reglas de negocios. La Tabla 1 muestra ejemplos de reglas de integridad y de cálculo con sus representaciones en AtOOmix.

La representación de expresiones complejas en XML puede llegar a ser larga, un poco incómoda y confusa a simple vista. Por otra parte, AtOOmix no es un lenguaje parecido al lenguaje natural y tampoco es declarativo, aunque varios autores insisten en que los lenguajes de reglas de negocios, idealmente, deben tener esas características [14,19,22,23]. Cabe recordar que estamos trabajando bajo la hipótesis de que el lenguaje será presentado al usuario por medio de un ambiente de desarrollo con prestaciones de interfaz avanzadas.

Por otra parte, las reglas de derivación que utilizan mecanismos de inferencia y las reglas de reacción no tienen una representación directa en AtOOmix, lo cual constituye una limitación.

Tabla 1. Ejemplos de diferentes tipos de reglas y sus representaciones en AtOOmix

Regla		**Enunciado Textual**	**Representación en AtOOmix**

Integridad		Un cliente de la compañía de alquiler de autos NM debe tener al menos 25 años	`<let name="tieneEdadRequerida" type="boolean"/>`
			`<set name="tieneEdadRequerida">`
			`  <call name="geq">`
			`  <get name="edadCliente"/>`
			`  <decimal>25</decimal>`
			`  </call>`
			`</set>`

Derivación	*Cálculo*	El precio neto de un producto es calculado como sigue: precio del producto * (1 + (porcentaje de impuesto / 100))	`<let name="precio_neto" type="decimal"/>`
			`<set name="precio_neto">`
			`  <call name="times">`
			`  <get name="precio_producto"/>`
			`  <call name="plus">`
			`  <decimal>1</decimal>`
			`  <call name="divide">`
			`  <get name="porc_impuesto"/>`
			`  <decimal>100</decimal>`
			`  </call>`
			`  </call>`
			`  </call>`
			`</set>`

Los conceptos desarrollados en este trabajo han sido aplicados en dos casos de validación: el lenguaje de representación de fechas, que se ha mostrado resumidamente en la sección anterior, y como solución alternativa a la implementada en el sistema Benefit Catalog / Configurator, un sistema de gran envergadura y complejidad.

El Benefit Catalog / Configurator es un sistema que permite definir planes de cobertura de seguros médicos. Fue implementado con éxito utilizando el lenguaje Cytera.Rules [15], para una compañía de seguros médicos norteamericana. Cytera.Rules es un lenguaje XML de reglas de negocios, predecesor de AtOOmix y que, a diferencia de éste último, no provee mecanismos de extensión adecuados a las necesidades del sistema mencionado. Los requerimientos de este sistema han sobrepasado la posibilidad de expresión y resolución de fórmulas lógicas y matemáticas que provee Cytera.Rules en su versión original, por lo que el código fuente de su intérprete tuvo que ser modificado para añadir los nuevos tipos de datos que fueron necesarios, con sus operadores y elementos XML asociados. Como parte de este trabajo se ha hecho una propuesta de implementación del Benefit Catalog / Configurator utilizando AtOOmix[1]. La extensibilidad sintáctica y de tipos de datos que provee AtOOmix permite que dicha implementación pueda ser realizada de manera más sencilla y ordenada. La idea básica de la solución consiste en declarar los nuevos lenguajes extendidos que se requieren, junto con sus nuevos tipos de datos correspondientes y las operaciones que resulten necesarias sobre esos nuevos tipos de datos. Con AtOOmix, es posible realizar todo esto sin necesidad de modificar el código fuente del intérprete.

En el ejemplo del lenguaje de representación de fechas, y en el sistema Benefit Catalog / Configurator, se ha podido apreciar la necesidad de creación de estructuras sintácticas a medida en XML, que representan datos y reglas de negocios en dominios específicos. Esas estructuras, que están relacionadas a nuevos tipos de datos con sus operaciones asociadas y que son utilizadas en expresiones, han sido desarrolladas eficazmente en AtOOmix.

[1] El sistema Benefit Catalog / Configurator, la descripción de su implementación con Cytera.Rules, y la propuesta de solución con AtOOmix se presentan detalladamente en "*AtOOmix, un Lenguaje XML Extensible de Reglas de Negocios con Aspectos de Orientación a Objetos*" [1]

3.1 Comparaciones con otros Lenguajes y Herramientas de Reglas de Negocios

Realizamos una comparación con BRBeans [8], ILOG JRules [10], Jess [6] y RuleML [4].

Según la clasificación de reglas de negocios de Wagner [23], con AtOOmix en su forma básica se pueden representar las reglas de derivación que utilizan cálculos matemáticos y/o lógicos para la adquisición de nuevos conocimientos y las restricciones de integridad que utilizan expresiones booleanas. Por su parte, Jess soporta reglas de producción y de derivación. IRL (ILOG Rule Language) [9] permite representar reglas de producción. RuleML utiliza una jerarquía en la que aparecen los tres tipos de reglas (integridad, derivación y reacción), y también *hechos*; pero sobre todo centran sus esfuerzos en las reglas de derivación y en los *hechos*. Sin embargo, es importante señalar que AtOOmix provee de una extensibilidad sintáctica que ninguna de estas otras soluciones posee.

AtOOmix no utiliza mecanismos de inferencia como los de JRules y Jess. Sin embargo, éstos mecanismos no son adecuados en ciertas situaciones, como cuando se tiene una secuencia predeterminada de reglas y hay que maximizar la velocidad de ejecución [10].

Al igual que AtOOmix, BRBeans no utiliza ningún tipo de mecanismo de inferencia; su idea básica consiste en identificar puntos potencialmente variables en el código de las aplicaciones (*trigger points*), a partir de los cuales se invocan métodos que ejecutan reglas específicas. Al terminar la ejecución de las reglas se retorna al punto de invocación del método. AtOOmix puede ser utilizado de esta forma, con la ventaja de que en un *trigger point* particular puede invocar código escrito en un lenguaje de dominio específico.

Para la aplicación de las reglas de negocios expresadas en RuleML existen implementaciones que traducen las reglas, usando XSLT, a código de Jess o de Prolog. Mandarax RuleML provee un ambiente para la entrada, procesamiento y salida de reglas de RuleML.

4 Conclusiones y Trabajos Futuros

Existen dominios en los cuales los lenguajes de reglas de negocios existentes sencillamente no sirven para expresar adecuadamente algunos conceptos. En estos casos, puede resultar necesaria la creación de lenguajes de dominio específico. Si a esto añadimos la necesidad de crear interfaces con prestaciones avanzadas como la utilización de componentes gráficos o de hipertextos, nos encontramos frente a situaciones que requieren la construcción de dos componentes complejos, fuertemente interrelacionados y que conllevan un esfuerzo de desarrollo considerable.

En éstos casos existen varios caminos en materia de diseño e implementación. Se podría por ejemplo, crear un lenguaje de dominio específico totalmente nuevo en XML, construir un intérprete para el mismo y un ambiente de desarrollo utilizando Java o algún otro lenguaje de propósito general.

Sin embargo, estos lenguajes de dominio específico (así como también los de propósito general), en la gran mayoría de los casos, utilizan un conjunto de tipos de datos y de operaciones bien conocido que suelen incluir cadenas de caracteres, tipos de datos numéricos, booleanos y las operaciones básicas asociadas a los mismos. Resulta innecesario tener que implementar éstos y otros componentes linguísticos comunes cada vez que se tiene que implementar un lenguaje de dominio específico. Ésta propuesta apunta a proveer de una plataforma sobre la cual se pueden construir lenguajes de dominio especifico teniendo: i) una base sintáctica ya desarrollada; ii) el diseño e implementación de un intérprete de lenguajes extendidos en base a Atoomix; iii) un conjunto de tipos de datos y operaciones ya

implementadas; iv) un conjunto de mecanismos de extensión; y v) mecanismos elegantes para asociar las extensiones del lenguaje con el código que las implemente.

Otra alternativa sería la de modificar directamente los lenguajes existentes y el código fuente de los intérpretes. Creemos que éste no es un camino adecuado desde el punto de vista de ingeniería de software ni posible en muchos casos puesto que el código fuente del intérprete bien puede ser propietario.

Dicho de otra forma, si bien se podría crear un lenguaje de dominio específico completamente nuevo (o modificar uno existente) en cada ocasión, la utilización de AtOOmix conllevaría como ventajas: i) un sustancial ahorro en el trabajo de desarrollo (no se tienen que volver a implementar los tipos de datos básicos); ii) un ahorro en materia de diseño del intérprete puesto que ya se tiene un intérprete extensible diseñado que posee elegantes mecanismos de extensión (no es necesario cambiar el código fuente del mismo); iii) ventajas de mantenimiento del lenguaje y el intérprete, pues resultaría más fácil realizar nuevas extensiones a medida que sea necesario utilizando los mecanismos proveídos o corregir la implementación de las extensiones del lenguaje (se encontrarían convenientemente separadas del código fuente del resto del intérprete); y iv) la mayor ventaja la podría dar el hecho de que con los mecanismos de extensión desarrollados, no sólo se puede utilizar la base de tipos de datos y operaciones proveídas por AtOOmix sino que se pueden combinar también diferentes extensiones del lenguaje ya implementadas a modo de librerías (lo cual resulta plausible en el ejemplo del lenguaje de representación de fechas).

En este trabajo, se explora el papel que la extensibilidad podría jugar en la solución del problema de la gran diversidad de reglas de negocios que se deben poder representar. Para el efecto se propone AtOOmix, un lenguaje extensible de reglas de negocios basado en XML, con su respectivo intérprete. AtOOmix ofrece cuatro tipos de extensibilidad: i) de tipos de datos; ii) de procedimientos, iii) de sobrecarga de operadores, y iv) sintáctica.

En particular, la extensibilidad de tipos de datos en combinación con la extensibilidad sintáctica constituyen un mecanismo elegante y general para la representación de expresiones para reglas de negocios a medida, específicas para dominios determinados de problemas que las requieran. Estos mecanismos también constituyen la principal diferencia con respecto a los lenguajes de reglas de negocios con los cuales se realizó una comparación.

Los conceptos desarrollados en este trabajo, además de haber sido aplicados en el ejemplo presentado resumidamente en la sección 2, han sido aplicados en el sistema Benefit Catalog / Configurator, un sistema que permite definir planes de cobertura de seguros médicos para una compañía de seguros médicos norteamericana. En estos ejemplos se puede notar cómo los mecanismos de extensibilidad introducidos permiten una ampliación elegante y controlada del lenguaje a nivel sintáctico que es acompañada por una ampliación, también elegante y controlada, de la funcionalidad del intérprete.

Cabe destacar que las expresiones del lenguaje AtOOmix son suficientes para representar una amplia e importante gama de reglas de negocios y que su representación en AtOOmix está pensada para ser utilizada en forma directa por programas.

A partir de estas consideraciones, se abren varias posibles líneas de investigación futuras. Por un lado y con el fin de facilitar la utilización de AtOOmix por un amplio espectro de usuarios, se requeriría de interfaces gráficas para la edición y manipulación de las reglas ofreciendo una representación visible para los usuarios expertos de un dominio y típicamente sin conocimientos de programación.

Referencias

1. Aquino, N.: AtOOmix, un Lenguaje XML Extensible de Reglas de Negocios con Aspectos de Orientación a Objetos. Universidad Católica Nuestra Señora de la Asunción (2006)
2. Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes, Second Edition, W3C Recommendation (2004), http://www.w3.org/TR/xmlschema-2
3. Cypher, A.: Watch What I Do: Programming by Demonstration. MIT Press, Cambridge, Massachusetts (1993).
4. Boley, H., Tabet, S., and Wagner, G.: Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium* (SWWS'01), pages 381-401. Standford University, July/August (2001)
5. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0, Third Edition, W3C Recommendation (2004), http://www.w3.org/TR/2004/REC-xml-20040204
6. Friedman-Hill, E.J.: Jess, The Expert System Shell for the Java Platform. Sandia National Laboratories (2003), http://herzberg.ca.sandia.gov/jess
7. Gosling, J., Joy, BB., Steele, G.: The Java Language Specification. Addison-Wesley, Reading, Massachusetts (1996).
8. IBM Corporation: Business Rules Beans (BRBeans) (2001) http://www.research.ibm.com/AEM/brb.html
9. ILOG JRules 4.0. Rule Language Reference Manual (2002), http:\\www.ilog.com
10. ILOG JRules, White Paper (2001), http:\\www.ilog.com
11. International Organization for Standardization: ISO/IEC 9075-2:1999 Information technology -- Database languages -- SQL -- Part 2: Foundation (SQL/Foundation) (1999)
12. Lieberman, H., Paterno, F., Wulf, V. (eds.): End User Development. Human-Computer Interaction Series, vol. 9, Springer, Heidelberg (2006).
13. Martin, J., Odell, J.J.: Métodos Orientados a Objetos. Conceptos Fundamentales. Prentice-Hall (1997)
14. Martin, J.: An Information Systems Manifesto. Englewood Cliffs, New Jersey: Prentice-Hall (1984)
15. Ortiz, S.: Cytera.Rules Language Specification. Technical Report, CyteraSystems (2001)
16. Poial, J.: Remarks on Language Extensibility. Proceedings of the Fourth Symposium on Programming Languages and Software Tools, Hungary (1995)
17. Pratt, T.W.; Zelkowitz, M.V.: Lenguajes de Programación: Diseño e Implementación, Tercera Edición, Prentice Hall (1998)
18. Standish, T.A.: Extensibility in Language Design. ACM SIGPlan Notices, pp. 18-21 (1975)
19. Taveter, K., Wagner, G.: Agent-Oriented Enterprise Modeling Based on Business Rules. Proceedings of 20th Int. Conf. On Conceptual Modeling (ER2001), pp. 527–540, Japón, (2001)
20. The Business Rules Group.: Defining business rules – what are they really?, Technical Report 1.3 (2000), http://businessrulesgroup.org/first_paper/br01c0.htm
21. Thompson, HH.S. et al.: XML Schema Part 1: Structures, Second Edition, W3C Recommendation (2004), http://www.w3.org/TR/xmlschema-1
22. Von Halle, B.: Business Rules Applied, John Wiley & Sons (2001)
23. Wagner, G.: How to Design a General Rule Markup Language, Proceedings of XSW2002, Berlin, Germany (2002)